# [MS-SSCLRT]:
# Microsoft SQL Server CLR Types Serialization Formats

**Intellectual Property Rights Notice for Open Specifications Documentation**

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft's Open Specification Promise (available here: http://www.microsoft.com/interop/osp) or the Community Promise (available here: http://www.microsoft.com/interop/cp/default.mspx). If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- **Fictitious Names.** The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

**Reservation of Rights.** All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

**Tools.** The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.



*Copyright © 2010 Microsoft Corporation.*

## Revision Summary

| Date | Revision History | Revision Class | Comments |
|---|---|---|---|
| 08/07/2009 | 0.1 | Major | First release. |
| 11/06/2009 | 0.1.1 | Editorial | Revised and edited the technical content. |
| 03/05/2010 | 0.2 | Minor | Updated the technical content. |
| 04/21/2010 | 1.0 | Major | Updated and revised the technical content. |
| 06/04/2010 | 1.0.1 | Editorial | Revised and edited the technical content. |
| 06/22/2010 | 2.0 | Major | Significantly changed the technical content. |
| 09/03/2010 | 3.0 | Major | Significantly changed the technical content. |



*Copyright © 2010 Microsoft Corporation.*

# Contents

**1 Introduction** ... **4**
1.1 Glossary ... 4
1.2 References ... 4
1.2.1 Normative References ... 4
1.2.2 Informative References ... 5
1.3 Structure Overview (Synopsis) ... 5
1.4 Relationship to Protocols and Other Structures ... 6
1.5 Applicability Statement ... 6
1.6 Versioning and Localization ... 6
1.7 Vendor-Extensible Fields ... 6

**2 Structures** ... **7**
2.1 System CLR Types Structure ... 7
2.1.1 GEOGRAPHY and GEOMETRY Structures ... 7
2.1.2 Basic GEOGRAPHY Structure (Version 1) ... 7
2.1.3 Basic GEOGRAPHY Structure (Version 2) ... 9
2.1.4 FIGURE Structure ... 12
2.1.5 SHAPE Structure ... 13
2.1.6 GEOGRAPHY POINT Structure ... 14
2.1.7 GEOMETRY POINT Structure ... 14
2.1.8 SEGMENT Structure ... 15
2.2 HIERARCHYID Structure ... 15
2.2.1 Logical Definition ... 15
2.2.2 Physical Representation ... 16
2.3 CLR UDTs ... 17
2.3.1 Native UDT Serialization ... 17
2.3.1.1 Binary Format of Each Byte ... 18
2.3.1.2 Binary Format of Primitive Types ... 18
2.3.1.3 Nested Structures ... 20
2.3.2 User-Defined UDT Serialization ... 20

**3 Structure Examples** ... **21**
3.1 GEOGRAPHY and GEOMETRY Structure Examples ... 21
3.1.1 Example of an Empty Point Structure ... 21
3.1.2 Example of a Geometry Point Structure ... 21
3.1.3 Example of a Linestring Structure ... 22
3.1.4 Example of a Geometry Collection Structure ... 23
3.1.5 Example of an Object Serialized in Version 2 ... 26
3.2 HIERARCHYID Examples ... 27
3.3 CLR UDT Serialization Example ... 28

**4 Security Considerations** ... **31**

**5 Appendix A: Product Behavior** ... **32**

**6 Change Tracking** ... **33**

**7 Index** ... **36**



*Copyright © 2010 Microsoft Corporation.*

# 1 Introduction

This document specifies the binary format of the **GEOGRAPHY**, **GEOMETRY**, **HIERARCHYID**, and **common language runtime (CLR) user-defined type (UDT)** structures managed by Microsoft® SQL Server® 2008 R2 and Microsoft® SQL Server® code-named Denali Community Technology Preview 1 (CTP1). Microsoft® SQL Server® provides the **geography**, **geometry**, and **hierarchyid** SQL Server data types as well as the CLR UDTs that use these structures.

The first two of these SQL Server types implement the OpenGIS Consortium's (OGC) Simple Feature Specification (SFS) [OGCSFS]. Thus, the content of these structures closely mirrors the SFS.

The structures that are used to transfer **geography** and **geometry** data types are identical. In this document, the term "**GEOGRAPHY** structure" refers to both the **GEOGRAPHY** and **GEOMETRY** structures, except where it is necessary to distinguish between the two structures. Likewise, "**geography** data type" refers to both the **geography** and **geometry** SQL Server data types.

CLR UDTs enable users to extend the SQL Server type system by creating new types. These types can include any fields and methods defined by the user. The exact structure depends on the user who is implementing CLR UDTs. The SQL Server client program must contain the knowledge of the internal structure of each CLR UDT before it can read that type's binary format.

## 1.1 Glossary

The following terms are defined in [MS-GLOS]:

**little-endian**

The following terms are specific to this document:

**common language runtime (CLR):** The common language runtime is the infrastructure that the .NET Framework uses to execute all managed applications. The runtime supplies managed code with services such as cross-language integration, code access security, object lifetime management, and debugging and profiling support.

**user-defined type (UDT):** User-defined types can extend the scalar type system of the SQL Server database, enabling storage of CLR objects in a SQL Server database. UDTs can contain multiple elements and they can have behaviors, which differentiates them from the traditional alias data types that consist of a single SQL Server system data type.

**MAY, SHOULD, MUST, SHOULD NOT, MUST NOT:** These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

## 1.2 References

### 1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.



*Copyright © 2010 Microsoft Corporation.*

[IEEE754] Institute of Electrical and Electronics Engineers, "Standard for Binary Floating-Point Arithmetic", IEEE 754-1985, October 1985, http://ieeexplore.ieee.org/servlet/opac?punumber=2355.

[MS-NRBF] Microsoft Corporation, ".NET Remoting: Binary Format Data Structure", July 2007.

[OGCSFS] Herring, J. R., "OpenGIS Implementation Specification for Geographic information – Simple feature access – Part 1: Common Architecture", OGC 06-103r3 Version 1.2.0, October 2006, http://portal.opengeospatial.org/files/?artifact_id=18241

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

### 1.2.2 Informative References

[IRE1098] D.A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes", Proceedings of the I.R.E., September 1952, pp 1098-1102, http://compression.ru/download/articles/huff/huffman_1952_minimum-redundancy-codes.pdf

[MS-BINXML] Microsoft Corporation, “SQL Server Binary XML Structure Specification”.

[MS-CLRUDT] Microsoft Corporation, “CLR User-Defined Types”, June 2009, http://msdn.microsoft.com/en-us/library/ms131120.aspx.

[MS-TDS] Microsoft Corporation,"Tabular Data Stream Protocol Specification", February 2008.

## 1.3 Structure Overview (Synopsis)

The **geography** and **geometry** data types are used by SQL Server to represent two-dimensional objects. The **geography** data type is designed to handle ellipsoidal coordinates, defined from a variety of standard Earth-shape references, and is used specifically to accommodate geospatial data. The **geometry** data type is nonspecific and can be used for geospatial and other spatial applications that use Cartesian coordinates.

Instances of the **geometry** and **geography** data types can be composed of a variety of complex features whose definitions are stored in various structures. These structures are described in detail later in this document.

The **hierarchyid** data type is used by a SQL Server application to model tree structures in a more efficient way than was formerly possible. This data type significantly improves on the performance of current solutions (for instance, recursive queries).

Values of the **hierarchyid**<1>data type represent nodes in a hierarchy tree. This data type is a system common language runtime (CLR) type, so applications interpret it the same way they would interpret any SQL Server CLR user-defined type (UDT). The binary structure of the data type, described in detail later in this document, uses a variant on Huffman encoding to represent the path from the root of a tree to a particular node in that tree. For more information about Huffman encoding, see [IRE1098].

CLR UDTs can represent any type defined by the user. The user implements a CLR UDT as a structure using the CLR type system. The binary format of a CLR UDT depends on two factors. The first factor is the CLR UDT’s internal structure, as defined by the user. The second factor is the serialization format also chosen by the user. To decode the binary format of a CLR UDT, it is necessary to know these two properties of the CLR UDT.



*Copyright © 2010 Microsoft Corporation.*

The user implementing CLR UDTs can include primitive types and other structures. The structures can include other CLR UDTs. The set of types available for fields may be limited, depending on the serialization format chosen by the user.

The user can choose between two available serialization formats: SQL Server native UDT serialization, and user-defined UDT serialization. SQL Server native UDT serialization is designed for simple CLR UDTs that have a simple structure and use only a specified set of simple primitive types. User-defined UDT serialization is more flexible and enables users to define complex and more dynamic CLR UDTs.

To learn more about CLR UDTs, see [MS-CLRUDT].

## 1.4 Relationship to Protocols and Other Structures

All structures described in this document are designed to be transported over Tabular Data Stream protocol as described in section 2.2.5.5.2 of [MS-TDS].

## 1.5 Applicability Statement

The spatial data format presented in this document is designed for the native code programmer (C and C++, for example) and documents the disk representation for the Microsoft® SQL Server® 2008 R2 **geography** and **geometry** data types. Programmers using managed code (the Microsoft® .NET Framework) are encouraged to use the SQL CLR Types library (SQLSysClrTypes.msi) and the corresponding builder API. Note that Microsoft reserves the right to make changes to this format at any time.

The **HIERARCHYID** format presented in this document is designed to be used solely with managed code (the .NET Framework) by using the SQL CLR Types library (SQLSysClrTypes.msi) and the corresponding APIs. Again, note that Microsoft reserves the right to make changes to this format at any time.

The format of common language runtime (CLR) user-defined types (UDTs) is designed to be used solely with managed code by using the same classes that define CLR UDTs in a Microsoft® SQL Server® client program. As stated earlier in this document, without knowledge of the internal structure of a CLR UDT and the serialization format that it is using, it is impossible to read the CLR UDT from the binary data representing it.

## 1.6 Versioning and Localization

This document describes versions 1 and 2 of the **GEOGRAPHY**, **GEOMETRY** structures and version 1 of the **HIERARCHY** structure.<2> The version number of the **GEOGRAPHY** and **GEOMETRY** structures is stored in the **Version** field in the structure. The version number of the **HIERARCHY** structure is not stored anywhere in the structure. All fields in the **GEOGRAPHY**, **GEOMETRY**, and **HIERARCHYID** structures contain either numeric or bit flag data. There are no localization implications for these structures.

SQL Server does not define any versioning scheme for common language runtime (CLR) user-defined types (UDTs). Any version data created by the user must be part of a CLR UDT itself.

## 1.7 Vendor-Extensible Fields

The **GEOMETRY**, **GEOGRAPHY**, and **HIERARCHY** structures do not contain any extensible fields.

All fields of a common language runtime (CLR) user-defined type (UDT) are defined by the user who creates the type. The serialization format of these fields can also be selected by the user.



*Copyright © 2010 Microsoft Corporation.*

# 2 Structures

## 2.1 System CLR Types Structure

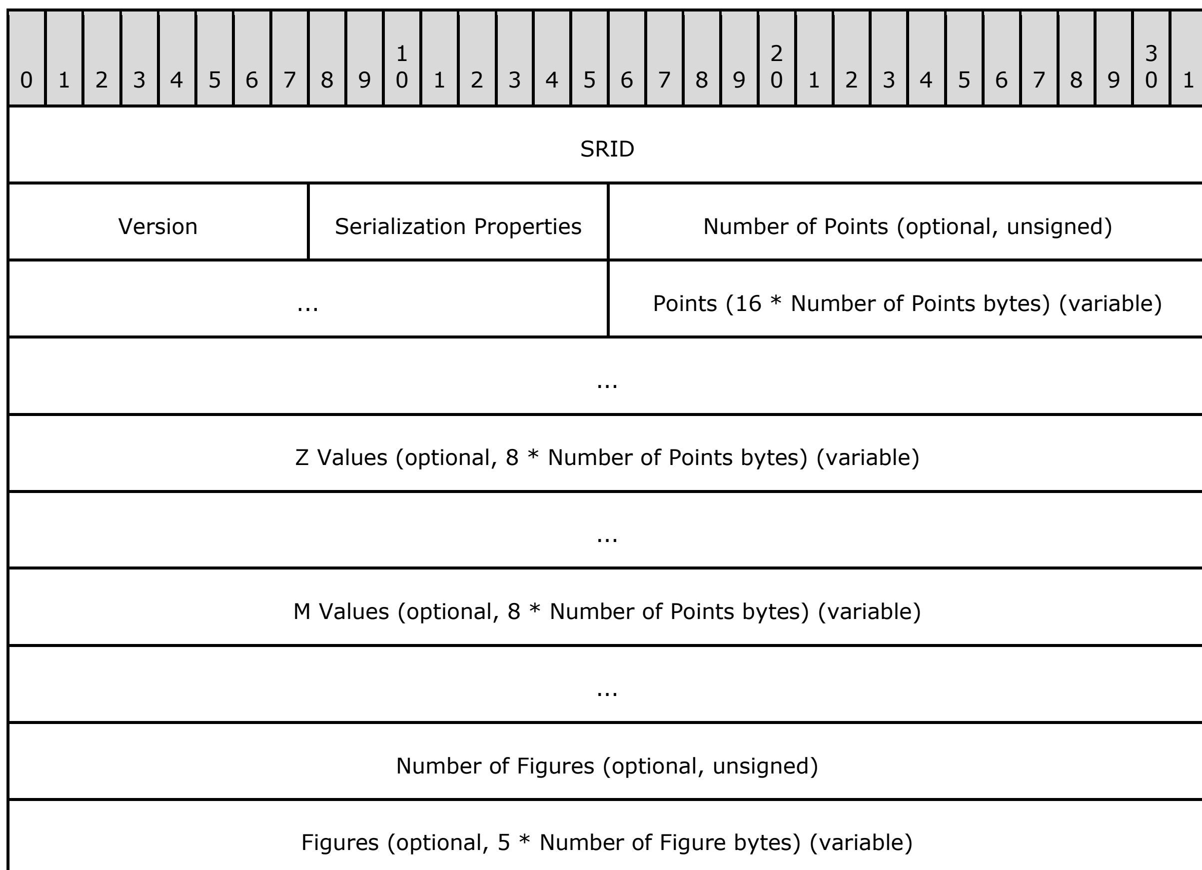
### 2.1.1 GEOGRAPHY and GEOMETRY Structures

The **GEOGRAPHY** and **GEOMETRY** structures are serialized using the binary format described later in this section. The **GEOGRAPHY** structure contains several fixed fields (or header fields) and either three or four secondary structures that are repeated, as necessary, to describe the geography fully.<3>

The **GEOGRAPHY POINT** structure contains the coordinates for an individual point and is repeated for as many points as are present in the **GEOGRAPHY** structure. One shape structure will appear for each OGC simple feature that is contained in the **GEOGRAPHY** structure. A shape can consist of multiple figures, each of which is defined by a single figure structure. The **GEOGRAPHY** structure contains flags and counts that indicate how many of these subsidiary structures are contained in the **GEOGRAPHY** structure.

### 2.1.2 Basic GEOGRAPHY Structure (Version 1)

Version 1 of the **GEOGRAPHY** structure is formatted as shown in the following packet diagram. All double fields contain double-precision floating-point numbers that are 64 bits (8 bytes) long. Integers and double-precision floating-point numbers are expressed in **little-endian** format.



| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 20 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 30 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SRID | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Version | | | | | | | | Serialization Properties | | | | | | | | Number of Points (optional, unsigned) | | | | | | | | | | | | | | | |
| ... | | | | | | | | | | | | | | | | Points (16 * Number of Points bytes) (variable) | | | | | | | | | | | | | | | |
| ... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Z Values (optional, 8 * Number of Points bytes) (variable) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| M Values (optional, 8 * Number of Points bytes) (variable) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Number of Figures (optional, unsigned) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Figures (optional, 5 * Number of Figure bytes) (variable) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |



*Copyright © 2010 Microsoft Corporation.*

| ... |
| --- |
| Number of Shapes (optional, unsigned) |
| Shapes (optional, 9 * Number of Shapes bytes) (variable) |
| ... |

**SRID (4 bytes):** (32 bit integer) The spatial reference identifier (SRID) for the geography. **GEOGRAPHY** structures MUST use SRID values in the range of 4120 through 4999, inclusive, with the exception of null geographies. A value of -1 indicates a null geography. When a null geography is indicated, all other fields are omitted.

**Version (1 byte):** The version of the **GEOGRAPHY** structure.<4>

**Serialization Properties (1 byte):** A bit field that contains individual bit flags that indicate which optional content is present in the structure as well as other attributes of the geography.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | L | P | V | M | Z |

Where the bits are defined as:

| Value | Description |
| --- | --- |
| Z (0x01) | The structure has Z values. |
| M (0x02) | The structure has M values. |
| V (0x04) | Geography is valid.<br>For **GEOGRAPHY** structures, **V** in version 1 is always set. |
| P (0x08) | Geography contains a single point. When **P** is set, **Number of Points**, **Number of Figures**, and **Number of Shapes** are implicitly assumed to be equal to 1 and are omitted from the structure. In addition, **Figures** is implicitly assumed to contain one figure representing a **Stroke** with a **Point Offset** of 0 (zero). Lastly, **Shape** is implicitly assumed to contain one shape of type **Point**, with a **Figure Offset** of 0 (zero) and without any parents (**Parent Offset** set to -1). This is an optimization for the common case of a single point. |
| L (0x10) | Geography contains a single line segment. When **L** is set, **Number of Points** is implicitly assumed to be equal to 2 and does not explicitly appear in the serialized data. In addition, **Figures** is implicitly assumed to contain one stroke figure (0x01) with a **Point Offset** of 0 (zero). Lastly, **Shape** is implicitly assumed to contain one shape of type **0x02** (LineString), with a **Figure Offset** of 0 and without any parents (**Parent Offset** set to -1). |



*Copyright © 2010 Microsoft Corporation.*

| Value | Description |
| --- | --- |
| | **P** and **L** are mutually exclusive properties. |

**Number of Points (optional, unsigned) (4 bytes):** The number of points in the **GEOGRAPHY** structure. This MUST be a positive number. If either the **P** or **L** bit is set in the **Serialization Properties** bit field, this field is omitted from the structure.

**Points (16 * Number of Points bytes) (variable):** A sequence of point structures. The point coordinates are contained in **GEOGRAPHY POINT** structures in **GEOGRAPHY** structures. Likewise, coordinates are contained in **GEOMETRY POINT** structures in **GEOMETRY** structures. Both structures contain a pair of doubles.

If neither the **P** nor **L** bit is set in the **Serialization Properties** bit field, there will be **Number of Points** points in the sequence. If the **P** bit is set, there will be one point. If the **L** bit is set, there will be two points.

**Z Values (optional, 8 * Number of Points bytes) (variable):** A sequence of double values for the Z value of each point. If the **Z** bit is set, there will be **Number of Points** doubles in the array. If a Z value for an individual point is NULL, it is represented by QNaN [IEEE754].

**M Values (optional, 8 * Number of Points bytes) (variable):** A sequence of double values for the M value of each point. If the **M** bit is set, there will be **Number of Points** doubles in the array. If an M value for an individual point is NULL, it is represented as QNaN.

**Number of Figures (optional, unsigned) (4 bytes):** The number of figures in the structure.

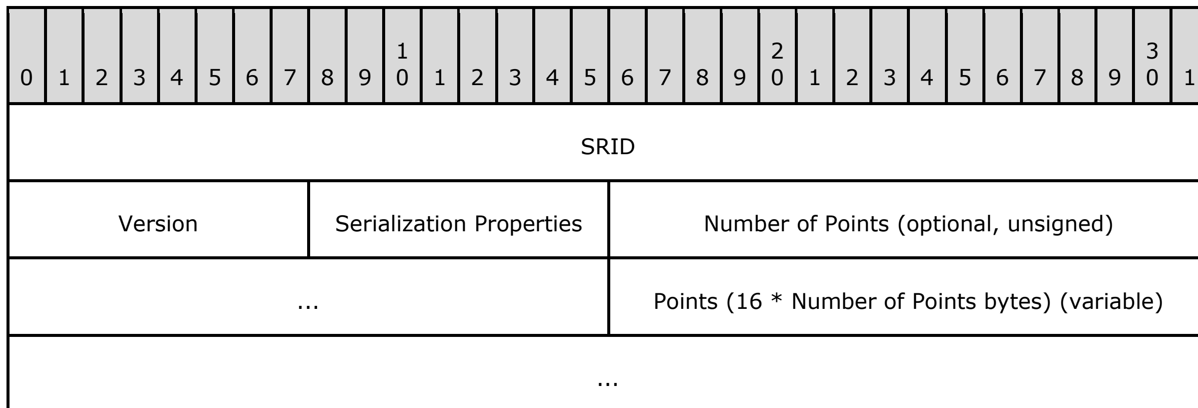
**Figures (optional, 5 * Number of Figure bytes) (variable):** A sequence of figure structures.

**Number of Shapes (optional, unsigned) (4 bytes):** The number of shapes in the structure.

**Shapes (optional, 9 * Number of Shapes bytes) (variable):** A sequence of shape structures.

### 2.1.3 Basic GEOGRAPHY Structure (Version 2)

Version 2 of the **GEOGRAPHY** structure is formatted as shown in the following packet diagram. All double fields contain double-precision floating-point numbers that are 64 bits (8 bytes) long. Integers and double-precision floating-point numbers are expressed in little-endian format.






*Copyright © 2010 Microsoft Corporation.*

| Z Values (optional, 8 * Number of Points bytes) (variable) |
| --- |
| ... |
| M Values (optional, 8 * Number of Points bytes) (variable) |
| ... |
| Number of Figures (optional, unsigned) |
| Figures (optional, 5 * Number of Figure bytes) (variable) |
| ... |
| Number of Shapes (optional, unsigned) |
| Shapes (optional, 9 * Number of Shapes bytes) (variable) |
| ... |
| Number of Segments (optional) |
| Segments (optional) (1 * Number of Segments bytes) (variable) |
| ... |

**SRID (4 bytes):** (32 bit integer) The SRID for the geography. **GEOGRAPHY** structures MUST use SRID values in the range of 4120 through 4999, inclusive, with the exception of null geographies. A value of -1 indicates a null geography. When a null geography is indicated, all other fields are omitted.

**Version (1 byte):** The version of the **GEOGRAPHY** structure. <5>

**Serialization Properties (1 byte):** A bit field that contains individual bit flags that indicate which optional content is present in the structure as well as other attributes of the geography.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | H | L | P | V | M | Z |

Where the bits are defined as:



*Copyright © 2010 Microsoft Corporation.*

| Value | Description |
| --- | --- |
| Z<br>(0x01) | The structure has Z values. |
| M<br>(0x02) | The structure has M values. |
| V<br>(0x04) | Geography is valid. |
| P<br>(0x08) | Geography contains a single point. When **P** is set, **Number of Points**, **Number of Figures**, and **Number of Shapes** are implicitly assumed to be equal to 1 and are omitted from the structure. In addition, **Figures** is implicitly assumed to contain one figure representing a **Stroke** with a **Point Offset** of 0 (zero). Lastly, **Shape** is implicitly assumed to contain one shape of type **Point**, with a **Figure Offset** of 0 (zero) and without any parents (**Parent Offset** set to -1). This is an optimization for the common case of a single point. |
| L<br>(0x10) | Geography contains a single line segment. When **L** is set, **Number of Points** is implicitly assumed to be equal to 2 and does not explicitly appear in the serialized data. In addition, **Figures** is implicitly assumed to contain one stroke figure (0x01) with a **Point Offset** of 0 (zero). Lastly, **Shape** is implicitly assumed to contain one shape of type **0x02** (LineString), with a **Figure Offset** of 0 and without any parents (**Parent Offset** set to -1).<br>**P** and **L** are mutually exclusive properties. |
| H<br>(0x20) | Geography is larger than a hemisphere. This bit is added in version 2 of the serialization format.<6> |

**Number of Points (optional, unsigned) (4 bytes):** The number of points in the **GEOGRAPHY** structure. If either the **P** or **L** bit is set in the **Serialization Properties** bit field, this field is omitted from the structure.

**Points (16 * Number of Points bytes) (variable):** A sequence of point structures. The point coordinates are contained in **GEOGRAPHY POINT** structures in **GEOGRAPHY** structures. Likewise, coordinates are contained in **GEOMETRY POINT** structures in **GEOMETRY** structures. Both structures contain a pair of doubles.

If neither the **P** nor **L** bit is set in the **Serialization Properties** bit field, there will be **Number of Points** points in the sequence. If the **P** bit is set, there will be one point. If the **L** bit is set, there will be two points.

**Z Values (optional, 8 * Number of Points bytes) (variable):** A sequence of double values for the Z value of each point. If the **Z** bit is set, there will be **Number of Points** doubles in the array. If a Z value for an individual point is NULL, it is represented by QNaN [IEEE754].

**M Values (optional, 8 * Number of Points bytes) (variable):** A sequence of double values for the M value of each point. If the **M** bit is set, there will be **Number of Points** doubles in the array. If an M value for an individual point is NULL, it is represented by QNaN.

**Number of Figures (optional, unsigned) (4 bytes):** The number of figures in the structure.

**Figures (optional, 5 * Number of Figure bytes) (variable):** A sequence of figure structures.

**Number of Shapes (optional, unsigned) (4 bytes):** The number of shapes in the structure.



*Copyright © 2010 Microsoft Corporation.*

**Shapes (optional, 9 * Number of Shapes bytes) (variable):** A sequence of shape structures.

**Number of Segments (optional) (4 bytes):** The number of segments in the structure. This MUST be a positive number. Segments are added in version 2 of the serialization format.<7>

**Segments (optional) (1 * Number of Segments bytes) (variable):** A sequence of segment structures.<8>

### 2.1.4 FIGURE Structure

The **FIGURE** structure defines the partitions in the **Points**, **Z Values**, and **M Values** sequences for each constituent of the simple feature represented by the geography. A simple feature may have more than one part, whereas the collection of simple feature types may contain more than one simple feature.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1<br>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2<br>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3<br>0</td><td>1</td></tr>
<tr><td colspan="8">Figures Attribute (byte)</td><td colspan="24">Point Offset (32-bit integer)</td></tr>
<tr><td colspan="8">...</td></tr>
</table>

**Figures Attribute (byte) (1 byte):** Determines the role of this figure within the **GEOMETRY** structure. Valid values in version 1 of the serialization format are as follows:<9>

**0 (0x00):** Figure is an interior ring in a polygon. Interior rings represent holes in exterior rings.

**1 (0x01):** Figure is a stroke. A stroke is a point or a line.

**2 (0x02):** Figure is an exterior ring in a polygon. An exterior ring represents the outer boundary of a polygon.

Valid values in version 2 of the serialization format are as follows:<10>

**0 (0x00):** Figure is a point.

**1 (0x01):** Figure is a line.

**2 (0x02):** Figure is an arc.

**3 (0x03):** Figure is a composite curve, i.e. it contains both line and arc segments.

The order of the coordinates in each ring of a geography polygon (but not a geometry polygon) is important. The outer rings for polygons are constructed by using the "left-hand" rule to determine the interior region of a polygon shape. Thus, outer polygon rings have their **GEOGRAPHY POINT** coordinate pairs ordered in a counter-clockwise direction. Polygon holes are constructed using the "right-hand" rule. Thus, the **GEOGRAPHY POINT** coordinate pairs of a polygon holes are ordered in a clockwise direction.

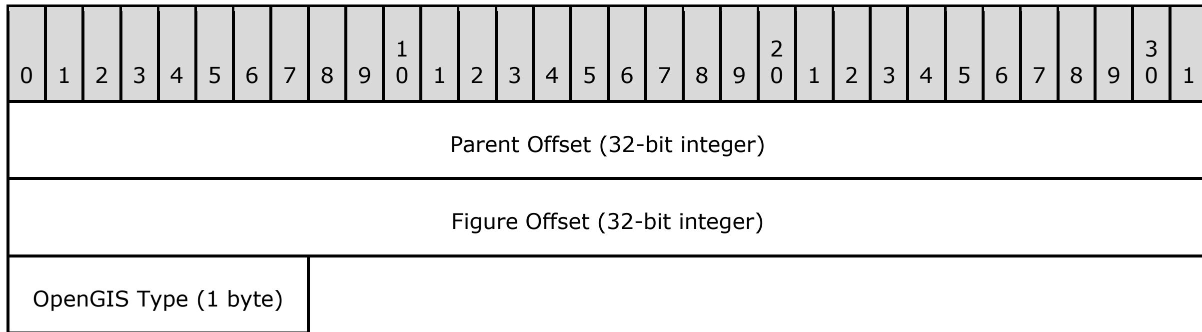
**Point Offset (32-bit integer) (4 bytes):** The offset to the **FIGURE** structure's first point in the **Points**, **Z Values**, and **M Values** sequences.



*Copyright © 2010 Microsoft Corporation.*

### 2.1.5 SHAPE Structure

The **SHAPE** structure identifies each simple feature contained in the **GEOGRAPHY** structure. It links together the simple feature type, the figure that represents it, and the parent simple feature that contains the present simple feature (if there is one).




**Parent Offset (32-bit integer) (4 bytes):** The offset to the **SHAPE** structure's parent (containing) shape in the **Shapes** sequence if the shape has a parent, such as an outer ring if a hole, or a multipart simple feature.

**Figure Offset (32-bit integer) (4 bytes):** The offset to the **SHAPE** structure's **Figure** in the **Figures** sequence.

**OpenGIS Type (1 byte) (1 byte):** The type of simple feature represented by the **SHAPE** structure.

Valid values in version 1 of the serialization format are as follows:<11>

**0 (0x00):** Unknown

**1 (0x01):** Point

**2 (0x02):** LineString

**3 (0x03):** Polygon

**4 (0x04):** MultiPoint

**5 (0x05):** MultiLine

**6 (0x06):** MultiPolygon

**7 (0x07):** GeometryCollection

Version 2 adds the following valid values:<12>

**8 (0x08):** CircularString

**9 (0x09):** CompoundCurve

**10 (0x0A):** CurvePolygon

**11 (0x0B):** FullGlobe

**Note** The example structure provided in this section uses the Well-Known Text (WKT) protocol of [OGCSFS].



*Copyright © 2010 Microsoft Corporation.*

### 2.1.6 GEOGRAPHY POINT Structure

The **GEOGRAPHY POINT** structure contains latitude and longitude coordinates as double values representing a point located on a spheroid.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 20 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 30 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Latitude (double) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Longitude (double) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

**Latitude (double) (8 bytes):** The **GEOGRAPHY POINT** structure's latitude.

**Longitude (double) (8 bytes):** The **GEOGRAPHY POINT** structure's longitude.

**Notes**

- The example structure provided in this section uses the Well-Known Text (WKT) protocol of [OGCSFS].
- Latitude and longitude coordinates are stored as decimal degree values. Negative values are used to designate south latitude and west longitude values.
- Latitude values MUST be between -90 and 90 degrees, inclusive.
- Longitude values MUST be between -15069 and 15069 degrees, inclusive.
- Latitude and Longitude values MUST NOT contain Infinity or NaN [IEEE754].

### 2.1.7 GEOMETRY POINT Structure

The **GEOMETRY POINT** structure contains x-coordinates and y-coordinates as double values representing a point located on a plane.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 20 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 30 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| X Coordinate (double) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Y Coordinate (double) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |



*Copyright © 2010 Microsoft Corporation.*

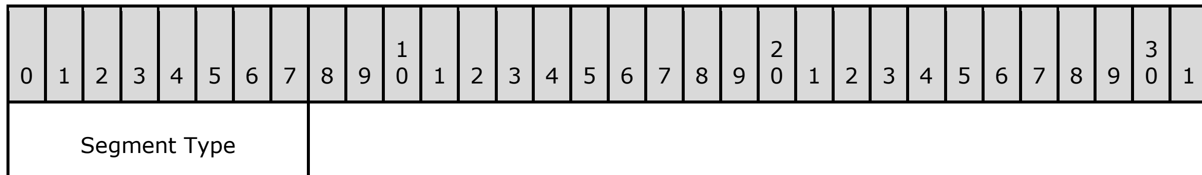
**X Coordinate (double) (8 bytes):** The **GEOMETRY POINT** structure's x-coordinate.

**Y Coordinate (double) (8 bytes):** The **GEOMETRY POINT** structure's y-coordinate.

**X Coordinate** and **Y Coordinate** values MUST NOT contain Infinity or NaN.

### 2.1.8 SEGMENT Structure

The **SEGMENT** structure defines the structure of a compound curve figure.<13> It contains only one byte, which represents type of the segment. Segments are stored only for figures whose Figure Attribute value is 0x03 in version 2 of the serialization format.




**Segment Type (1 byte):** Determines the type of the segment within the figure.

Valid values are as follows:

**0 (0x00):** Segment is a line.

**1 (0x01):** Segment is an arc.

**2 (0x02):** Segment is a first line.

**3 (0x03):** Segment is a first arc.

The first line and first arc segments mark the start of the sequence of segments of the same type, which are line and arc respectively. Subsequent segments have types line and arc.

## 2.2 HIERARCHYID Structure

### 2.2.1 Logical Definition

A hierarchy tree is an abstract ordered tree. This means that for each node *n*, there is a "less-than" (<) order relation on all children of *n*. This tree has infinite depth. For each node *n* in the tree, the children of *n* are in 1-to-1 correspondence with finite nonempty sequences of integers, which are called node labels. Given any two children *m1* and *m2* of *n*, *m1* < *m2* if and only if the label of *m1* comes before the label of *m2* in the lexicographical order in integer sequences. Thus, for a node *n,* each child of *n* has siblings before and after it, and any two children of *n* have siblings between them.

The logical representation of a node label for a child of a given node is a sequence of integers separated by dots (for example, 1, 1.3, or -7.0.-8.9). The **hierarchyid** data type logically encodes information about a single node in the hierarchy tree by encoding the path from the root of the tree to the node. Such a path is logically represented as a sequence of node labels of all children visited after the root. Each label is followed by a slash, and a slash begins the representation. Thus, a path that visits only the root is represented by a single slash. For example, /, /1/, /0.3.-7/, /1/3/, and /0.1/0.2/ are valid **hierarchyid** paths of lengths 1, 2, 2, 3, and 3, respectively.

The **hierarchy** data type represents a node in the hierarchy tree based on a binary encoding of the following logical representation. This encoding is described in section 3.2.




*Copyright © 2010 Microsoft Corporation.*

### 2.2.2 Physical Representation

The logical representation of a node in the hierarchy tree is encoded into a sequence of bits according to the following representation:

| $L_0$ | $O_0$ | $F_0$ | $L_1$ | $O_1$ | $F_1$ | . . . | $L_k$ | $O_k$ | $F_k$ | W |
|---|---|---|---|---|---|---|---|---|---|---|

In the preceding diagram, each L/O pair encodes one integer in the logical representation of the node; each $F_1$ is a single bit that is 0 (zero) if the integer is followed by a dot in the logical representation, and 1 if it is followed by a slash. W is a string of 0 to 7 bits, padding the representation to the nearest byte; all bits in W have value 0.

In the following text, each $L_1/O_1/F_1$ triple is referred to as a level. If $F_1$ is 0 (zero), the level is said to be *fake*; otherwise, it is said to be *real*.

$L_1/O_1$ pairs encode an integer according to the following description. If the *i*th integer in the logical representation of the node is *n*, the $L_1/O_1$ pair encodes *n* for real levels and *n+1* for fake levels. This is done so that, in varbinary (variable-length binary data) comparisons, fake levels compare above real levels.

Each $L_1$ prefix of an $L_1/O_1$ pair specifies a range of integers and a bit size for the following $O_1$ field, as shown in the following table. Each $O_1$ field has some antiambiguity bits that are always of a fixed value for a particular $L_1$ value. These bits are used to enable unambiguous backward parsing of the representation. The third column in the table shows the format of $L_1/O_1$ pair with antiambiguity bits in the $O_1$ field, with all other bits of the $O_1$ field shown as dots.

The actual value of the integer encoded is the value of the $O_1$ field (ignoring antiambiguity bits and interpreting the rest of the bits as an unsigned integer) added to the lower limit of the range corresponding to the $L_1$ field.

| $L_1$ | Bit size of $O_1$ (without/with antiambiguity bits) | Full format of the $L_1/O_1$ pair | Range |
|---|---|---|---|
| 000100 | 48/53 | 000100.............0....................0......0...0.1... | -281479271682120 to -4294971465 |
| 000101 | 32/36 | 000101..................0......0...0.1... | -4294971464 to -4169 |
| 000110 | 12/15 | 000110.....0...0.1... | -4168 to -73 |
| 0010 | 6/8 | 0010..0.1... | -72 to -9 |
| 00111 | 3/3 | 00111... | -8 to -1 |
| 01 | 2/2 | 01.. | 0 to 3 |
| 100 | 2/2 | 100.. | 4 to 7 |
| 101 | 3/3 | 101... | 8 to 15 |
| 110 | 6/8 | 110..0.1... | 16 to 79 |
| 1110 | 10/13 | 1110...0...0.1... | 80 to 1103 |



Copyright © 2010 Microsoft Corporation.

| $L_1$ | Bit size of $O_1$ (without/with antiambiguity bits) | Full format of the $L_1/O_1$ pair | Range |
| --- | --- | --- | --- |
| 11110 | 12/15 | 11110.....0...0.1... | 1104 to 5199 |
| 111110 | 32/36 | 111110..................0......0...0.1... | 5200 to 4294972495 |
| 111111 | 48/53 | 111111.............0....................0......0...0.1... | 4294972496 to 281479271683151 |

No integer outside of the range -281479271682120 – 281479271683119 can be represented in this encoding.

Also, note that the encoding used in the **hierarchyid** data type is limited to 892 bytes. Consequently, nodes that have too many levels in their representation to fit into 892 bytes cannot be represented by the **hierarchyid** data type.

The encoding for the root node is a binary string of length 1. Thus, there is one level and no W field.

**Note** The encoding represented in the preceding table has three useful properties:

- It is parsable. That is, for any binary string, there is at most one interpretation of it as a sequence of $L_1/O_1/F_1$ triples, and there is an efficient parsing algorithm.
- The representation is also parsable backward (that is, starting from the last byte). This enables an algorithm to determine a node's parent without having to parse the entire binary string.
- Comparing two encodings by lexicographical binary comparison is equivalent to conducting depth-first comparisons on the corresponding tree nodes.

## 2.3 CLR UDTs

This section describes the binary format of common language runtime (CLR) user-defined types (UDTs).

Two serialization formats affect the binary format of a CLR UDT.

Native UDT serialization is designed for simple CLR UDTs that have a simple structure and use only a certain set of simple primitive types.

User-defined UDT serialization is more flexible and lets the user define complex and dynamic CLR UDTs. For more information, see [MSDN-UDTR].

### 2.3.1 Native UDT Serialization

Native user-defined type (UDT) serialization is designed to simplify the serialization of simple common language runtime (CLR) UDTs. Therefore, CLR UDTs that use native UDT serialization have to adhere to certain limitations, as specified in this section.

All primitive fields MUST be of one of the following types:

**BOOL**, **BYTE**, **SBYTE**, **USHORT**, **SHORT**, **UINT**, **INT**, **ULONG**, **LONG**, **FLOAT**, **DOUBLE**, **SqlByte**, **SqlInt16**, **SqlInt32**, **SqlInt64**, **SqlBoolean**, **SqlSingle**, **SqlDouble**, **SqlDateTime**, **SqlMoney**



*Copyright © 2010 Microsoft Corporation.*

CLR UDTs that use native UDT serialization can contain nested structures. Fields defined by these nested structures must adhere to the same limits that apply to fields of CLR UDTs that use native UDT serialization.

The binary format of a CLR UDT that uses native UDT serialization is defined by all of the UDT's fields, in the order in which they are defined by the user. The format of each field depends on its type.

#### 2.3.1.1 Binary Format of Each Byte

Each data type that is formatted by using native user-defined type (UDT) serialization consists of a series of bytes. Each byte is formatted as 8 bits representing the byte value in binary representation in a little-endian bit ordering (formatting all bits in order of their significance, starting with the least significant bit and ending with the most significant bit).

#### 2.3.1.2 Binary Format of Primitive Types

- **BOOL** values are represented as a single byte. Depending on the **BOOL** value, the byte takes one of the following two values: 0x01 for **True** or 0x00 for **False**.
- **BYTE** values are represented as a single byte.
- **SBYTE** values are represented as a single byte, but the most significant bit is reversed from 0 (zero) to 1 or from 1 to 0.
- **USHORT** values are represented as 2 bytes. The most significant byte is first, followed by the least significant byte.
- **SHORT** values are represented as 2 bytes. The most significant byte is first, and it has the most significant bit reversed from 0 (zero) to 1 or from 1 to 0. It is followed by the least significant byte.
- **UINT** values are represented as 4 bytes, in the order of their significance, starting with the most significant byte and ending with the least significant byte.
- **INT** values are represented as 4 bytes, in the order of their significance, starting with the most significant byte and ending with the least significant byte. The most significant byte has the most significant bit reversed from 0 (zero) to 1 or from 1 to 0.
- **ULONG** values are represented as 8 bytes, in the order of their significance, starting with the most significant byte and ending with the least significant byte.
- **LONG** values are represented as 8 bytes, in the order of their significance, starting with the most significant byte and ending with the least significant byte. The most significant byte has the most significant bit reversed, from 0 (zero) to 1 or from 1 to 0.
- **FLOAT** values are represented as defined by 4-byte, [IEEE754] single-precision, floating-point format, but the order of the bytes is reversed.

  For positive values (including positive 0 (zero)), the most significant bit of the first byte is reversed from 0 (zero) to 1.

  For negative 0 (zero), all bits remain unchanged.
- **DOUBLE** values are represented as defined by 8-byte, double-precision, floating-point format, but the order of the bytes is reversed.



*Copyright © 2010 Microsoft Corporation.*

For positive values (including positive 0 (zero), the most significant bit of the first byte is reversed, from 0 (zero) to 1.

For negative values, all bits of all bytes are reversed, from 0 (zero) to 1 or from 1 to 0.

For negative 0 (zero), all bits remain unchanged.

- **SqlByte** values are represented as 2 bytes. The first byte is a **BOOL** value that indicates whether or not the **SqlByte** value is NULL (**True** indicates that this value is not NULL; **False** indicates that it is NULL). The second byte is the actual **BYTE** value representing the **SqlByte** value.
- **SqlInt16** values are represented as 3 bytes. The first byte is a **BOOL** value that indicates whether or not the **SqlInt16** value is NULL (**True** indicates that this value is not NULL; **False** indicates that it is NULL). The other 2 bytes are the actual **SHORT** value representing the **SqlInt16** value.
- **SqlInt32** values are represented as 5 bytes. The first byte is a **BOOL** value that indicates whether or not the **SqlInt32** value is NULL (**True** indicates that this value is not NULL; **False** indicates that it is NULL). The other 4 bytes are the actual **INT** value representing the **SqlInt32** value.
- **SqlInt64** values are represented as 9 bytes. The first byte is a **BOOL** value that indicates whether or not the **SqlInt64** value is NULL (**True** indicates that this value is not NULL; **False** indicates that it is NULL). The other 8 bytes are the actual **LONG** value representing the **SqlInt64** value.
- **SqlBoolean** values are represented as a single byte. Depending on the value of **SqlBoolean**, this byte can have any of the following three values: 0x00 for NULL, 0x01 for **False**, or 0x02 for **True**.
- **SqlSingle** values are represented as 5 bytes. The first byte is a **BOOL** value that indicates whether or not the **SqlSingle** value is NULL (**True** indicates that this value is not NULL; **False** indicates that it is NULL). The other 4 bytes are the actual **FLOAT** value representing the **SqlSingle** value.
- **SqlDouble** values are represented as 5 bytes. The first byte is a **BOOL** value that indicates whether or not the **SqlDouble** value is NULL (**True** indicates that this value is not NULL; **False** indicates that it is NULL). The other 4 bytes are the actual **DOUBLE** value representing the **SqlDouble** value.
- **SqlDateTime** values are represented as 9 bytes. The first byte is a **BOOL** value that indicates whether or not the **SqlDateTime** value is NULL (**True** indicates that this value is not NULL; **False** indicates that it is NULL). The next 4 bytes are an **INT** value representing the date as the number of days elapsed since 1/1/1900 (for dates before 1/1/1900, this will be a negative value). The final 4 bytes are an **INT** value representing the number of ticks elapsed since midnight of the day represented by the date part. The following rules can be used to calculate the number of elapsed ticks from the number of elapsed milliseconds:

  Each second consists of 300 ticks. All ticks represent values with the number of milliseconds ending in 0, 3, or 7. For example: 000, 003, 007, 010, 013, 017, 020, …, 990, 993, 997.

  The valid range for **SqlDateTime** values is from 1753-1-1 00:00:00.000 through 9999-12-31 23:59:59.997.
- **SqlMoney** values are represented as 9 bytes. The first byte is a **BOOL** value that indicates whether or not the **SqlMoney** value is NULL (**True** indicates that this value is not NULL; **False**



*Copyright © 2010 Microsoft Corporation.*

indicates that it is NULL). The other 8 bytes are a **LONG** value representing the **SqlMoney** value multiplied by 10000.

#### 2.3.1.3 Nested Structures

Common language runtime (CLR) user-defined types (UDTs) that use native UDT serialization can include nested structures. Nested structures are represented by formatting all their fields in the order in which they are defined by the user.

### 2.3.2 User-Defined UDT Serialization

User-defined User-defined type (UDT) serialization is used when native UDT serialization does not provide enough flexibility to express more complex and dynamic structures. The user-defined approach lets users implement their own serialization formats using types defined in .NET Remoting Binary Format.

For more details about .NET Remoting Binary Format, see [MS-NRBF].



*Copyright © 2010 Microsoft Corporation.*

# 3 Structure Examples

## 3.1 GEOGRAPHY and GEOMETRY Structure Examples

The following examples illustrate how a selection of simple features is represented in the structures defined in this document.

### 3.1.1 Example of an Empty Point Structure

POINT EMPTY is designed to handle a non-null condition when a function returns an empty set. This may occur, for instance, when two disjoint spatial features are intersected.

POINT EMPTY is represented by the following binary string:

```
0x00000000 01 04 00000000 00000000 01000000 FFFFFFFF FFFFFFFF 01
```

This string is interpreted as shown in the following table.

| Binary value | Description |
| --- | --- |
| 00000000 | SRID = 0 |
| 01 | Version = 1 |
| 04 | Serialization Properties = V (is valid) |
| 00000000 | Number of Points = 0 (no points) |
| 00000000 | Number of Figures = 0 (no figures) |
| 01000000 | Number of Shapes = 1 |
| FFFFFFFF | 1st Shape Parent Offset = -1 (no parent) |
| FFFFFFFF | 1st Shape Figure Offset = -1 (no figure) |
| 01 | 1st Shape OpenGIS Type = 1 (point) |

### 3.1.2 Example of a Geometry Point Structure

POINT(5 10) holds a 0-dimension feature that represents a point location. The following figure shows a geometry point feature located at the intersection of 5 on the x-axis and 10 on the y-axis (the actual point is surrounded by a circular symbol to make it easier to see).

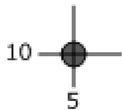



**Figure 1: A geometry point**

POINT (5 10) in SRID 4326 is represented by the following binary string:



*Copyright © 2010 Microsoft Corporation.*

```
0xE6100000 01 0C 0000000000001440 0000000000002440
```

This string is interpreted as shown in the following table.

| Binary value | Description |
|---|---|
| E6100000 | SRID = 4326 |
| 01 | Version = 1 |
| 0C | Serialization Properties = V + P (geometry is valid, single point) |
| 0000000000001440 | X = 5 |
| 0000000000002440 | Y = 10 |

### 3.1.3 Example of a Linestring Structure

A LINESTRING is an ordered series of connected points. The LINESTRING (0 1 1, 3 2 2, 4 5 NULL) contains a Z value for each point location, with the last Z value being NULL. The following figure represents the x and y coordinates only for a geometry type.

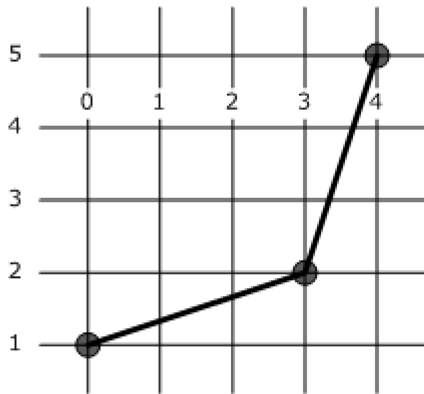



**Figure 2: A geometry linestring**

LINESTRING (0 1 1, 3 2 2, 4 5 NULL) is represented by the following binary string:

0xE6100000 01 05 03000000 0000000000000000 000000000000F03F 0000000000000840 0000000000000040 0000000000001040 0000000000001440 000000000000F03F 0000000000000040 000000000000F8FF 01000000 01 00000000 01000000 FFFFFFFF 00000000 02

This string is interpreted as shown in the following table.

| Binary value | Description |
|---|---|
| E6100000 | SRID = 4326 |
| 01 | Version = 1 |
| 05 | Serialization Properties = V + Z (geometry is valid, has Z values) |
| 03000000 | Number of Points = 3 |



*Copyright © 2010 Microsoft Corporation.*

| Binary value | Description |
|---|---|
| 0000000000000000 | 1st point X = 0 |
| 000000000000F03F | 1st point Y = 1 |
| 0000000000000840 | 2nd point X = 3 |
| 0000000000000040 | 2nd point Y = 2 |
| 0000000000001040 | 3rd point X = 4 |
| 0000000000001440 | 3rd point Y = 5 |
| 000000000000F03F | 1st point Z = 1 |
| 0000000000000040 | 2nd point Z= 2 |
| 000000000000F8FF | 3rd point Z = QNaN |
| 01000000 | Number of Figures = 1 |
| 01 | 1st Figure Attribute = 1 (stroke) |
| 00000000 | 1st Figure Point Offset = 0 (figure starts with 1st point) |
| 01000000 | Number of Shapes = 1 |
| FFFFFFFF | 1st Shape Parent Offset = -1 (no parent) |
| 00000000 | 1st Shape Figure Offset = 0 (shape starts with 1st figure) |
| 02 | 1st Shape OpenGIS Type = 2 (linestring) |

### 3.1.4 Example of a Geometry Collection Structure

A GEOMETRYCOLLECTION is a heterogeneous collection of simple features. The following figure shows a geography containing a single point, a single linestring, and a polygon with an interior ring (hole).

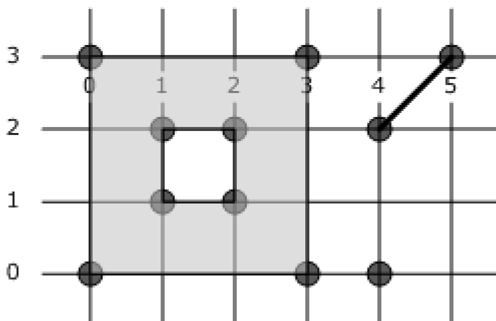



**Figure 3: A geometry collection containing a point, a linestring, and a polygon with a hole**

A GEOMETRYCOLLECTION (POINT (4 0), LINESTRING (4 2, 5 3), POLYGON ((0 0, 3 0, 3 3, 0 3, 0 0), (1 1, 1 2, 2 2, 2 1, 1 1))) is represented by the following binary string:



*Copyright © 2010 Microsoft Corporation.*

```
0xE6100000 01 04 0D000000
0000000000000000 0000000000001040 0000000000000040 0000000000001040 0000000000000840
0000000000001440 0000000000000000 0000000000000000 0000000000000000 0000000000000840
0000000000000840 0000000000000840 0000000000000840 0000000000000000 0000000000000000
0000000000000000 000000000000F03F 000000000000F03F 0000000000000040 000000000000F03F
0000000000000040 0000000000000040 000000000000F03F 0000000000000040 000000000000F03F
000000000000F03F
04000000 01 00000000 01 01000000 02 03000000 00 08000000
04000000 FFFFFFFF 00000000 07 00000000 00000000 01 00000000 01000000 02
00000000 02000000 03
```

This string is interpreted as shown in the following table.

| Binary value | Description |
| --- | --- |
| E6100000 | SRID = 4326 |
| 01 | Version = 1 |
| 04 | Serialization Properties = V (geography is valid) |
| 0D000000 | Number of Points = 13 |
| 0000000000000000 | 1st point latitude = 0 |
| 0000000000001040 | 1st point longitude = 4 |
| 0000000000000040 | 2nd point latitude = 2 |
| 0000000000001040 | 2nd point longitude = 4 |
| 0000000000000840 | 3rd point latitude = 3 |
| 0000000000001440 | 3rd point longitude = 5 |
| 0000000000000000 | 4th point latitude = 0 |
| 0000000000000000 | 4th point longitude = 0 |
| 0000000000000000 | 5th point latitude = 0 |
| 0000000000000840 | 5th point longitude = 3 |
| 0000000000000840 | 6th point latitude = 3 |
| 0000000000000840 | 6th point longitude = 3 |
| 0000000000000840 | 7th point latitude = 3 |
| 0000000000000000 | 7th point longitude = 0 |
| 0000000000000000 | 8th point latitude = 0 |
| 0000000000000000 | 8th point longitude = 0 |
| 000000000000F03F | 9th point latitude = 1 |
| 000000000000F03F | 9th point longitude = 1 |



*Copyright © 2010 Microsoft Corporation.*

| Binary value | Description |
| --- | --- |
| 0000000000000040 | 10th point latitude = 2 |
| 000000000000F03F | 10th point longitude = 1 |
| 0000000000000040 | 11th point latitude = 2 |
| 0000000000000040 | 11th point longitude = 2 |
| 000000000000F03F | 12th point latitude = 1 |
| 0000000000000040 | 12th point longitude = 2 |
| 000000000000F03F | 13th point latitude = 1 |
| 000000000000F03F | 13th point longitude = 1 |
| 04000000 | Number of Figures = 4 |
| 01 | 1st Figure Attribute = 1 (stroke) |
| 00000000 | 1st Figure Point Offset = 0 (figure starts with 1st point) |
| 01 | 2nd Figure Attribute = 1 (stroke) |
| 01000000 | 2nd Figure Point Offset = 1 (figure starts with 2nd point) |
| 02 | 3rd Figure Attribute = 2 (exterior polygon ring) |
| 03000000 | 3rd Figure Point Offset = 3 (figure starts with 4th point) |
| 00 | 4th Figure Attribute = 0 (interior polygon ring) |
| 08000000 | 4th Figure Point Offset = 8 (figure starts with 9th point) |
| 04000000 | Number of Shapes = 4 |
| FFFFFFFF | 1st Shape Parent Offset = -1 (no parent) |
| 00000000 | 1st Shape Figure Offset = 0 (shape starts with 1st figure) |
| 07 | 1st Shape OpenGIS Type = 7 (GeometryCollection) |
| 00000000 | 2nd Shape Parent Offset = 0 (parent shape is 1st shape) |
| 00000000 | 2nd Shape Figure Offset = 0 (shape starts with 1st figure) |
| 01 | 2nd Shape OpenGIS Type = 1 (Point) |
| 00000000 | 3rd Shape Parent Offset = 0 (parent shape is 1st shape) |
| 01000000 | 3rd Shape Figure Offset = 1 (shape starts with 2nd figure) |
| 02 | 3rd Shape OpenGIS Type = 2 (LineString) |
| 00000000 | 4th Shape Parent Offset = 0 (parent shape is 1st shape) |
| 02000000 | 4th Shape Figure Offset = 2 (shape starts with 3rd figure) |
| 03 | 4th Shape OpenGIS Type = 3 (Polygon) |



*Copyright © 2010 Microsoft Corporation.*

### 3.1.5 Example of an Object Serialized in Version 2

This CURVEPOLYGON instance is a surface whose boundary is a curve; in this case, the curve is a COMPOUNDCURVE. It is an instance of geography type that is a hole, so it is larger than a hemisphere.<14>

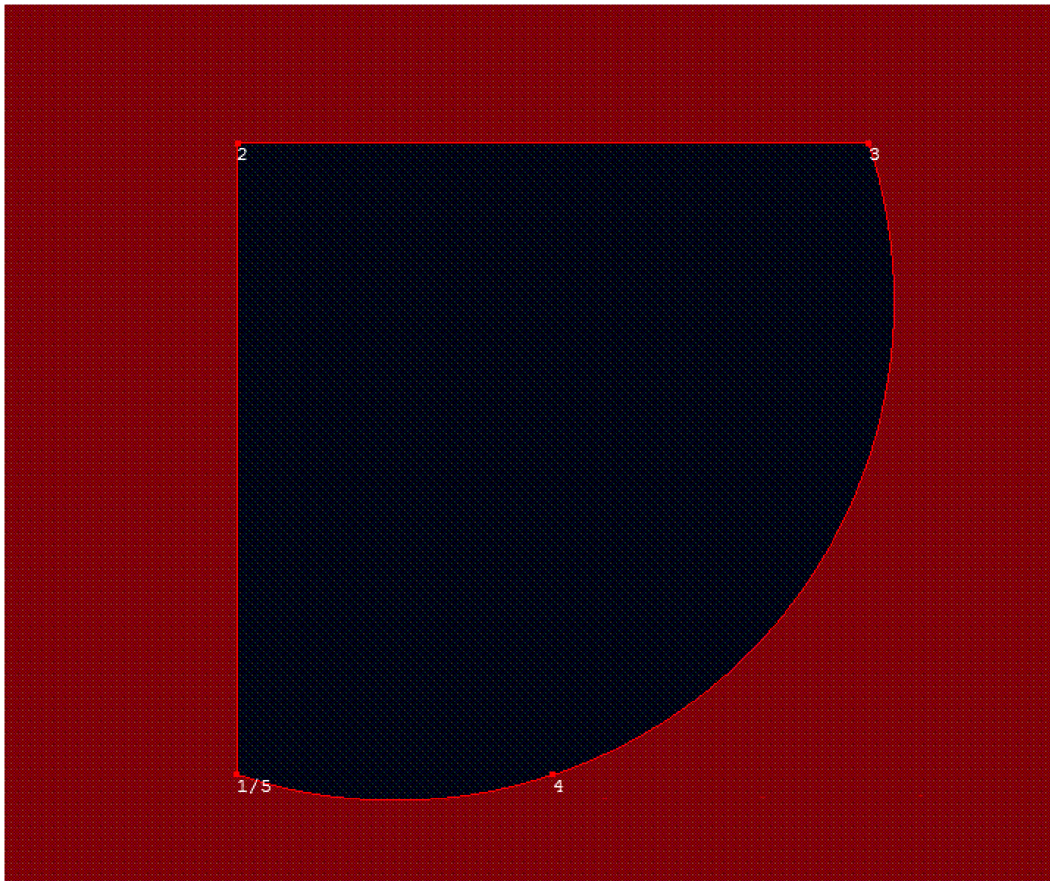



**Figure 4: A curve polygon hole**

A CURVEPOLYGON(COMPOUNDCURVE((0 0, 0 2, 2 2), CIRCULARSTRING (2 2, 1 0, 0 0))) is represented by the following binary string:

```
E6100000 02 24 05000000
0000000000000000 0000000000000000 0000000000000040 0000000000000000 0000000000000040
0000000000000040 0000000000000000 000000000000F03F 0000000000000000 0000000000000000
01000000 03 00000000
01000000 FFFFFFFF 00000000 0A
03000000 02 00 03
```

This string is interpreted as shown in the following table.



Copyright © 2010 Microsoft Corporation.

| Binary value | Description |
|---|---|
| E6100000 | SRID = 4326 |
| 02 | Version = 2 |
| 24 | Serialization Properties = VH (geography which is valid and larger than a hemisphere) |
| 05000000 | Number of Points = 5 |
| 0000000000000000 | 1st point latitude = 0 |
| 0000000000000000 | 1st point longitude = 0 |
| 0000000000000040 | 2nd point latitude = 2 |
| 0000000000000000 | 2nd point longitude = 0 |
| 0000000000000040 | 3rd point latitude = 2 |
| 0000000000000040 | 3rd point longitude = 2 |
| 0000000000000000 | 4th point latitude = 0 |
| 000000000000F03F | 4th point longitude = 1 |
| 0000000000000000 | 5th point latitude = 0 |
| 0000000000000000 | 5th point longitude = 0 |
| 01000000 | Number of Figures = 1 |
| 03 | 1st Figure Attribute = 3 (compound curve) |
| 00000000 | 1st Figure Point Offset = 0 (figure starts with 1st point) |
| 01000000 | Number of Shapes = 1 |
| FFFFFFFF | 1st Shape Parent Offset = -1 (no parent) |
| 00000000 | 1st Shape Figure Offset = 0 (shape starts with 1st figure) |
| 0A | 1st Shape OpenGIS Type = 10 (CurvePolygon) |
| 03000000 | Number of Segments = 3 |
| 02 | 1st Segment Segment Type = 2 (First Line) |
| 00 | 2nd Segment Segment Type = 0 (Line) |
| 03 | 3rd Segment Segment Type = 3 (First Arc) |

## 3.2 HIERARCHYID Examples

The root node is represented by a **HIERARCHYID** structure.

**Example 1**



*Copyright © 2010 Microsoft Corporation.*

The first child of the root node, with a logical representation of /1/, is represented as the following bit sequence:

01011000

The first two bits, 01, are the L1 field, meaning that the first node has a label between 0 (zero) and 3. The next two bits, 01, are the O1 field and are interpreted as the integer 1. Adding this to the beginning of the range specified by the L1 yields 1. The next bit, with the value 1, is the F1 field, which means that this is a "real" level, with 1 followed by a slash in the logical representation. The final three bits, 000, are the W field, padding the representation to the nearest byte.

**Example 2**

As a more complicated example, the node with logical representation /1/-2.18/ (the child with label -2.18 of the child with label 1 of the root node) is represented as the following sequence of bits (a space has been inserted after every grouping of 8 bits to make the sequence easier to follow):

01011001 11111011 00000101 01000000

The first three fields are the same as in the first example. That is, the first two bits (01) are the L1 field, the second two bits (01) are the O1 field, and the fifth bit (1) is the F1 field. This encodes the /1/ portion of the logical representation.

The next 5 bits (00111) are the L2 field, so the next integer is between -8 and -1. The following 3 bits (111) are the O2 field, representing the offset 7 from the beginning of this range. Thus, the L2 and O2 fields together encode the integer -1. The next bit (0) is the F2 field. Because it is 0 (zero), this level is fake, and 1 has to be subtracted from the integer yielded by the L2 and O2 fields. Therefore, the L2, O2, and F2 fields together represent -2 in the logical representation of this node.

The next 3 bits (110) are the L3 field, so the next integer is between 16 and 79. The subsequent 8 bits (00001010) are the L4 field. Removing the anti-ambiguity bits from there (the third bit (0) and the fifth bit (1)) leaves 000010, which is the binary representation of 2. Thus, the integer encoded by the L3 and O3 fields is 16+2, which is 18. The next bit (1) is the F3 field, representing the slash (/) after the 18 in the logical representation. The final 6 bits (000000) are the W field, padding the physical representation to the nearest byte.

## 3.3 CLR UDT Serialization Example

The following example of a common language runtime (CLR) user-defined type (UDT) contains all of the primitive types described in this document. The CLR UDT is defined in the C# programming language as follows.

```
[SqlUserDefinedType(Format.Native)]
public struct SampleNativeUdt : INullable
{
    public bool BoolValue;
    public byte ByteValue;
    public sbyte SByteValue;
    public short ShortValue;
    public ushort UShortValue;
    public int IntValue;
    public uint UIntValue;
    public long LongValue;
    public ulong ULongValue;
    public float FloatValue;
    public double DoubleValue;
    public SqlByte SqlByteValue;
```



Copyright © 2010 Microsoft Corporation.

```
    public SqlInt16 SqlInt16Value;
    public SqlInt32 SqlInt32Value;
    public SqlInt64 SqlInt64Value;
    public SqlDateTime SqlDateTimeValue;
    public SqlSingle SqlSingleValue;
    public SqlDouble SqlDoubleValue;
    public SqlMoney SqlMoneyValue;
    public SqlBoolean SqlBooleanValue;

    // Implementation methods
}
```

In the preceding example, the CLR UDT's fields are initialized with the following values.

```
BoolValue = true;
ByteValue = 1;
SByteValue = -2;
ShortValue = 3;
UShortValue = 4;
IntValue = -5;
UIntValue = 6;
LongValue = 7;
ULongValue = 8;
FloatValue = 1.234568E+08;
DoubleValue = 123456789.0123456;
SqlByteValue = 9;
SqlInt16Value = -10;
SqlInt32Value = 11;
SqlInt64Value = 12;
SqlDateTimeValue = "1/1/2000 12:00:00";
SqlSingleValue = 1.234568E+08;
SqlDoubleValue = 123456789.0123456;
SqlMoneyValue = "$13";
SqlBooleanValue = true;
```

Binary formatting of this CLR UDT produces the following stream of bytes in hexadecimal notation. Anything after "--" is a comment intended to improve the readability of this example and is not part of the binary format for this CLR UDT.

```
01 -- bool true
01 -- byte 1
7E -- sbyte -2
8003 -- short 3
0004 -- ushort 4
7FFFFFFB -- int -5
00000006 -- uint 6
8000000000000007 -- long 7
0000000000000008 -- ulong 8
CCEB79A3 -- float 123456789.0123456789
3E6290CBABF35BA7 -- double -123456789.0123456789
0109 -- SqlByte [bool true, byte 9]
017FF6 -- SqlInt16 [bool true, short -10]
018000000B -- SqlInt32 [bool true, int 11
01800000000000000C -- SqlInt64 [bool true, long 12]
```



Copyright © 2010 Microsoft Corporation.

```
0180008EAC80C5C100 -- SqlDateTime [bool true, int 36524 days since 1/1/1900, int 12960000
ticks since midnight]
013314865C -- SqlSingle [bool true, float 1.234568E+08]
01C19D6F34540CA458 -- SqlDouble [bool true, double 123456789.0123456]
01800000000001FBD0 -- SqlMoney [bool true, long 130000]
02 -- SqlBoolean true
```



Copyright © 2010 Microsoft Corporation.

Release: Friday, September 3, 2010

# 4 Security Considerations

None.



Copyright © 2010 Microsoft Corporation.

# 5 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products:

- Microsoft® SQL Server® 2008 R2
- Microsoft® SQL Server® code-named Denali Community Technology Preview 1 (CTP1)

Exceptions, if any, are noted below. If a service pack number appears with the product version, behavior changed in that service pack. The new behavior also applies to subsequent service packs of the product unless otherwise specified.

Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that product does not follow the prescription.

<1> Section 1.3: The Microsoft implementation does not produce values outside of the range 00:00:00.0000000 through 23:59:59.9999999, but it will accept values outside of the range as described in section 2.4.2 in [MS-BINXML].

<2> Section 1.6: Version 1 denotes SQL Server 2008 R2. Version 2 denotes SQL Server Denali CTP1 .

<3> Section 2.1.1: There are three secondary structures in SQL Server 2008 R2 and four secondary structures in SQL Server Denali CTP1 .

<4> Section 2.1.2: Version 1 denotes the SQL Server 2008 R2 version of the structure. Version 2 denotes the SQL Server Denali CTP1 version of the structure.

<5> Section 2.1.3: Version 1 denotes the SQL Server 2008 R2 version of the structure. Version 2 denotes the SQL Server Denali CTP1 version of the structure.

<6> Section 2.1.3: This bit is introduced in SQL Server Denali CTP1

<7> Section 2.1.3: This bit is introduced in SQL Server Denali CTP1 .

<8> Section 2.1.3: This bit is introduced in SQL Server Denali CTP1 .

<9> Section 2.1.4: These values apply to SQL Server 2008 R2.

<10> Section 2.1.4: These values apply to SQL Server Denali CTP1 .

<11> Section 2.1.5: These values apply to SQL Server 2008 R2.

<12> Section 2.1.5: These values apply to SQL Server Denali CTP1 .

<13> Section 2.1.8: This structure is introduced in SQL Server Denali CTP1 .

<14> Section 3.1.5: This example applies to SQL Server Denali CTP1 .



*Copyright © 2010 Microsoft Corporation.*

# 6 Change Tracking

This section identifies changes that were made to the [MS-SSCLRT] protocol document between the June 2010 and September 2010 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- An extensive rewrite, addition, or deletion of major portions of content.
- Changes made for template compliance.
- Removal of a document from the documentation set.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the language and formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.



*Copyright © 2010 Microsoft Corporation.*

- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- Obsolete document removed.

Editorial changes are always classified with the change type "Editorially updated."

Some important terms used in revision type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact protocol@microsoft.com.

| Section | Tracking number (if applicable) and description | Major change (Y or N) | Change Type |
|---|---|---|---|
| 1 Introduction | Updated document to reflect that it specifies the binary format for SQL Server Denali CTP1, in addition to SQL Server 2008 R2. | Y | Content updated. |
| 1.6 Versioning and Localization | Added content that is specific to SQL Server Denali CTP1. | Y | New product behavior note added. |
| 2.1.1 GEOGRAPHY and GEOMETRY Structures | Added content that is specific to SQL Server Denali CTP1. | N | New product behavior note added. |
| 2.1.2 Basic GEOGRAPHY Structure (Version 1) | Updated content to include information that is specific to SQL Server Denali CTP1. | Y | New product behavior note added. |
| 2.1.3 Basic GEOGRAPHY Structure (Version 2) | Updated content to include information that is specific to SQL Server Denali CTP1. | Y | New product behavior note added. |
| 2.1.4 FIGURE Structure | Updated content to include information that is specific to SQL Server Denali CTP1. | Y | New product behavior note added. |
| 2.1.5 SHAPE Structure | Updated content to include information that is specific to SQL Server Denali CTP1. | Y | New product behavior note added. |
| 2.1.8 | Added section on the SEGMENT structure, which | Y | New content |



Copyright © 2010 Microsoft Corporation.

| Section | Tracking number (if applicable) and description | Major change (Y or N) | Change Type |
| --- | --- | --- | --- |
| SEGMENT Structure | is specific to SQL Server Denali CTP1. | | added. |
| 3.1.5 Example of an Object Serialized in Version 2 | Added section on example of an object serialized in versions 2, which is specific to SQL Server Denali CTP1. | Y | New content added. |



Copyright © 2010 Microsoft Corporation.

# 7 Index

**A**

Applicability statement 6

**B**

Basic GEOGRAPHY structure 7

**C**

Change tracking 33

**E**

Examples
- Empty point structure 21
- GEOGRAPHY structure 21
- Geometry collection structure 23
- Geometry point structure 21
- GEOMETRY structure 21
- Linestring structure 22

**F**

FIGURE packet 12
FIGURE structure 12

**G**

GEOGRAPHY packet (section 2.1.2 7, section 2.1.3 9)
GEOGRAPHY POINT packet 14
GEOGRAPHY POINT structure 14
GEOGRAPHY structure 7
GEOMETRY POINT packet 14
GEOMETRY POINT structure 14
GEOMETRY structure 7
Glossary 4

**H**

HIERARCHYID examples 27
HIERARCHYID structure 15
- Logical definition 15
- Physical representation 16

**I**

Informative references 5
Introduction 4

**L**

Localization 6

**N**

Normative references 4

**P**

Product behavior 32

**R**

References
- Informative 4
- Normative 4

Relationship to protocols and other structures 6

**S**

Security considerations 31
SEGMENT packet 15
SHAPE packet 13
SHAPE structure 13
SQL Server versions 32
Structure examples 21
Structure overview (synopsis) 5
Structures 7
- FIGURE 12
- GEOGRAPHY 7
- GEOGRAPHY POINT 14
- GEOMETRY 7
- GEOMETRY POINT 14
- SHAPE 13

System CLR Types structure 7

**T**

Tracking changes 33

**V**

Vendor-extensible fields 6
Versioning 6



*Copyright © 2010 Microsoft Corporation.*