Datatypes and the Default Mask with Dynamic Data Masking

In this article we will examine the default data mask available and how it interacts with the various data types in SQL Server.

The Masks

There are four different types of masks available in SQL Server 2016. Each of these is designed to work with certain types of data.

The four types of masks are:

- Default
- Email
- Custom String
- Random

Default

The default mask works with many datatypes. When the default mask is used, the value returned depends on the datatype. This means that a user querying the column, with knowledge of DDM, will know what datatype the underlying column contains. This is a bit of a security weakness.

The default mask values returned are:

- 4 Xs if the type is a string type and the length is longer than 3 characters. Fewer Xs if the size is 3 or smaller.
- 0 for the numeric data types.
- Default date of 1900-01-01 for date datatypes.

Let's create a table for character data.

```
--  Now create our character table
  CREATE TABLE CharTest (
    Testchar1 CHAR(1) MASKED WITH (FUNCTION='default()') DEFAULT ('a')
  , Testchar2 CHAR(2) MASKED WITH (FUNCTION='default()') DEFAULT ('aa')
  , Testchar3 CHAR(3) MASKED WITH (FUNCTION='default()') DEFAULT ('aaa')
  , Testchar4 CHAR(4) MASKED WITH (FUNCTION='default()') DEFAULT ('aaaa')
  , Testchar5 CHAR(5) MASKED WITH (FUNCTION='default()') DEFAULT ('aaaaa')
  , Testchar10 CHAR(10) MASKED WITH (FUNCTION='default()') DEFAULT ('aaaaaaaaaa')
  , Testnchar1 NCHAR(1) MASKED WITH (FUNCTION='default()') DEFAULT ('a')
  , Testnchar2 NCHAR(2) MASKED WITH (FUNCTION='default()') DEFAULT ('aa')
  , Testnchar3 NCHAR(3) MASKED WITH (FUNCTION='default()') DEFAULT ('aaa')
```

```
  , Testnchar4 NCHAR(4) MASKED WITH (FUNCTION='default()') DEFAULT ('aaaa')
  , Testnchar5 NCHAR(5) MASKED WITH (FUNCTION='default()') DEFAULT ('aaaaa')
  , Testnchar10 NCHAR(10) MASKED WITH (FUNCTION='default()') DEFAULT ('aaaaaaaaaa')
  , Testvarchar varchar(2000) MASKED WITH (FUNCTION='default()') DEFAULT
('aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa')
  , Testnvarchar nvarchar(2000) MASKED WITH (FUNCTION='default()') DEFAULT
('aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa')
  , Testtext TEXT MASKED WITH (FUNCTION='default()') DEFAULT ('1234567890')
  , Testntext NTEXT MASKED WITH (FUNCTION='default()') DEFAULT ('1234567890')
  )
GO
-- insert data
INSERT dbo.CharTest
  DEFAULT VALUES
INSERT dbo.CharTest
  VALUES ('1', '12', '123', '1234', '12345', '1234567890', '0', '21', '321', '4321',
'54321', '9876543210'
         , 'qwertyuiopasdfghjkl', 'qwertyuiopasdfghjkl', 'qwertyuiopasdfghjkl',
'qwertyuiopasdfghjkl')
```

If we query this data as a non-privileged user, we find these results:

```
CREATE USER ddmtest WITHOUT LOGIN
GRANT SELECT ON [dbo].[CharTest] TO ddmtest
GO

-- Query as the test user
EXECUTE AS USER = 'ddmtest'
GO
SELECT
       *
FROM
       dbo.CharTest
GO
REVERT
GO
```

Results | Messages

	Testchar1	Testchar2	Testchar3	Testchar4	Testchar5	Testchar10	Testnchar1	Testnchar2	Testnchar3	Testnchar4	Testnchar5	Testnchar10	Testvarchar	Testnvarchar	Testtext	Testntext
1	x	xx	xxx	xxxx	xxxx	xxxx	x	xx	xxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
2	x	xx	xxx	xxxx	xxxx	xxxx	x	xx	xxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx

These results are what we expect, based on the description. The number of Xs corresponds to the size of the string, with a max of 4.

Now let's look at numeric types. We see something similar if we create a table with the various types, masked with default. Here's the table:

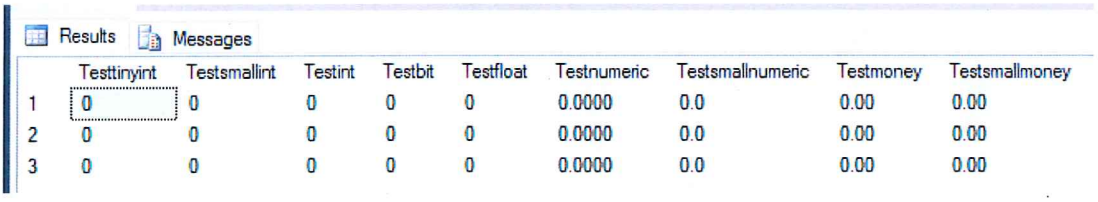
```
CREATE TABLE NumbersTest(
  Testtinyint TINYINT MASKED WITH (FUNCTION='default()') DEFAULT (0)
, Testsmallint SMALLINT MASKED WITH (FUNCTION='default()') DEFAULT 1
, Testint INT MASKED WITH (FUNCTION='default()') DEFAULT 0
, Testbit BIT MASKED WITH (FUNCTION='default()') DEFAULT 0
, Testfloat FLOAT MASKED WITH (FUNCTION='default()') DEFAULT (0.0)
, Testnumeric NUMERIC(10, 4) MASKED WITH (FUNCTION='default()') DEFAULT (0.0)
, Testsmallnumeric NUMERIC(2,1) MASKED WITH (FUNCTION='default()') DEFAULT (0.0)
, Testmoney MONEY MASKED WITH (FUNCTION='default()') DEFAULT 0.0
, Testsmallmoney SMALLMONEY MASKED WITH (FUNCTION='default()') DEFAULT 0.0
)
```

```
GO
-- insert data
INSERT dbo.NumbersTest
  DEFAULT VALUES
INSERT dbo.NumbersTest
  VALUES
    (12, 13, 14, 1, 10.0, 11.3, 3.3, 10.0, 5.5)
    , (127, 255, 14000000, 0, 187756560.433, 11644.3333, 2.4, 5000.50, 25.25)
GO
```

If we query this data as a non-privileged user, we find these results:

```
GRANT SELECT ON [dbo].[NumbersTest] TO ddmtest
GO

-- Query as the test user
EXECUTE AS USER = 'ddmtest'
GO
SELECT
       *
FROM
       dbo.NumbersTest
GO
REVERT
GO
```



Results | Messages

	Testtinyint	Testsmallint	Testint	Testbit	Testfloat	Testnumeric	Testsmallnumeric	Testmoney	Testsmallmoney
1	0	0	0	0	0	0.0000	0.0	0.00	0.00
2	0	0	0	0	0	0.0000	0.0	0.00	0.00
3	0	0	0	0	0	0.0000	0.0	0.00	0.00

This is what we expect. The numeric columns are masked with zeros.

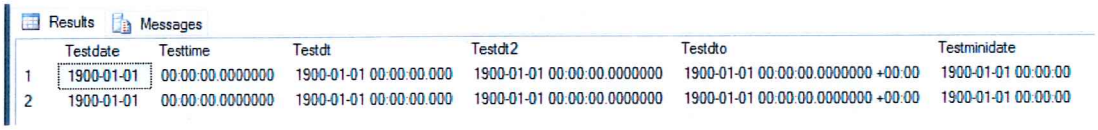
Now let's look at date types.

```
-- check dates
CREATE TABLE DateTest (
  Testdate DATE MASKED WITH (FUNCTION='default()') DEFAULT GETDATE()
, Testtime TIME MASKED WITH (FUNCTION='default()') DEFAULT GETDATE()
, Testdt DATETIME MASKED WITH (FUNCTION='default()') DEFAULT GETDATE()
, Testdt2 DATETIME2 MASKED WITH (FUNCTION='default()') DEFAULT GETDATE()
, Testdto DATETIMEOFFSET MASKED WITH (FUNCTION='default()') DEFAULT GETDATE()
, Testminidate SMALLDATETIME MASKED WITH (FUNCTION='default()') DEFAULT GETDATE()
)
GO
INSERT dbo.DateTest
  DEFAULT VALUES;
INSERT dbo.DateTest
  VALUES
    ('20160101', '10:00', '20160102', '20160103', '20160104', '20160105')
```

If we query this data as a non-privileged user, we find these results:

```
-- Grant rights
```

```
GRANT SELECT ON dbo.DateTest TO ddmtest;
GO
-- query the table
EXECUTE AS USER = 'ddmtest';
GO
SELECT *
 FROM dbo.DateTest
 GO
 REVERT
 GO
```



Results Messages

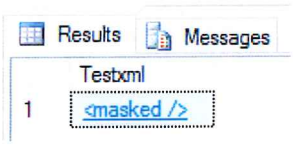
	Testdate	Testtime	Testdt	Testdt2	Testdto	Testminidate
1	1900-01-01	00:00:00.0000000	1900-01-01 00:00:00.000	1900-01-01 00:00:00.0000000	1900-01-01 00:00:00.0000000 +00:00	1900-01-01 00:00:00
2	1900-01-01	00:00:00.0000000	1900-01-01 00:00:00.000	1900-01-01 00:00:00.0000000	1900-01-01 00:00:00.0000000 +00:00	1900-01-01 00:00:00

What about the special data types? Let's try them and see.

```
--What about special data types?
-- XML
CREATE TABLE XMLTest(
 Testxml XML MASKED WITH (FUNCTION='default()')
 )
 GO
INSERT dbo.XMLTest
  VALUES
    ('<root><Orders><Order>11</Order></Orders></root>')
GO
```

If we query this data as a non-privileged user, we find these results:

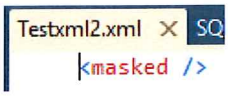
```
-- Grant rights
GRANT SELECT ON dbo.XMLTest TO ddmtest;
GO
-- query the table
EXECUTE AS USER = 'ddmtest';
GO
SELECT *
 FROM dbo.XMLTest
GO
REVERT
GO
```



Results Messages

	Testxml
1	<masked />

That works fine. The entire XML document is:



Testxml2.xml SQ

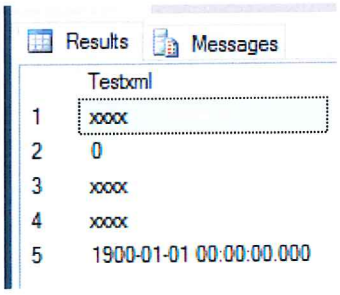
```
<masked />
```

Let's insert a variety of data:

```
-- SQLVariant
CREATE TABLE VariantTest(
 Testxml SQL_VARIANT MASKED WITH (FUNCTION='default()')
 )
 GO
INSERT dbo.VariantTest
  VALUES ('<root><Orders><Order>11</Order></Orders></root>')
INSERT dbo.VariantTest
  VALUES (21453)
INSERT dbo.VariantTest
  VALUES ('Z')
INSERT dbo.VariantTest
  VALUES ('A longer test')
INSERT dbo.VariantTest
  VALUES (DATETIME2FROMPARTS(2012, 02, 14, 8, 15, 22, 4, 7))
GO
```

If we query this data as a non-privileged user, we find these results:

```
-- Grant rights
GRANT SELECT ON dbo.VariantTest TO ddmtest;
GO
-- query the table
EXECUTE AS USER = 'ddmtest';
GO
SELECT *
 FROM dbo.VariantTest
GO
REVERT
GO
```



Now let's look at GUIDs.

```
-- GUID
CREATE TABLE GUIDTest(
 Testguid UNIQUEIDENTIFIER MASKED WITH (FUNCTION='default()')
 )
 GO
INSERT dbo.GUIDTest
  VALUES (NEWID())
GO
```

If we query this data as a non-privileged user, we find these results:

```
-- Grant rights
GRANT SELECT ON dbo.GUIDTest TO ddmtest;
GO
```

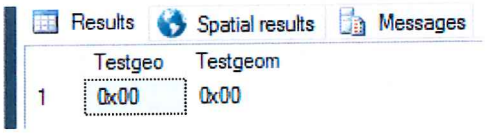
```
-- query the table
EXECUTE AS USER = 'ddmtest';
GO
SELECT *
 FROM dbo.GUIDTest
GO
REVERT
GO
```

Now let's look at spatial data.

```
-- Spatial
CREATE TABLE SpatialTypes (
  Testgeo GEOGRAPHY MASKED WITH (FUNCTION='default()')
, Testgeom GEOMETRY MASKED WITH (FUNCTION='default()')
);
GO
INSERT dbo.SpatialTypes
  VALUES
    (geography::STGeomFromText('LINESTRING(-122.360 47.656, -122.343 47.656)', 4326)
       , geometry::STGeomFromText('LINESTRING (100 100, 20 180, 180 180)', 0)
       )
GO
```

If we query this data as a non-privileged user, we find these results:

```
-- Grant rights
GRANT SELECT ON dbo.SpatialTypes TO ddmtest;
GO
-- query the table
EXECUTE AS USER = 'ddmtest';
GO
SELECT *
 FROM dbo.SpatialTypes
GO
REVERT
GO
```



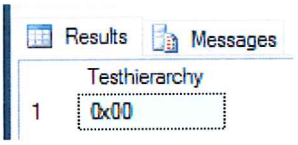
Now let's look at hierarchyID.

```
-- HierarchyID
CREATE TABLE HierarchyTest (
  Testhierarchy HIERARCHYID MASKED WITH (FUNCTION='default()')
);
GO
```

```
INSERT HierarchyTest
    VALUES(hierarchyid::GetRoot());
GO
```

If we query this data as a non-privileged user, we find these results:

```
-- Grant rights
GRANT SELECT ON dbo.HierarchyTest TO ddmtest;
GO
-- query the table
EXECUTE AS USER = 'ddmtest';
GO
SELECT *
 FROM dbo.HierarchyTest
GO
REVERT
GO
```



Microsoft appears to have considered the default for all types built into SQL Server. There are a few we didn't test here (real, for example), but I suspect they all behave in a similar manner.

Conclusion

The default data mask for DDM seems to be well thought out and covers all the data types most of you will use in your system. We have seen how the various results are affected by the datatype chosen by the column, with each result being what one might guess.

A malicious user will know roughly what type of data is stored in the columns and could tailor attacks for the data types.

Additional References

- https://msdn.microsoft.com/en-us/library/mt130841.aspx
- https://azure.microsoft.com/en-us/documentation/articles/sql-database-dynamic-data-masking-get-started/