Handling Errors and Messages in Applications

Source	http://msdn2.microsoft.com/en-us/library/ms189583.aspx

Errors raised either by the SQL Server Database Engine or the RAISERROR statements are not part of a result set. Errors are returned to applications through an error-handling mechanism that is separate from the processing of result sets.

Each database application programming interface (API) has a set of functions, interfaces, methods, objects, or structures through which they return errors and messages. Each API function or method typically returns a status code indicating the success of that operation. If the status is anything other than success, the application can call the error functions, methods, or objects to retrieve the error information.

The Database Engine can return information to the caller in one of two ways:

- Errors
  - The errors from sys.messages with a severity of 11 or higher.
  - Any RAISERROR statement with a severity of 11 or higher.
- Messages
  - The output of the PRINT statement.
  - The output of several DBCC statements.
  - The errors from sys.messages with a severity of 10 or lower.
  - Any RAISERROR statement with a severity of 10 or lower.

Applications using APIs such as ActiveX Data Object (ADO) and OLE DB cannot generally distinguish between errors and messages. In Open Database Connectivity (ODBC) applications, messages generate a SQL_SUCCESS_WITH_INFO function return code, and errors usually generate a SQL_ERROR return code. The difference is most pronounced in DB-Library, in which errors are returned to the application error-handler function, and messages are returned to the application message-handler function. Similarly, when using the SqlClient provider, errors cause the SqlException exception to be thrown; messages do not alter control flow and can be intercepted by application code by registering a callback for the InfoMessage event handler.

Other components can also raise errors:

- **The OLE DB for SQL Server provider and SQL Server ODBC driver raise errors of their own. The format of these errors is consistent with the formats defined in the API specifications.**
- **The Net-Libraries raise errors of their own.**
- **The Extended Stored Procedure API raises errors in its own format.**
- **The SQL Server wizards, applications, and utilities such as, the SQL Server Management Studio and the sqlcmd utility, can raise their own errors.**

The errors from these components are returned to the calling application using the same basic mechanism as errors from the Database Engine. An application can process these errors using the same error-handling logic as is used for Database Engine errors. Because these errors are raised outside of the Database Engine, they cannot be processed in Transact-SQL TRY…CATCH constructs.

ODBC Error Handling

The ODBC specification introduced an error model that has served as the foundation of the error models of the generic database APIs, such as ADO and OLE DB, and the APIs built over ODBC—RDO, Data Access Object (DAO), and the Microsoft Foundation Classes (MFC) Database Classes. This also applies to the SQL Native Client ODBC driver. In the ODBC model, errors have the following attributes:

- **SQLSTATE**
  The SQLSTATE is a five-character error code defined originally in the ODBC specification. SQLSTATE codes are common across all ODBC drivers and provide a way for applications to code basic error handling without testing for all of the different error codes returned by various databases. The ODBC SQLSTATE has nothing to do with the state attribute of Database Engine error messages.
  ODBC 2.*x* returns one set of SQLSTATE codes, and ODBC 3.*x* returns a set of SQLSTATE codes aligned with the X/Open Data Management: Structured Query Language (SQL), version 2 standard. Because all ODBC drivers return the same sets of SQLSTATE codes, applications basing their error handling on SQLSTATE codes are more portable.
- **Native error number**
  The native error number is the error number from the underlying database. ODBC applications receive the Database Engine error numbers as native error numbers.
- **Error message string**
  The error message is returned in the error message string parameter.

When an ODBC function returns a status other than SQL_SUCCESS, the application can call SQLGetDiagRec to get the error information. For example, if an ODBC application gets a syntax error (SQL Server error number 170), SQLGetDiagRec returns the following.

```
szSqlState = 42000, pfNative = 170

szErrorMsg =

'[Microsoft][ODBC SQL Server Driver][SQL Server]

                                        Line 1: Incorrect syntax near *'
```

The ODBC SQLGetDiagField function allows ODBC drivers to specify driver-specific diagnostic fields in the diagnostic records returned by the driver. The SQL Server ODBC driver specifies driver-specific fields to hold Database Engine error information, such as the Database Engine severity and state codes.

ADO Error Handling

ADO uses an errors object and errors collection to return standard error information such as SQLSTATE, native error number, and the error message string. These are the same as their ODBC counterparts. ADO does not support any provider-specific error interfaces; Database Engine-specific error information, such as the severity or state, is not available to ADO applications.

OLE DB Error Handling

OLE DB uses the IErrorInfo interface to return standard error information, such as the SQLSTATE, native error number, and error string. These are the same as their ODBC counterparts. The OLE DB Provider for SQL Server defines an ISQLServerErrorInfo interface to return Database Engine-specific information, such as the severity, state, procedure name, and line number.

SqlClient Error Handling

The SqlClient managed provider throws an SqlException exception when an unhandled error is raised by the SQL Server Database Engine. Through the SqlException class, applications can retrieve information about errors produced on the server side, including error number, error message, error severity, and other exception context information.

For processing warnings or informational messages sent by the SQL Server Database Engine, applications can create a SqlInfoMessageEventHandler delegate to listen for the

InfoMessage event on the SqlConnection class. Similar to the exception case, message context information such as severity and state are passed as arguments to the callback.

~~~ End of Article ~~~