[MS-SSAS8]:
SQL Server Analysis Services
Version 8.0 Protocol

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 Open Specification Promise or the Community Promise. 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, email 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.

Revision Summary

Date	Revision History	Revision Class	Comments
06/27/2008	1.0	Major	First release.
12/12/2008	1.01	Editorial	Revised and edited the technical content.
08/07/2009	1.1	Minor	Updated the technical content.
11/06/2009	1.1.1	Editorial	Revised and edited the technical content.
03/05/2010	1.1.2	Editorial	Revised and edited the technical content.
04/21/2010	1.1.3	Editorial	Revised and edited the technical content.
06/04/2010	1.1.4	Editorial	Revised and edited the technical content.
09/03/2010	1.1.4	No change	No changes to the meaning, language, or formatting of the technical content.
02/09/2011	1.1.4	No change	No changes to the meaning, language, or formatting of the technical content.
07/07/2011	1.1.4	No change	No changes to the meaning, language, or formatting of the technical content.
11/03/2011	1.1.4	No change	No changes to the meaning, language, or formatting of the technical content.
01/19/2012	1.1.4	No change	No changes to the meaning, language, or formatting of the technical content.
02/23/2012	1.1.4	No change	No changes to the meaning, language, or formatting of the technical content.
03/27/2012	1.1.4	No change	No changes to the meaning, language, or formatting of the technical content.
05/24/2012	1.1.4	No change	No changes to the meaning, language, or formatting of the technical content.
06/29/2012	1.1.4	No change	No changes to the meaning, language, or formatting of the technical content.
07/16/2012	1.1.4	No change	No changes to the meaning, language, or formatting of the technical content.
10/08/2012	1.1.4	No change	No changes to the meaning, language, or formatting of the technical content.

Contents

1 Introduction

SQL Server Analysis Services Version 8.0 provides methods for a client to communicate with, and perform operations on, an **Online Analytical Processing (OLAP)** server.

Sections 1.8, 2, and 3 of this specification are normative and can contain the terms MAY, SHOULD, MUST, MUST NOT, and SHOULD NOT as defined in RFC 2119. Sections 1.5 and 1.9 are also normative but cannot contain those terms. All other sections and examples in this specification are informative.

1.1 Glossary

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

Component Object Model (COM)
flag
handshake
object
Transmission Control Protocol (TCP)
Unicode

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

data definition language (DDL)
data source
dimension
hierarchy
measure
measure group
member
Multidimensional Expressions (MDX)
Online Analytical Processing (OLAP)
ragged hierarchy
record

The following terms are specific to this document:

allow axis: When Dimension Security is enabled, it is the set of **tuples** which are allowed on the axis, based on the current user's security permissions.

B-level: A level automatically created in a **hierarchy** that is not defined as part of the permanently stored **hierarchy** metadata. An end user creates it by utilizing a custom grouping feature.

cache ratio: The ratio of **members** within a particular subhierarchy of a **dimension**. This allows the **SONAR** algorithm to optimally distribute computation between the client and the server.

calculation pass: A stage of calculation in a multidimensional cube in which applicable calculations are evaluated. Multiple passes may be required to complete all calculations.

cell writeback: Refers to an OLAP feature in which the client can define calculations that are written back to the server.

commit version: Within a transaction, there may be multiple times that changes are committed to the database. Each time such an internal commit operation is performed, the **commit version** is incremented.

custom member formula: An expression associated with a **dimension member** that defines how the cell value at this **member** should be calculated.

custom rollup: An aggregation calculation that is customized for a **dimension** level or **member** and that overrides the aggregate functions of a cube's **measures**.

data member: A child **member** associated with a parent **member** in a parent-child **hierarchy**. A **data member** contains the data value for its parent **member**, rather than the aggregated value for its children.

Decision Support Objects (DSO): A programming language provided to define OLAP objects, such as cubes and **dimensions**.

dimension writeback: Refers to the capability where a client may perform an operation that modifies a **dimension's** structure on the server, for example, by adding or deleting **members** on the **dimension**.

drillthrough: This is the OLAP capability to go beyond browsing the data in a cube and to drill into the original data store.

fact table: A central table in a data warehouse schema that contains numerical **measures** and keys relating facts to **dimension** tables. **Fact tables** contain data that describes specific events within a business, such as bank transactions or product sales.

flexible dimension: A **dimension** in which vertices and subtrees can be flexibly moved around during an incremental update operation. If a **dimension** is not flexible and needs to have **members** moved around, it must be processed in a full processing operation, not in an incremental processing operation.

linked cube: A cube based on a cube defined on another OLAP server.

master dimension name: For a virtual **dimension**, this is the **dimension** upon which it is based.

master level name: For a level within a virtual **dimension**, this is the level upon which it is based.

master property name: For a virtual **dimension**, this is the property upon which it is based.

member key: The property of a **member** that specifies the identifier used for that **member**. The value of this property can specify either a cell that contains the identifier or an expression that evaluates to the identifier.

member name: The property of a **member** that specifies the name of the **member**. The value of this property can specify a cell that contains the name or an expression that evaluates to the name.

member qualified name: A **member name**, with some qualifiers added to it. However, even with qualifiers, it is not guaranteed to be unique.

member unique name: A **member name**, with qualifiers added, such that is guaranteed to be unique within a cube.

named set: A set of **dimension members** or a set expression that is created for reuse, for example, in **Multidimensional Expressions (MDX)** queries.

nonleaf: In a tree structure, an element that has one or more subordinate elements. It may be a **dimension member** that has one or more descendants or, in indexes, an intermediate index node that points to other intermediate nodes or leaf nodes.

Ntab: An internal structure, stored in either client or server memory, which allows mapping of a PATH to a **member object** that contains information about the **member**.

partition: One of the storage containers for data and aggregations of a cube. Every cube contains one or more **partitions**. For a cube with multiple **partitions**, each **partition** can be stored separately in a different physical location. Each **partition** can be based on a different **data source**. **Partitions** are not visible to users; the cube appears to be a single **object**.

private dimension: A **dimension** created for and used by a specific cube. Unlike shared **dimensions**, **private dimensions** are available only to the cube in which they are created.

real-time OLAP operation: It is usual that OLAP cubes are processed periodically, and so there is a time lag between data updates in the source store and the OLAP cube. With real-time OLAP, the cube is updated immediately whenever source data changes in the source data store.

rigid dimension: A **dimension** in which **members** cannot change position within the **dimension's hierarchy** during an incremental processing operation. If a **member** has to change its position in the **hierarchy**, it would require a full reprocessing of the **dimension**.

rowset: The OLE DB **object** that is used to contain a result set. It also exhibits cursor behavior depending on the **rowset** properties set by an application.

SET: An ordered set of **tuples** with the same dimensionality.

sibling: A **member** in a **dimension hierarchy** that is a child of the same parent as a specified **member**. For example, in a Time **dimension** with Year and Month levels, the **members** January 1997 and February 1997 are **siblings**.

solve order: The order of evaluation (from highest to lowest **solve order**) and calculation (from lowest to highest **solve order**) for calculated **members**, custom **members**, **custom rollup** formulas, and calculated cells in a single **calculation pass** of a multidimensional cube. **Solve order** is used to determine formula precedence when calculating values for cells in multidimensional cubes, but only within a single **calculation pass**.

SONAR: An algorithm used to divide computation of query results between the client and server in the most optimal way.

Stateful Visual Total: A displayed, aggregated cell value for a **dimension member** that is consistent with the displayed cell values for its displayed children. The visual total of a cell can vary from the actual total if some children of the cell are hidden.

tuple: An ordered set of **members** from different **hierarchies**. It is not necessary that every **hierarchy** be represented in a **tuple**.

type A hole: Holes can occur in a **dimension** when a **member's** parent is not on the level directly above the **member**. A **type A hole** is where the hole is in the middle of the **dimension** structure (as opposed to at the leaf or at a point where all descendants are also holes, or as opposed to the case of a user-defined grouping level).

unary operator: A property associated with a **dimension member** that controls how the **member** should be aggregated. The **unary operator** overrides the defined aggregation function for the **measure**.

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

References to Microsoft Open Specifications documentation do not include a publishing year because links are to the latest version of the documents, which are updated frequently. References to other documents include a publishing year when one is available.

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.

[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-ERREF] Microsoft Corporation, "Windows Error Codes".

[MS-LCID] Microsoft Corporation, "Windows Language Code Identifier (LCID) Reference".

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981, http://www.ietf.org/rfc/rfc0793.txt

[RFC2109] Kristol, D., and Montulli, L., "HTTP State Management Mechanism", RFC 2109, February 1997, http://www.ietf.org/rfc/rfc2109.txt

[RFC2616] Fielding, R., Gettys, J., Mogul, J., et al., "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999, http://www.ietf.org/rfc/rfc2616.txt

[RFC2246] Dierks, T., and Allen, C., "The TLS Protocol Version 1.0", RFC 2246, January 1999, http://www.ietf.org/rfc/rfc2246.txt

[RFC4178] Zhu, L., Leach, P., Jaganathan, K., and Ingersoll, W., "The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism", RFC 4178, October 2005, http://www.ietf.org/rfc/rfc4178.txt

[RFC4234] Crocker, D., Ed., and Overell, P., "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005, http://www.ietf.org/rfc/rfc4234.txt

[UNICODE] The Unicode Consortium, "Unicode Home Page", 2006, http://www.unicode.org/

[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

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary".

[MS-OFCGLOS] Microsoft Corporation, "Microsoft Office Master Glossary".

[MS-OFSGLOS] See [MS-OFCGLOS].

[MS-SSAS9] Microsoft Corporation, "SQL Server Analysis Services Protocol".

[MSDN-ASPIA] Berger, A., and Sharma, A., "Analysis Services: Performance Implications of the Architecture", http://msdn.microsoft.com/en-us/library/aa375780.aspx

[MSDN-CacheRatio] Microsoft Corporation, "Cache Ratio Property", http://msdn.microsoft.com/en-us/library/aa237427(SQL.80).aspx

[MSDN-DefaultMDXVisualMode] Microsoft Corporation, "Default MDX Visual Mode Property", http://msdn.microsoft.com/en-us/library/aa256082(SQL.80).aspx

[MSDN-DTMRP] Microsoft Corporation, "Data Type Mapping in Rowsets and Parameters", http://msdn.microsoft.com/en-us/library/aa198346(SQL.80).aspx

[MSDN-ExecutionLocation] Microsoft Corporation, "Execution Location Property", http://msdn.microsoft.com/en-us/library/aa237489(SQL.80).aspx.

[MSDN-MDSCHEMA_PROPERTIES] Microsoft Corporation, "MDSCHEMA_PROPERTIES Rowset", http://msdn2.microsoft.com/en-us/library/ms126309.aspx

[MSDN-MDXCompatibility] Microsoft Corporation, "MDX Compatibility Property", http://msdn.microsoft.com/en-us/library/aa256070.aspx

[MSDN-MDXUniqueNameStyle] Microsoft Corporation, "MDX Unique Name Style Property", http://msdn.microsoft.com/en-us/library/aa237332(sql.80).aspx

[MSDN-SafetyOptions] Microsoft Corporation, "Safety Options Property", http://msdn.microsoft.com/en-us/library/aa237323(sql.80).aspx.

[MSDN-SecuredCellValue] Microsoft Corporation, "Secured Cell Value Property", http://msdn.microsoft.com/en-us/library/aa237345(SQL.80).aspx

[MSDN-SID] Microsoft Corporation, "SID Structure", http://msdn.microsoft.com/en-us/library/aa379594(VS.85).aspx.

[MSDN-SQLCompatibility] Microsoft Corporation, "SQL Compatibility Property", http://msdn2.microsoft.com/en-us/library/aa237364(SQL.80).aspx

[MSDN-SSPI] Microsoft Corporation, "Security Support Provider Interface (SSPI)", http://msdn.microsoft.com/en-us/library/aa378663(VS.85).aspx

[MSDN-Variant.VariantType] Microsoft Corporation, "Variant.VariantType Enumeration", http://msdn.microsoft.com/en-us/library/microsoft.visualstudio.package.variant.varianttype.aspx

[MSDN-WSEC] Microsoft Corporation, "Windows Sockets Error Codes", http://msdn.microsoft.com/en-us/library/ms740668(VS.85).aspx

[MSFT-JET] Woods, L., "Microsoft Jet 4.0 Sorting APIs: DBCompareStringW and DBLCMapStringW", March 2007, http://technet.microsoft.com/en-us/library/bb293146.aspx

[MSFT-KB210276] Microsoft Corporation, "How to Store, Calculate, and Compare Date/Time Data in Microsoft Access", http://support.microsoft.com/kb/210276

[MSFT-KB51414] Microsoft Corporation, "Internal Format of CURRENCY Data Type in Basic Products", http://support.microsoft.com/kb/51414

1.3 Protocol Overview (Synopsis)

SQL Server Analysis Services Version 8.0 provides methods for a client to communicate with, and perform operations on, an Online Analytical Processing (OLAP) server.

The client and server exchange messages as binary packages over TCP/IP. Alternatively, the same binary packages can be exchanged between the client and server by wrapping them in HTTP or HTTPS, with a few exceptions in binary package composition as noted in section 2.1.2.

This protocol specification documents the content of these messages. Requests are initiated by the client and responded to by the server. In this protocol, the client and server are very tightly coupled and at times exchange memory blocks and pointers to memory blocks with each other. Both the client and server can modify the blocks and pointers and pass them back to the other. This protocol assumes that the client and server have shared information about how the memory is configured on the other end of the protocol.

The following diagram illustrates this concept.

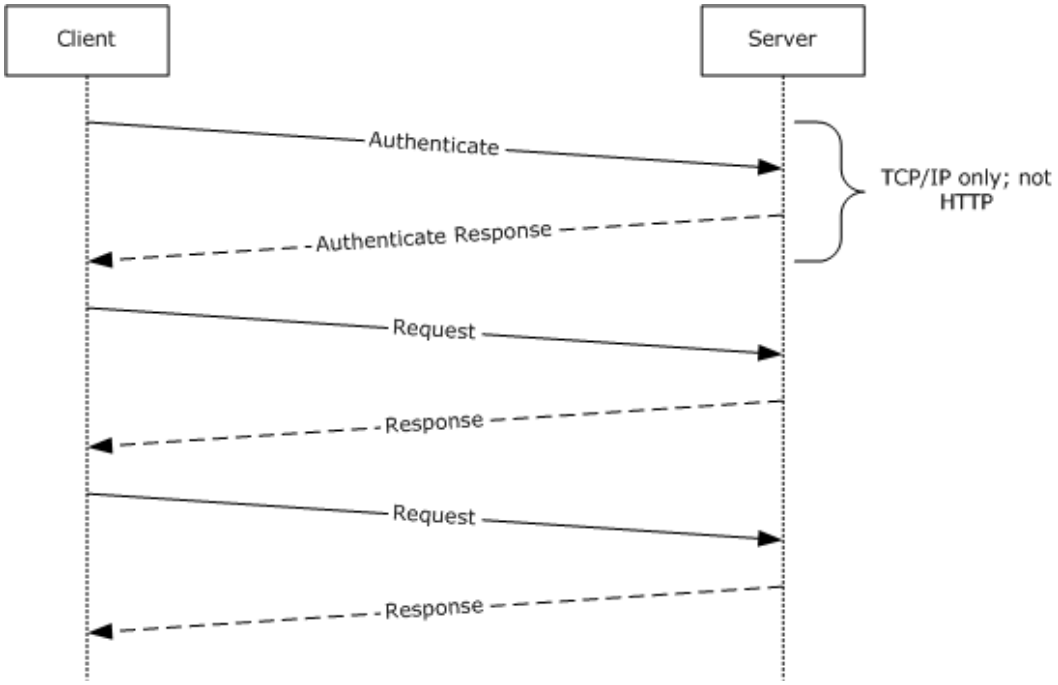


Figure 1: Data transfer between a client and an analysis server

1.4 Relationship to Other Protocols

The conversation between a client and a server is performed over **Transmission Control Protocol (TCP)** as defined in [RFC793] or HTTP 1.1 as defined in [RFC2616] with a number of layers added on top of them.

When the protocol is performed over TCP/IP, it is stacked as shown in the following diagram.

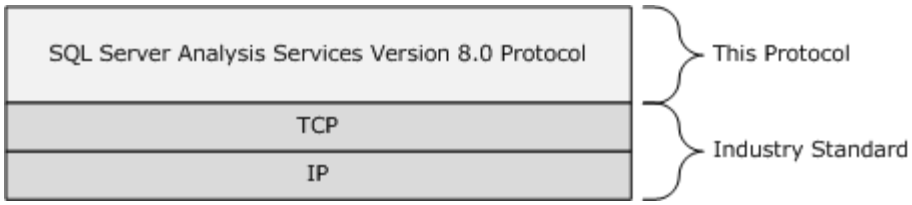


Figure 2: TCP/IP

Alternatively, the conversation can be performed over HTTP or HTTPS by using a data pump component which tunnels the binary packages over HTTP or HTTPS, as specified in section 2.2.1.6. In this method, the protocol transmits the same binary content that is transmitted over TCP/IP, but it is now wrapped in HTTP or HTTPS. A few small exceptions between the TCP/IP and the HTTP or HTTPS binary packages are noted in section 2.1.2. The protocol stack when using HTTP or HTTPS is as shown in the following diagram.

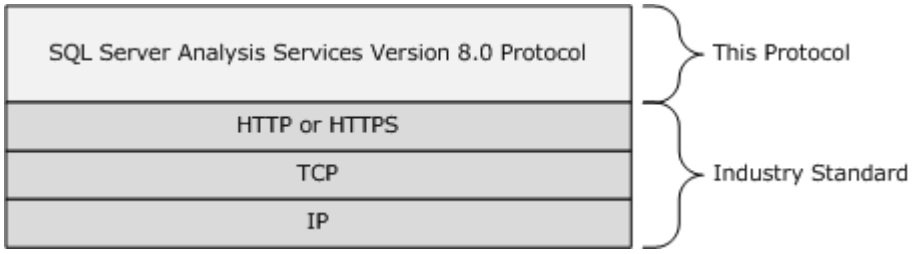


Figure 3: HTTP or HTTPS over TCP/IP

1.5 Prerequisites/Preconditions

Throughout this document, it is assumed that the client has already discovered the server and established an HTTP or HTTPS or TCP/IP connection.

1.6 Applicability Statement

This protocol supports the exchange of messages between a client and an OLAP server.

SQL Server Analysis Services Version 8.0 is a legacy version of the protocol used by Microsoft products for the exchange of messages between a client and an OLAP server. [MS-SSAS9] describes the protocol currently in use by Microsoft products. It is highly recommended that implementers implement the version of the protocol described in [MS-SSAS9].

1.7 Versioning and Capability Negotiation

This protocol does not exhibit explicit or implicit versioning or capability negotiation behavior. This protocol supports the ability of the client and server to exchange version information, as described in section 2.2.3.1.3. However, there is no specific action recommended for the client or server to take if the versions do not match.

1.8 Vendor-Extensible Fields

There are no vendor extensible fields.

This protocol uses HRESULT values as defined in section 2.1 of [MS-ERREF]. Vendors can define their own HRESULT values, provided they set the C bit (0x20000000) for each vendor-defined value, indicating the value is a customer code.

This protocol uses Win32 error codes as defined in section 2.2 of [MS-ERREF]. Vendors SHOULD reuse those values with their indicated meaning. Choosing any other value runs the risk of a collision in the future.

1.9 Standards Assignments

None.

2 Messages

The formal syntax of all messages is provided in Augmented Backus-Naur Form (ABNF), as specified in [RFC4234].

However, note that this document deviates from the convention of using block-formatted protocol diagrams to depict message structure. This protocol specifies its message syntax by using self-describing elements that are concatenated to create a flexible message system. This is in contrast to C structure-based protocols, such as TCP/IP or HTTP, which often have fixed structures in fixed orders. The ABNF constructs in this specification precisely describe the allowable message contents.

2.1 Transport

The conversation between a client and a server MUST be performed either over TCP as specified in [RFC793] or over HTTP 1.1 as specified in [RFC2616] with a number of layers added on top of them. The diagram in section 1.4 describes the high-level relationship between the different layers that MAY be used.<1>

2.1.1 TCP

The protocol MUST be carried over a single TCP connection from the client to the server TCP port 2725.

2.1.2 HTTP

HTTP version 1.1 MUST be used in conjunction with HTTP State Management Mechanism as specified in [RFC2109].

The structures of the protocol are the same as with TCP, with the following exceptions:

There is no Authenticate request when using HTTP transport. Instead, the HTTP authentication model as specified in [RFC2616] MUST be used.

The response to the Calculate MDX Fragment has different structure, as specified in section 2.2.10.2.

The data structures MUST be wrapped using HTTP elements as specified in section 2.2.1.6.

2.1.2.1 HTTPS

HTTP communications MAY be encrypted with Transport Layer Security (TLS) as specified in [RFC2246].

2.2 Message Syntax

The request and response messages of this protocol are binary data structures that flow between an OLAP client and an OLAP database server.

2.2.1 Common Usage

2.2.1.1 Base Data Blocks

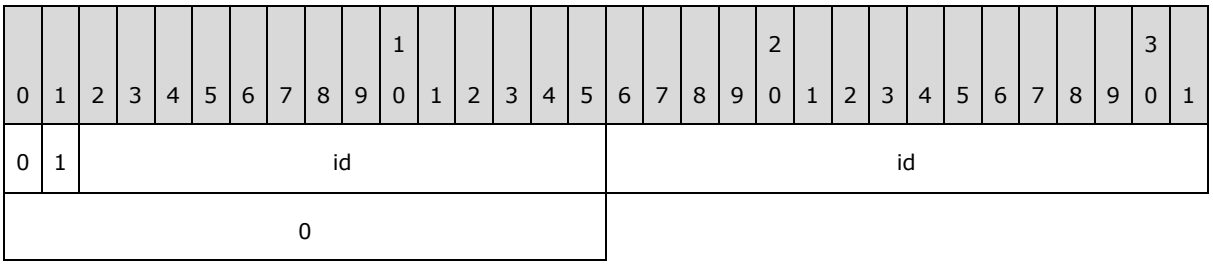
This section describes the packet format for protocol blocks. All the **objects** that are subsequently defined in this protocol description are based upon packets of these base types being exchanged between the client and the server. Implementers can skip ahead and then come back to this section

when they need to understand the actual bit layout of the base data blocks upon which the rest of the specification is based.

2.2.1.1.1 Open block with identifier id

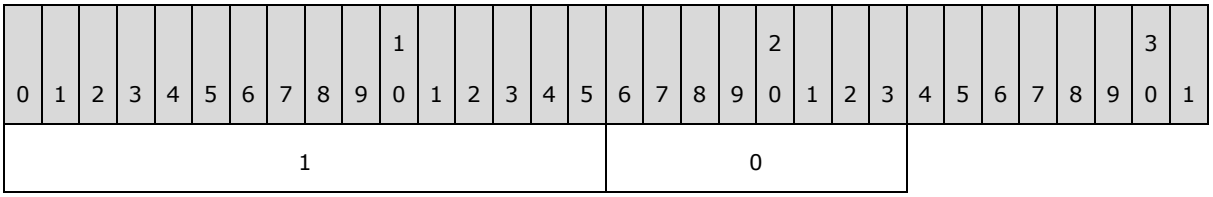
```
OPEN(id)= IDOBJ id %x0000
```

IDOBJ is obtained by setting the first most significant bit in the 16-bit representation of *id* to 0 and setting the second most significant in 16 bit representation of *id* to 1.



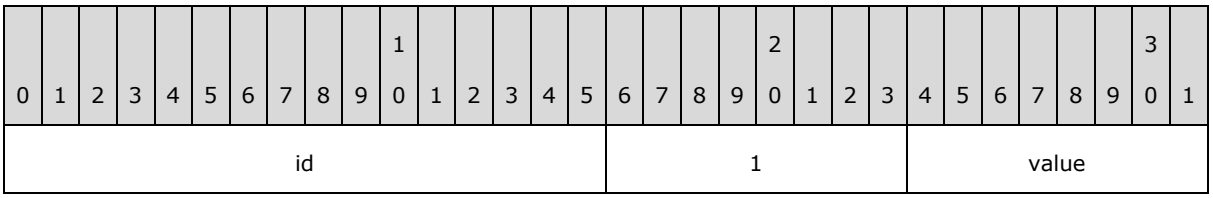
2.2.1.1.2 Close most recent block

```
CLOSE = %x000100
```



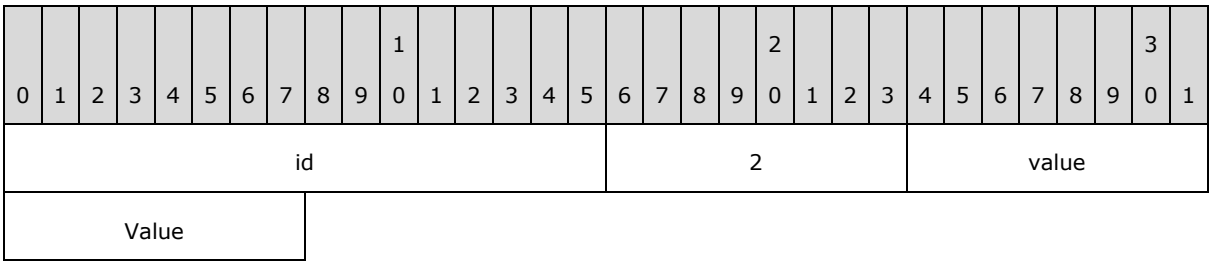
2.2.1.1.3 Byte with identifier id

```
INT8(id)
```



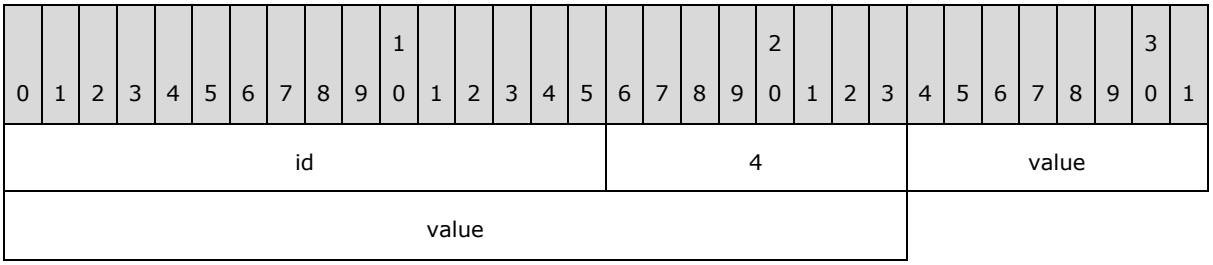
2.2.1.1.4 16 bit integer with identifier id

```
INT16(id)
```



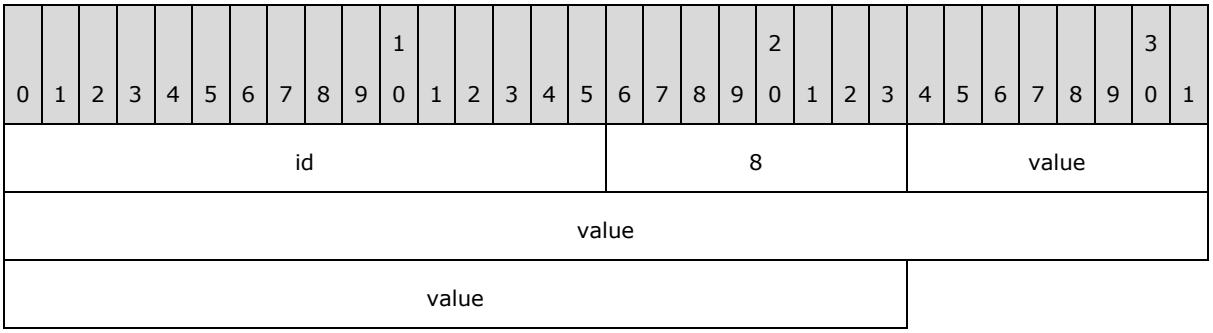
2.2.1.1.5 32 bit integer with identifier id

```
INT32(id)
```



2.2.1.1.6 64 bit integer with identifier id

```
INT64(id)
```



2.2.1.1.7 Unsigned integers with identifier id

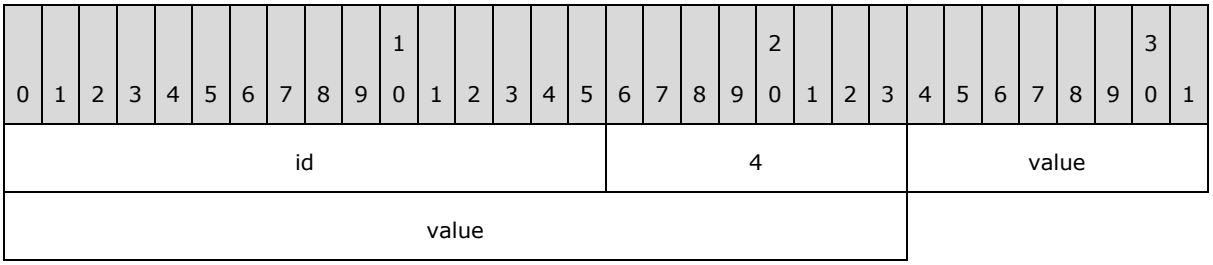
```
UINT16(id), UINT32(id), UINT64(id)
```

Unsigned integers are the same bit format as integers of the same length. Unsigned integers MUST also include an identifier id. The value part of the UINT MUST be an unsigned integer, instead of a signed integer. See sections 2.2.1.1.4, 2.2.1.1.5, and 2.2.1.1.6 for more information.

2.2.1.1.8 32 bit real number with identifier id

```
REAL32(id)
```

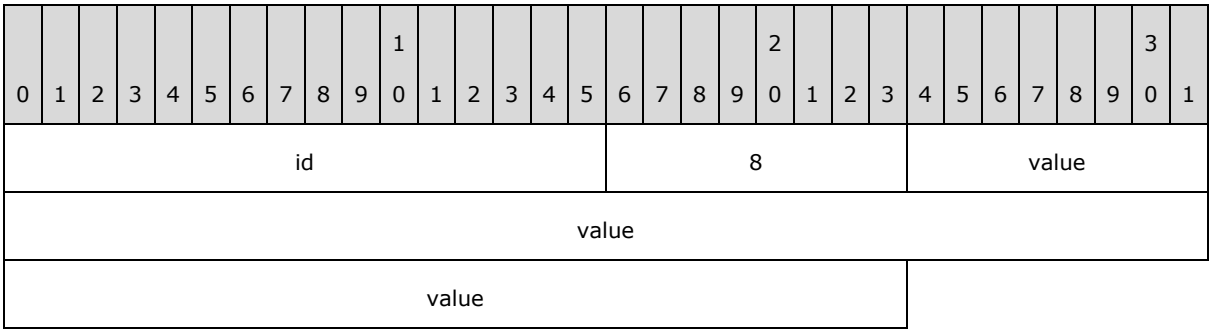
The physical layout of floating point value MUST follow the IEEE 754 Standard for Binary Floating Point Arithmetic as specified in [IEEE754].



2.2.1.1.9 64 bit real number with identifier id

```
REAL64(id)
```

The physical layout of floating point value MUST follow the IEEE 754 Standard for Binary Floating Point Arithmetic as specified in [IEEE754].



2.2.1.1.10 String with identifier id

```
STRING(id)
```

All strings within the protocol MUST be **Unicode** UTF-16. By default, all strings are always NULL-terminated, unless stated otherwise for a specific string. The number of characters in the string MUST include the NULL termination character. Strings are represented on the protocol as an array of bytes, and the number of bytes MUST be equal to the number of characters in the string multiplied by two. If present, the NULL termination character MUST be included in the character count of the string.

2.2.1.1.11 Array of bytes with identifier id and number of bytes length

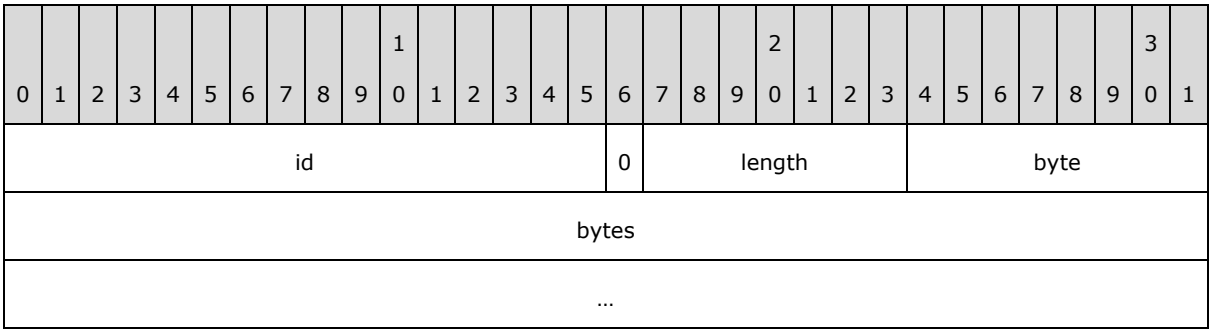
```
ARRAY(id, length)
```

Depending on the number of bytes, there are three different representations of an array of bytes in the protocol.

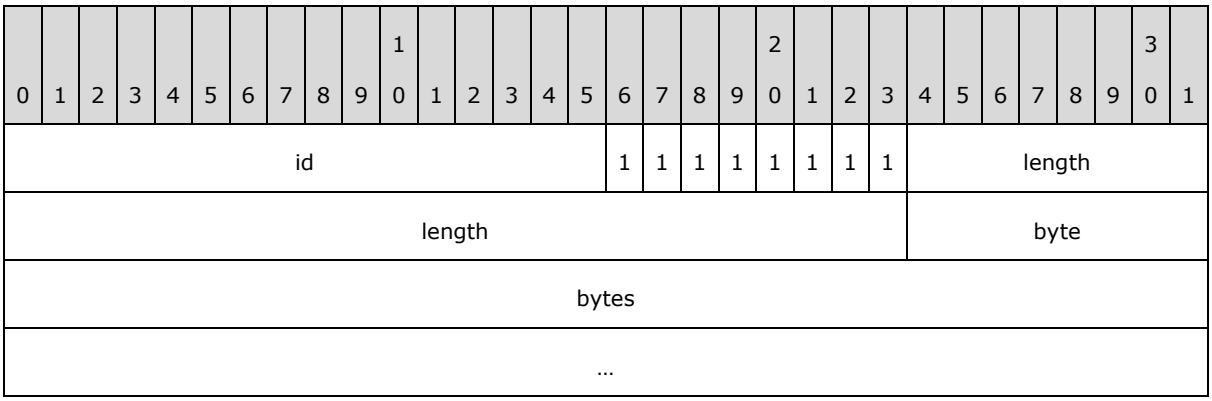
The array of bytes MAY be used for any of the preceding data types described, in addition to characters of a string.<2> References to the following data types in the document refer to this section:

- BYTEARRAY
- INT16ARRAY
- INT32ARRAY
- INT64ARRAY
- REAL32ARRAY
- REAL64ARRAY

2.2.1.1.11.1 Number of bytes less than 128

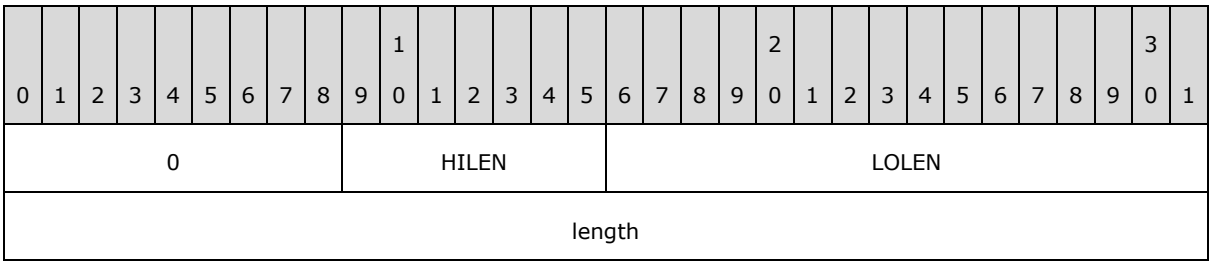


2.2.1.1.11.2 Number of bytes greater than 8323071 (0x7EFFFF)

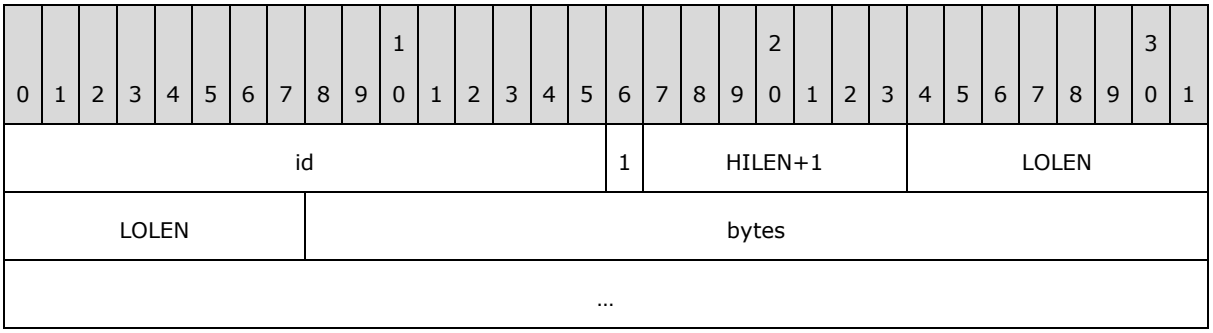


2.2.1.1.11.3 Number of bytes greater or equal to 128 and less or equal to 8323071

The length is divided into two parts. LOLEN and HILEN as shown in the diagram below:



Number 1 is added to HILEN, and the most significant bit in the byte representation of the HILEN+1 MUST be set to 1, as demonstrated at the diagram below:



2.2.1.1.12 Variant

```
Variant(id) =
STRING(id) | INT16 (id)| INT32(id) | INT64(id) | REAL 32(id) | REAL 64(id)
```

2.2.1.2 Request Message Syntax

All request messages MUST be in Unicode as specified in [UNICODE].

```
UNICODECHAR = OCTET OCTET
```

The basic request syntax MUST contain the REQSPEC element and MAY contain the REQDATA element:<3>

```
REQUEST = REQSPEC [REQDATA]
```

Value	Meaning
REQSPEC	This block is request specification and MUST be present.
[REQDATA]	Optional data MAY be appended to the Request. See the description for each individual Request Type to determine if optional REQDATA is allowed.

2.2.1.2.1 REQSPEC

The request specification MUST have the following structure:

```
REQSPEC= REQLENGTH PARAM_STRING [OTHER_PARAMS]
```

Value	Meaning
REQLENGTH	A32-bit value that specifies the request length. The request length is the number of bytes for the entire REQSPEC minus 8 bytes.
PARAM STRING	PARAM_STRING is a Unicode non-NULL terminated string that contains collection of parameters.
[OTHER PARAMS]	OTHER_PARAMS is optional. It specifies additional data to the request which length is included in the REQLENGTH.

2.2.1.2.1.1 PARAM STRING Element

Requests MUST include parameters. The PARAM STRING contains a collection of parameters that are part of the request statement:

```
PARAM_STRING= *( PARAM  ";" )
```

A parameter is a name-value pair that is defined by the following parts:

```
PARAM = PARAM_NAME "=" PARAM_VALUE
```

Value	Meaning
PARAM_NAME	A Unicode string value that specifies the parameter name.
PARAM_VALUE	A Unicode string value that specifies the parameter value.

The first parameter of any request MUST be the REQNAME parameter:

```
REQNAME= "REQUEST" "=" REQCODE
REQCODE= UNICODECHAR
```

Value	Meaning
REQCODE	REQCODE is a request code. It is a Unicode character that is used to differentiate between different requests.

The second parameter of any request MUST be the REQFLAGS parameter:

```
REQFLAGS= "STATE" "=" FLAGS
FLAGS= *( "0"-"9" / "A" / "B" / "C" / "D" / "E" / "F" )
```

The FLAGS property is interpreted as string representation of the 32 bit hexadecimal value.

Throughout this protocol specification, there is reference toINT32REQFLAGS. INT32REQFLAGS refers to the 32 bit unsigned integer bitmask parsed from the string FLAGS, which is described above.

2.2.1.2.2 REQDATA

REQDATA is optional data that MAY be added to a request.<3> Its length is not included in REQLENGTH. The syntax for REQDATA will vary depending on the data structure that formulates the actual request. REQDATA syntax will be defined in the syntax blocks for the messages that support it.

2.2.1.2.3 Object references

The following objects MAY be referred to in the request as part of PARAM_STRING.<4>

2.2.1.2.3.1 Database reference

To reference a database in the request, the following structure MUST be used:

```
DBREF =
""TYPE=b;NAME="" DBNAME "";VER="" DBVER "";LAST=Y;""
```

2.2.1.2.3.2 Cube reference

To reference a cube in the request, the following structure MUST be used:

```
CUBEREF =
""TYPE=b;NAME="" DBNAME "";VER="" DBVER "";LAST=N;""
""TYPE=m;NAME="" CUBENAME "";VER="" CUBEVER "";LAST=Y;DVER="" DIMVER
"";CVER="" DATAVER "";""
```

Value	Meaning
DBNAME	The name of the database where the cube exists.
DBVER	The version of the database where the cube exists.
CUBENAME	The name of the cube to retrieve.
CUBEVER	The version of the cube to retrieve.
DIMVER	The version of the **dimensions** of the cube to retrieve.
DATAVER	The version of the data of the cube to retrieve.

2.2.1.3 Response Message Syntax

This section specifies the syntax of server response messages that are sent to the client. See section 4.4 for an example.

The syntax is defined as follows:

```
RESPONSE = STATUS [RESPONSEDATA]
```

Element	Description
STATUS	Indication of request success or failure.
[RESPONSEDATA]	If the request succeeded, data specific to the request.

2.2.1.3.1 STATUS

```
STATUS    =
  OPEN(170)
        INT32(176)
        OPEN(171)
        RESPONSESTATUS
        RESPONSEERRORCODE
        RESPONSEERRORCODEEX
        RESPONSEERRORNOTE
        CLOSE
  CLOSE
```

Value	Meaning
OPEN(170)	A common data structure that indicates the beginning of the element.
INT32(176)	Constant 0xFFFF (65535).
OPEN(171)	A common data structure that indicates the beginning of the element.
RESOPONSESTATUS	STATUS for the response.
RESPONSEERRORCODE	Error code if the STATUS is not success.
RESPONSEERRORCODEEX	For some results of STATUS, additional information is provided about the error.
RESPONSEERRORNOTE	Text with additional information about the error.
CLOSE	A common data structure that indicates the end of the element.
CLOSE	A common data structure that indicates the end of the element.

2.2.1.3.1.1 RESPONSESTATUS

Returns 1 for success or a series of codes for different types of failures.

Value	Meaning
INT32(172)	Response status indicates whether the request succeeded or failed. Valid values include the following: 1 Success -1 Failed (reason unknown) -2 Internal error on the server

Value	Meaning
	-3 Error raised by the operating system -4 Error occurred during a file operation -5 Error occurred during a memory operation -6 Error occurred during a network operation -7 Error occurred when accessing a **data source** -8 Error occurred during a **Component Object Model (COM)** operation -11 Error occurred during a processing operation -12 Error in **Multidimensional Expressions (MDX)** -13 Error in MDX execution -14 Error accessing data -15 Error in MDX query state or MDX session state -16 User's specified error -17 Error during **dimension writeback** -18 Error in the Lock Manager -19 Error during **cell writeback** -20 Error during **drillthrough** -30 Security related error -31 Error occurred during an HTTP operation -32 Error in data mining component -33 Error occurred during XML operation -34 Error related to **real-time OLAP operation** 2 Server cannot respond to the request because it is waiting for a resource 3 Object or property requested does not exist 4 Object or property already exists and cannot be recreated 5 End of data encountered 6 Timeout occurred 7 Operation was canceled 8 Client metadata is out of date and needs to be refreshed 9 Object or property requested has been permanently deleted 10 Protocol used by the client is not compatible with the server 11 Object is read only and cannot be modified

2.2.1.3.1.2 RESPONSEERRORCODE

Value	Meaning
INT32(173)	RESPONSEERRORCODE indicates a more detailed error code as described in section 6.1.

2.2.1.3.1.3 RESPONSEERRORCODEEX

In the case of some STATUS codes, RESPONSEERRORCODEEX provides additional information about the error.

Value	Meaning
INT32(174)	The meaning of RESPONSEERRORCODEEX differs depending upon the value of RESPONSESTATUS. RESPONSESTATUS valueRESPONSEERRORCODEEX meaning -3 or -2Windows error code as specified in [MS-ERREF] -6Winsock error code as specified in [MSDN-WSEC]. -12MDX error; as specified in section 6.2. Other Ignore

2.2.1.3.1.4 RESPONSEERRORNOTE

Value	Meaning
STRING(175)	RESPONSEERRORNOTE is a string that is appended to the end of the error message.

2.2.1.4 Paths

2.2.1.4.1 DataID

```
DataID= INT16
```

The DataID is a 16-bit integer value that represents a position within any of the various PATH definitions described in this section, such as PATH, DPath, TPath, and so on. This is a two-byte integer value with no ID.

For the explanation about data ID, see section 7.

2.2.1.4.2 DPath

DPath is the full set of DataIDs for a dimension **member**.

```
D_PATH(id) = INT16ARRAY(id)
```

Value	Meaning
INT16ARRAY(id)	There MUST be one INT16 for each level of the dimension. A member which is not a leaf member of the dimension will have 0 value INT16s as placeholders for levels of the dimension below the member. See section 7.

2.2.1.4.3 TPath

TPath is the concatenation of the DPaths that form a **tuple**. It is the concatenation of the DPaths for the members on each dimension that form the tuple. Every dimension in the cube does not have to contribute to the tuple.

```
T_PATH(id) = INT16ARRAY(id)
```

Value	Meaning
INT16ARRAY(id)	Concatenation of DPaths. The number of dimensions in the tuple is variable depending on context. See section 7.

2.2.1.4.4 Path

PATH is a concatenation of all of the DPaths to a particular cell of a cube. Every dimension in the cube MUST be represented in the PATH.

```
PATH(id) = INT16ARRAY(id)
```

Value	Meaning
INT16ARRAY(id)	Each INT16 value in the array is a component of a DPath. The number and order of dimensions in the PATH depends upon the number and order of dimensions in the cube. No **measure** is included in the PATH. See section 7.

2.2.1.4.5 MPath

MPath is a concatenation of a PATH, plus a reference to a measure. The reference to the Measure MUST be at the beginning of the MPath.

```
MPATH(id) = INT16ARRAY(id)
```

Value	Meaning
INT16ARRAY(id)	Each INT16 value in the array is a component of a DPath. The number of dimensions in the MPath depends upon the number of dimensions in the cube. A DataID for a Measure precedes the concatenation of the DPaths. See section 7.

2.2.1.4.6 Record

```
RECORD = BYTEARRAY(id)
```

A RECORD is a PATH concatenated with a full vector of all the measure values for that cell, each in its own data type.

Value	Meaning
BYTEARRAY(id)	The array consists of two components. There is a component of INT16 bytes that represent DPath INT16 components, followed by a vector of Measure values. Each measure value will be in the respective data type for that measure. See section 7.

2.2.1.4.7 Slice

```
SLICE(id) = INT16ARRAY
```

A SLICE is structurally identical to a PATH. However, in some contexts the structure will be referred to as a SLICE and in other contexts it will be referred to as a PATH. Therefore, it is defined with both names.

Value	Meaning
INT16ARRAY(id)	There are two components to the array. Each INT16 value in the array is a component of a DPath. The granularity vector consists of an integer for each dimension included in the TPath, giving the starting level in that dimension for the slice. See section 7.

2.2.1.4.8 DataSet

```
DATASET(id) =BYTEARRAY(id)
```

A DataSet is an array of levels that is an indication of a depth in each dimension. Every dimension MUST be represented in the DataSet.

Value	Meaning
BYTEARRAY(id)	The array consists of two components. There is a component of INT16 bytes that represent DPath INT16 components (which define a slice), followed by a vector of measure values. Each measure value MUST be in the respective data type for that measure. See section 7.

2.2.1.5 Base Objects

2.2.1.5.1 Object

An **Object** specifies basic information about an object.

The syntax is defined as follows.

```
Object =
 OPEN(7)
  STRING(2)
  INT32(3)
  INT32(4)
  INT64(322)
  REAL64(5)
  STRING(6)
 CLOSE
```

Element	Description
OPEN(7)	A common data structure that indicates the beginning of the element.
STRING(2)	A value that specifies the name of the object.
INT32(3)	A value that specifies the ID of the object.
INT32(4)	A bitmask field of **flags** that specifies additional information about the object.

Element	Description
INT64(322)	An additional bitmask field of flags that specify additional information about the object.
REAL64(5)	A value that specifies the date on which the object was last modified. See section 2.2.1.5.1.1 for the storage format of the date value.
STRING(6)	A value that specifies the description of the object.
CLOSE	A common data structure that indicates the end of the element.

The **INT32(4)** bitmask field MUST have a combination of zero or more of the values that are described in the following tables.

The following table applies to a cube.

Value	Description
0x00000001	The cube is disabled and cannot be queried.
0x00000002	The cube is a virtual cube.
0x00000004	The cube has distinct count measures.
0x00000008	The cube is a **linked cube**.
0x00000010	Drillthrough is enabled in this cube.
0x00000020	The cube has no additive measures.
0x00000040	This value is not used.
0x00000080	The cube exists in the client session only.
0x00000100	The cube represents a **measure group** of the virtual cube from the linked cube.
0x00000200	The cube is a subscriber to the linked cube.

The following table applies to a dimension.

Value	Description
0x00000001	The dimension is disabled.
0x00000002	If this bit is set, the dimension is a shared dimension. If this bit is off, the dimension is a **private dimension**.
0x00000004	This value is not used.
0x00000008	The dimension is a virtual dimension.
0x00000010	The dimension is a ROLAP dimension.
0x00000020	The dimension is an unbalanced dimension.
0x00000040	This value is not used.
0x00000080	Dimension writeback is enabled for this dimension.

Value	Description
0x00000100	The dimension has **custom member formulas**.
0x00000200	The dimension participates in real time OLAP updates.

The following table applies to a measure group.

Value	Description
0x00000001	The measure group is disabled.
0x00000004	The measure group has distinct count measures.
0x00000008	The measure group belongs to a linked cube.
0x00000010	Drillthrough is enabled on the measure group.
0x00000020	The measure group has no additive measures.

The following table applies to a measure.

Value	Description
0x00000001	The measure is disabled.
0x00000002	The measure is for internal use and is not to be queried by the user.
0x00000004	Although the measure aggregation function is set to SUM, the aggregation function to be used is COUNT.
0x00000008	The measure is a nullable measure.

The following table applies to all other objects.

Value	Description
0x00000001	The object is disabled.

The **INT64(332)** bitmask field can have a combination of the values that are described in the following tables.

The following table applies to a dimension.

Value	Description
0x8000000000000000	The dimension is hidden.
0x4000000000000000	The dimension has hidden levels.
0x2000000000000000	The dimension has holes.
0x1000000000000000	The dimension has **B-levels**.
0x0800000000000000	**Member names** are unique across all members of the dimension.

Value	Description
0x0400000000000000	**Member keys** are unique across all members of the dimension.
0x0200000000000000	The dimension can have **siblings** that have the same member name.
0x0100000000000000	The dimension is considered to be small by the server.
0x0010000000000000	The dimension is a data mining dimension.

The following table applies to a level.

Value	Description
0x8000000000000000	The level is hidden.
0x4000000000000000	The level is a B-level.
0x2000000000000000	The level has holes.
0x1000000000000000	The level has a **custom rollup**.
0x0800000000000000	The level has **data members**.
0x0400000000000000	The parent level of this level is a B-level.
0x0200000000000000	The level is unbalanced.
0x0100000000000000	The level has data members.
0x0080000000000000	The level has **Ntab**.
0x0040000000000000	The level has **unary operators**.

2.2.1.5.1.1 Format for Date representation

The Date/Time data type is stored as a double-precision, floating-point number up to 15 decimal places. The integer part of the double-precision number represents the date. The decimal portion represents the time. Valid date values range from -657,434 (January 1, 100 A.D.) to 2,958,465 (December 31, 9999 A.D.). A date value of 0 represents December 30, 1899. Dates before December 30, 1899 are stored as negative numbers.

Valid time values range from .0 (00:00:00) to .99999 (23:59:59). The numeric value represents a fraction of one day. The numeric value can be converted to hours, to minutes, and to seconds by multiplying the numeric value by 24. See [MSFT-KB210276] for additional information.

2.2.1.5.2 LockObject

A **LockObject** describes an object for use by the Lock Manager.

The syntax is defined as follows.

```
LockObject =
  INT8(388)
  ARRAY(385, 16)
```

Element	Description
INT8(388)	A value that indicates whether the unique Object ID (OID) is specified for the Lock Manager.
ARRAY(385, 16)	16 byte array of characters used as Unique Object ID by the Lock Manager.

2.2.1.6 Tunneling messages over HTTP

This protocol implements support for HTTP and HTTPS by wrapping the same binary package that would have been transmitted by TCP within the appropriate HTTP or HTTPS elements, as described in this section and its subsections. HTTP or HTTPS MUST be version 1.1, as specified in [RFC2616] and [RFC2109]. The protocol also includes the following element that does not follow standard HTTP usage:

Element	Type	Role
Pump-Error	HTTP response header	Emitted when an error occurs in the HTTP component. The purpose of the element is to send extended error information to the caller.

2.2.1.6.1 HTTP Request

The request from the HTTP Data Pump component to the server MUST be a valid HTTP 1.1 request that MUST contain the following components:

```
Request = HTTP Method
          Request URI
          HTTP version number
          Host HTTP server
          Content length
          Pragma = Pragma: string
```

2.2.1.6.1.1 Mandatory Request Elements

The following elements MUST be part of the header of an HTTP request from the HTTP Data Pump component to the server.

2.2.1.6.1.1.1 HTTP Method

The HTTP method name is a string value that contains the method name.

```
HTTP Method = POST
```

For a description of the syntax of the POST method, see [RFC2616].

2.2.1.6.1.1.2 Request URI

The Request URI is a string value that contains the Uniform Resource Identifier (URI) of the multidimensional database server that is providing the web service. The URI MUST point to the

msolap.asp object on the web server. The string that specifies the URI MUST end with ""/msolap.asp"".

```
Request URI= abs_path
```

For a description of the syntax of an absolute path, see [RFC2616].

2.2.1.6.1.1.3 HTTP version

The HTTP version number element is a string value that contains the version of the HTTP protocol that is used to parse the message.

```
HTTP version number = HTTP""/"" %.%
```

2.2.1.6.1.1.4 Host

The HOST element contains a string value that specifies the name of the host HTTP server.

```
Request =  HTTP server name
```

2.2.1.6.1.1.5 Content-Length

The Content-Length element is a numeric value that indicates the length of the request body. If the value of Content-Length is greater than 0, the request body MUST contain exactly as many bytes as specified in the Content-Length header. The length of the request body represents the size of the binary package, as defined by the binary protocol.

```
Content-Length =  0 / %%
```

If the value of Content-Length is 0, the request fails. The HTTP server expects the request body to be a buffer containing at least 12 bytes.

2.2.1.6.1.1.6 Pragma header

The Pragma header contains a string value that specifies an optional behavior from the viewpoint of the protocol.

For more information about the syntax and possible values for pragma, see [RFC2616].

```
Pragma= Pragma: string
```

By default, the client emits the string ""no-cache"" as the value for this header, as specified in [RFC2616].

2.2.1.6.1.2 Optional Request Elements

The following elements are optional and MAY be included in the HTTP request from the HTTP Data Pump component to the server.

2.2.1.6.1.2.1 User-Agent header

The User-Agent header is a string value that contains the name of the user agent that originates the request. It SHOULD be included in the request.

For more information about the syntax for User-Agent, see [RFC2616] User-Agent = User-Agent: string.

The client emits the string ""Light DCube"" as the value for the User-Agent header. Any string MAY be emitted by a client for the User-Agent, without impact on the protocol.<5>

2.2.1.6.1.2.2 Proxy-Connection header

The Proxy-Connection element contains a string value that specifies whether the connection should be kept open after the request has been sent. If the value indicates that the connection may be kept open, the HTTP 1.1 proxy will accept subsequent requests that may be executed against the server URI.

```
Proxy-Connection= Proxy-Connection: string
```

By default, the client emits the string ""Keep-Alive"" as the value for this header.<6>

2.2.1.6.1.2.3 Cookie

The Cookie header is a string value that contains information about the client and requests that can be used for managing state information across multiple requests. This is not required, but SHOULD be used for superior performance.

For more information about the syntax for cookies, see [RFC2109].

```
Cookie = none /
  = Cookie: string /
  = Set-cookie: string
```

For this header, by default the client creates a value determined by using the following procedure:

First request: no Cookie header is included.

First response from the server: contains a Set-Cookie header that provides a cookie value to be used during the entire communication session. For information about what constitutes a session, see section 3.1.3.

Subsequent requests in the same session: repeat value of the Cookie header.

For more details, see the Communication Example section 3.1.<7>

2.2.1.6.1.3 Request Body

The Request Body contains a collection of request elements as outlined in section 2.2.1.2 as a binary package.

The size of the binary package that constitutes the request must be exactly the same as the value of the Content-Length header.

2.2.1.6.2 HTTP Responses

The response is collected in the server and sent to the client in one or more chunks, depending on the response size and the HTTP configuration options. Depending on the size of the response body and the HTTP configuration, the response may be one of the following cases:

CASE 1: A binary package with no length specified.

The binary package is wrapped in an HTTP response that explicitly closes the connection, and all bytes in the transmission are considered part of the response. See section 4.1.1 for an example.

CASE 2: A binary package of defined length.

The Data Pump component uses chunked transfer encoding. For more information about this encoding, see [RFC2616]. In this case the connection is not explicitly closed. Subsequent chunks are identified as being part of the same response by use of the Transfer-Encoding field. See section 4.1.2 for an example.

CASE 3: An error.

The error is encapsulated as an HTTP header, as specified in section 2.2.1.6.2.1.4.

The requirements for each case are specified in section 2.2.1.6.2.1.1.6.

2.2.1.6.2.1 Response status and headers

The response MUST include the mandatory elements. However, the response MAY include additional optional elements, as specified in section 2.2.1.6.2.1.5.

2.2.1.6.2.1.1 Mandatory Response Elements

The following headers or elements MUST be present in the response.

2.2.1.6.2.1.1.1 Response status line

The elements in the response status line include the HTTP version, and an HTTP status code, which may include a human-readable reason-phrase. The elements in the Response status line are described in [RFC2616].

```
Status= [HTTP version] [status code] [status text]
Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF
```

HTTP version:

The version is always HTTP/1.1

HTTP status code:

The possible status codes and their associated text descriptions are defined in [RFC2616].

2.2.1.6.2.1.1.2 Response Date

The Response Date element indicates the date and time that the response was generated, as defined in [RFC2616].

```
Date = "Date" ":" HTTP-date
```

2.2.1.6.2.1.1.3 Server

The server field contains information about the software used on the origin server to handle the request. For more information, see [RFC2616].

```
Server  = "Server" ":" 1*( product | comment )
```

2.2.1.6.2.1.1.4 Content-Type

The Content-Type element defines the media type of the entity-body sent to the client. For more information, see [RFC2616].

```
Content-Type = "Content-Type" ":" media-type
```

This field always contains the ""text/html"" value.

2.2.1.6.2.1.1.5 Expires

The Expire element defines the time that the response was generated. For more information, see [RFC2616].

```
Expires = "Expires" ":" HTTP-date
```

Ordinarily, the Expires entity-header field gives the date/time after which the response is considered stale. However, the response is not static, hence it expires immediately.

2.2.1.6.2.1.1.6 Cache-Control

The Cache-Control element indicates that the response is intended for a single user and MUST NOT be cached, as described by [RFC2616].

```
Cache-Control = cache-control: ""private""
```

This header always has the """private"" value.

2.2.1.6.2.1.2 Mandatory Elements for Case 1 Responses

As described in section 2.2.1.6.2, a Case 1 response represents a binary package with no length specified.

2.2.1.6.2.1.2.1 Connection header

The Connection header is always present in Case 1 responses. This header is used when the binary response is sent in a single package to the client, without using chunked encoding.

As specified in [RFC2616], it tells the client that the connection will be closed after completion of the response.

For Case 1 responses, the value of this header is always ""close"".

```
Connection: close
```

Because the connection will be closed after the response, the response body MUST include the response size, stated as a number of bytes.

2.2.1.6.2.1.3 Mandatory Elements for Case 2 Responses

As described in section 2.2.1.6.2.1.3, a Case 2 response represents a chunked binary package.

2.2.1.6.2.1.3.1 Transfer-Encoding header

The Transfer-Encoding header is always present in Case 2 responses. This header is used when the binary response is sent is sent in multiple chunks to the client. For more information, see [RFC2616].

```
Transfer-Encoding: chunked
```

For Case 2 responses, the value for this header is always ""chunked"".

2.2.1.6.2.1.4 Mandatory Elements in Case 3 Responses

As described in section 2.2.1.6.2, a Case 3 response represents an error.

2.2.1.6.2.1.4.1 Pump-Error header

The Pump-Error header MUST be present in Case 3 responses. This header is used when an error occurred in processing the request. The possible values of this header are described in section 2.2.1.6.3.

2.2.1.6.2.1.5 Optional Response Elements

The following headers MAY be present in the response. They MAY appear in the response depending on the configuration of the web server. They do not affect the communication.

2.2.1.6.2.1.5.1 X-Powered-By header

The X-Powered-By header contains additional information about the software that is installed on the server.

```
X-Powered-By: string
```

An example of a value that may be emitted during communication is ""ASP.NET"".<8>

2.2.1.6.2.1.5.2 Set-Cookie header

The Set-Cookie header MAY be included in the response for the first HTTP request sent by a client.

The first request creates a session and the value of this header contains the session identifier, as described in [RFC2109].<9>

2.2.1.6.2.2 Response Body

The response body is a binary package in the following form

0x0D	0x0A	0x3C	0x48	0x54	0x4D	0x4C	0x3E	RESPONSE

The syntax for the HTTP response is as follows:

```
HTTPRESPONSE = 0x0D  0x0A  0x3C 0x48 0x54 0x4D 0x4C 0x3E RESPONSE
```

Specification of RESPONSE is in section 2.2.1.3.

2.2.1.6.3 Error Reporting from the HTTP Server

Errors are reported back to the client in a custom HTTP header named ""Pump-Error."" The Pump-Error header specifies the information that is returned by the HTTP server in response to a failed server request.

The Pump-Error header is an ASCII string with the syntax defined as follows:

```
PUMP-ERROR =      HttpErrorHeader
                  Error
                  Note
                  ResponseStatusStr
                  ResponseErrorCodeStr
                  ResponseErrorCodeExStr
                  ErrorMessage

HttpErrorHeader = [Error]
                  [ExtError]
                  [SysError]
                  [Note]

Error =           "<Error>"
                  ResponseStatusStr
```

```
                "</Error>"

ExtError=       "<ExtError>"
                ResponseErrorCodeStr
                "</ExtError>"

SysError=       "<SysError>"
                ResponseErrorCodeExStr
                "</SysError>"

Note =          "<Note>"
                ErrorMessage
                "</Note>"

ResponseStatusStr = STRING

ResponseErrorCodeStr = STRING

ResponseErrorCodeExStr = STRING

ErrorMessage = STRING
```

ResponseStatusStr is a string representation of integer RESPONSESTATUS, ResponseErrorCodeStr is a string representation of integer RESPONSEERRORCODE, ResponseErrorCodeExStr is a string representation of integer RESPONSEERRORCODEEX as specified in section 2.2.1.3.1.

2.2.2 Authenticate

This request authenticates the user on the server and establishes the user's identity. The data that is being sent is obtained from the calls to the SSPI functions with the ""NTLM"" package.

For more information about SSPI, see [MSDN-SSPI]. SSPI is based on GSS, as specified in [RFC4178].

2.2.2.1 Authenticate Request

2.2.2.1.1 Request Code

```
REQCODE = '\'
```

2.2.2.1.2 PARAM_STRING

The following is the PARAM_STRING that is specific for this request.

```
PARAM_STRING = ""REQUEST=\;STATE="" FLAGS "";""
```

2.2.2.2 Authenticate response

There is no **RESPONSEDATA** element for the Authenticate request.

2.2.2.3 Authentication sequence

Unlike other requests, Authentication is not limited to the single request/response exchange. Additional messages are being sent after the request/response exchange occurs.

1. Client sends STARTSSPI message.
2. Client sends CLIENTSSPI message.
3. If the authentication is completed, go to step 6.
4. Server sends SERVERSSPI message.
5. Go to step 2.
6. Server sends ENDSSPI message.

2.2.2.3.1 STARTSSPI

```
STARTSSPI =
OPEN(391)
STRING(392)
CLOSE
```

Value	Meaning
Open(391)	
STRING(392)	Constant ""NTLM""
Close	

2.2.2.3.2 CLIENTSSPI

```
CLIENTSSPI =

OPEN(348)
ARRAY(349) ; SSP package
CLOSE
```

Value	Meaning
Open(348)	A common data structure that indicates the beginning of the element.
ARRAY(349)	A byte array that contains the data that is returned by the **InitializeSecurityContext** SSPI function.
Close	A common data structure that indicates the end of the element.

2.2.2.3.3 SERVERSSPI

```
SERVERSSPI =
OPEN(350)
```

```
INT32(545)
[STRING(450)]
INT32(351
ARRAY(352)
INT32(354)
CLOSE
```

Value	Meaning
Open(350)	A common data structure that indicates the beginning of the element.
INT32(545)	A Boolean that indicates whether the user name is sent in this iteration.
STRING(450)	If the value of **INT32(545)** is 1, this string contains the user name.
INT32(351)	A value that indicates authentication status. If SSPI iterations are ongoing or complete and there is no error, this value is 1. Any other value indicates that authentication failed.
ARRAY(352)	A byte array that contains the data that is returned by the **AcceptSecurityContext** SSPI function. This structure is present if the value of **INT32(351)** is 1.
INT32(354)	If the value of **INT32(351)** is not 1, this value is present and **INT32(351)** has the same meaning as **RESPONSESTATUS** from section 2.2.1.3. **INT32(354)** has the same meaning as **RESPONSEERRORCODE** from section 2.2.1.3.
Close	A common data structure that indicates the end of the element.

2.2.2.3.4 ENDSSPI

```
ENDSSPI =
OPEN(350)
STRING(450)
INT32(351)
[INT32(354)]
CLOSE
```

Value	Meaning
Open(350)	A common data structure that indicates the beginning of the element.
STRING(450)	A string that contains the user name.
INT32(351)	This value has the same meaning as **RESPONSESTATUS** from section 2.2.1.3.
INT32(354)	If the value of **INT32(351)** is not 1, this value is present and has the same meaning as **RESPONSEERRORCODE** from section 2.2.1.3.
Close	A common data structure that indicates the end of the element.

2.2.3 Handshake

This request establishes common settings between client and server.

2.2.3.1 Handshake Request

See section 4.2.1 for an example.

2.2.3.1.1 Request Code

```
REQCODE = '|'
```

2.2.3.1.2 PARAM_STRING

The following is the PARAM_STRING that is specific for this request.

```
PARAM_STRING = ""REQUEST=|;STATE="" FLAGS "";""
```

2.2.3.1.3 REQDATA

The REQDATA part of the request is as follows.

```
REQDATA =
  OPEN(202)
          Array (203, 11)
          Int32(204)
          Int32(205)
          Int32(549)
          Int32(251)
          Int32(253)
          Int32(419)
          Int32(369)
          Int32(325)
          String (287)
          Int32(425)
          Int32(569)
          Int32(570)
  CLOSE
```

Element	Description
Open(202)	A common data structure that indicates the beginning of the element.
Int32(204)	This value is a constant. It MUST be 257.
Int32(205)	This value is a constant. It MUST be 130.
Int32(549)	This value indicates whether a client is 64-bit. If the value is 1, the client is 64-bit.
Int32(251)	A value that specifies the ""Large Level Threshold"" connection string property if one was specified by the user. If the user did not specify this property, this value is 0.
Int32(253)	This value is a constant. It MUST be 0.
Int32(419)	This value is a constant. It MUST be 0.

Element	Description
Int32(369)	A value that represents the Client Locale ID (LCID) of the client as specified in [MS-LCID].
Int32(325)	This value indicates a transport to be used for the protocol, as follows: 5 - TCP/IP 2 - HTTP
String (287)	This string contains the value of the ""Roles"" connection string parameter that is specified by the user.
Int32(425)	This enum value indicates the authentication status, as follows: 0 - No authentication request has been made. 1 - There will be no authentication; the user is an anonymous user. 2 - Authentication has already occurred.
Int32(569)	This value is a constant. It MUST be 0.
Int32(570)	This value is a constant. It MUST be 1.
Close	A common data structure that indicates the end of the element.

2.2.3.2 Handshake Response

See section 4.2.2 for an example.

```
RESPONSEDATA =
  OPEN(206)
          Int32(207)
          Int32(208)
          Int32(209)
          Int32(210)
          Int32(211)
          Int32(212)
          Int32(213)
          Int32(214)
          Int32(550)
          Int32(566)
          Int32(573)
          Int32(574)
          Int32(576)
          Int32(575)
          Int32(588)
          String(422)
          Int32(215)
          Int32(216)
          Int32(217)
          Int32(239)
          Int32(424)
          String(240)
  CLOSE
```

Element	Description
Open(206)	A common data structure that indicates the beginning of the element.

Element	Description
Int32(207)	This value is a constant. It MUST be 0x0239.
Int32(208)	This value is a constant. It MUST be 1.
Int32(209)	This value is a constant. It MUST be 257.
Int32(210)	This value is a constant. It MUST be 130.
Int32(211)	This value is a constant. It MUST be 0.
Int32(212)	This value is a constant. It MUST be 0.
Int32(213)	This value is a constant. It MUST be 0.
Int32(214)	This value is a constant. It MUST be 0.
Int32(550)	If the server is 64-bit, this value is 1. If the server is 32-bit, this value is 0.
Int32(566)	This value is a constant. It MUST be 1.
Int32(573)	This value is a constant. It MUST be 1.
Int32(574)	This value is a constant. It MUST be 1460.
Int32(576)	This value is a constant. It MUST be 0.
Int32(575)	This value is a constant. It MUST be 0.
Int32(588)	This value is a constant. It MUST be 1.
String(422)	This value indicates the product version string (for example, 8.0.2544).
Int32(215)	This value indicates the Language Code Identifier as specified in [MS-LCID]. This Language Code Identifier is used when a server performs a string compare.
Int32(216)	This value indicates the flag that is used for a case-sensitive string compare on the server side. The default value is 0.
Int32(217)	This value indicates the flag that is used for a case-insensitive string compare on the server side. The default value is 0x30001. The flags are as follows: NORM_IGNORECASE 0x00000001 NORM_IGNORENONSPACE 0x00000002 NORM_IGNORESYMBOLS 0x00000004 NORM_IGNOREKANATYPE 0x00010000 NORM_IGNOREWIDTH 0x00020000 SORT_STRINGSORT 0x00001000 NORM_PREFIX 0x80000000 NORM_PADSPACE 0x08000000 For more information about these flags, see [MSFT-JET].
Int32(239)	This value indicates the server edition, as follows: 2 - Standard Edition 3 - Enterprise Edition

Element	Description
Int32(424)	This value indicates the authentication status, as follows: 0 - No authentication request has been made. 1 - There will be no authentication; the user is an anonymous user. 2 - Authentication has been already performed.
String(240)	This value indicates the user name.
Close	A common data structure that indicates the end of the element.

2.2.4 Get Database Collection

The database collection is a set of one or more OLAP databases that are available on the server. This protocol includes a client request structure and a server response structure for these databases.

2.2.4.1 Get Database Collection Request

The client issues a **Get Database Collection** request to retrieve a list of the available databases from the server.

2.2.4.1.1 Request Code

REQCODE = 'G'

2.2.4.1.2 PARAM_STRING

The following is the PARAM_STRING that is specific for this request.

```
PARAM_STRING =
""REQUEST=G;STATE="" FLAGS "";TYPE=B;LAST=Y;""
```

2.2.4.2 Get Database Collection Response

The server response describes the OLAP databases that are available on the server. The database collection contains a collection of **DB** elements, each of which describes an individual OLAP database that is available on the server.

The syntax is defined as follows.

```
RESPONSEDATA =
     OPEN(102)
     INT32(103)
     *DB
     CLOSE
```

Element	Description
OPEN(102)	A common data structure that indicates the beginning of the element.

Element	Description
INT32(103)	A value that specifies the count of databases in the collection.
DB	A data structure that describes an individual OLAP database. This data structure repeats for each OLAP database that is available on the server.
CLOSE	A common data structure that indicates the end of the element.

2.2.4.2.1 DB

The **DB** data structure describes an individual OLAP database. This data structure repeats for each OLAP database that is available on the server.

The syntax is defined as follows.

```
DB      =
OPEN(101)

     Object
     INT32(222)
     INT32(226)
     INT64(236)
     LockObject

CLOSE
```

Element	Description
OPEN(101)	A common data structure that indicates the beginning of the element.
Object	A common data structure that contains metadata that describes the database.
INT32(222)	A value that specifies the version of the database.
INT32(226)	A value that specifies the **commit version** of the database.
INT64(236)	A value that specifies the size, in kilobytes, of the database on disk.
LockObject	A common data structure for use by the Lock Manager.
CLOSE	A common data structure that indicates the end of the element.

2.2.5 Get Database

The client can request information about a specific database. This section describes the structure of the client request and the server response.

2.2.5.1 Get Database Request

The client issues a request to retrieve information about the specified database from the server. The **Get Database** request is a request for information about an individual OLAP database on the server.

2.2.5.1.1 Request Code

```
REQCODE   = 'G'
```

2.2.5.1.2 PARAM_STRING

The following is the PARAM_STRING that is specific for this request.

```
PARAM_STRING =
""REQUEST=G;STATE="" FLAGS "";"" DBREF
```

2.2.5.2 Get Database Response

The **Get Database** response describes an individual OLAP database that is available on the server. This response contains a series of elements, each of which describes particular features of the specified database. The syntax is defined as follows.

```
RESPONSEDATA =
OPEN(101)
     Object
     INT32(222)
     INT32(226)
     INT64(236)
     LockObject
     [CUBE]
     [AUX_ARRAY]
     [CUBE_DIMENSION_ARRAY]
     [ROLE_ARRAY]
     DB_SECURITY
     CUBE_ARRAY
CLOSE
```

Element	Description
OPEN(101)	A common data structure that indicates the beginning of the element.
Object	A common data structure that contains metadata that describes the object.
INT32(222)	A value that specifies the version of the base cube.
INT32(226)	A value that specifies the commit version of the database.
INT64(236)	A value that specifies the size, in kilobytes, of the database on disk.
LockObject	A common data structure for use by the Lock Manager.
[CUBE]	A data structure that contains information about the base cube in the database. This data structure is returned if **INT32REQFLAGS** satisfies the following conditions: Bit 0x00000001 is set.

Element	Description
	Bit 0x00008000 is not set.
[AUX_ARRAY]	An array of data structures that contain information about additional objects in the database. This data structure is returned if **INT32REQFLAGS** satisfies the following conditions: Bit 0x00000001 is set. Bit 0x00008000 is not set.
[CUBE_DIMENSION_ARRAY]	An array of data structures that contain information about the database dimensions. This data structure is returned if **INT32REQFLAGS** satisfies the following conditions: Bit 0x00000001 is set. Bit 0x00008000 is not set.
[ROLE_ARRAY]	An array of data structures that contain information about the database security roles. This data structure is returned if **INT32REQFLAGS** satisfies the following condition: Bit 0x00000001 is set. Bit 0x00008000 is not set.
DB_SECURITY	A data structure that contains information about the security settings of the database.
CUBE_ARRAY	A data structure that contains information about the cubes in the database.
CLOSE	A common data structure that indicates the end of the element.

2.2.5.2.1 CUBE

The CUBE data structure contains information about the cubes in the database. The syntax is defined as follows:

```
CUBE=                          ;
  OPEN(94)
  ARRAY_CUBE                        ;
  INT32(234)                        ;
  [CUBE_DIMENSION_ARRAY_DETAILS]    ;
  CLOSE
```

Element	Description
OPEN(94)	A common data structure that indicates the beginning of the element.
ARRAY_CUBE	An array of data structures that contain information about the cube.
INT32(234)	Always returns 0.

Element	Description
[CUBE_DIMENSION_ARRAY_DETAILS]	An array of data structures that contain information about the dimensions defined in the cube. This data structure is returned if INT32REQFLAGS satisfies the following condition: Bit 0x00000001 is set Bit 0x20000000 is set
CLOSE	A common data structure that indicates the end of the element.

2.2.5.2.2 ARRAY_CUBE

The syntax of **ARRAY_CUBE** is defined as follows.

```
ARRAY_CUBE=
OPEN(85)
OBJECT
LOCKOBJECT
INT32(235) ;
INT32(490) ;
INT32(530) ;
INT64(237) ;
INT32(577) ;
```

The **THE_REST_OF_ARRAY_CUBE** data structure is returned if **INT32REQFLAGS** satisfies the following condition:

Bit 0x00000001 is set.

The syntax of **THE_REST_OF_ARRAY_CUBE** is defined as follows.

```
THE_REST_OF_ARRAY_CUBE =
INT32(86)
INT32(87);
INT32(88);
INT32(89) ;
INT32(90);
INT32(390)
INT32(395)
INT32(396)
INT32(91)
INT32(92)
INT32(93)
STRING(547)
STRING(548)
[MEASURE_GROUP_ARRAY]
[AUX_ARRAY]
INT32(386)
CLOSE
```

Element	Description
OPEN(85)	A common data structure that indicates the beginning of the element.
OBJECT	A common data structure that contains metadata that describes the cube.
LOCKOBJECT	A common data structure for use by the Lock Manager.
INT32(235)	This value indicates that the server sends back the same **INT32REQFLAGS** that it received.
INT32(490)	A value that specifies whether the cube is write-enabled.
INT32(530)	A value that specifies the safety options that are enabled on the cube.
INT64(237)	A value that specifies the size of the cube on disk.
INT32(577)	A value that specifies the estimate of the size of the cube, expressed as number of rows in the **fact table**.
[INT32(86)]	A value that specifies the count of the number of dimensions in the cube.
[INT32(87)]	A value that specifies the count of the number of levels in the cube.
[INT32(88)]	A value that specifies the count of the number of details in the cube.
[INT32(89)]	A value that specifies the count of the number of measures in the cube.
[INT32(90)]	A value that specifies the count of the number of auxiliary elements in the cube.
[INT32(390)]	A value that specifies whether unique name generation is enabled on the cube.
[INT32(395)]	A value that specifies MDX compatibility for the cube.
[INT32(396)]	A value that specifies SQL compatibility for the cube.
[INT32(91)]	A value that specifies the version of the cube.
[INT32(92)]	A value that specifies the dimension version of the cube.
[INT32(93)]	A value that specifies the cache version of the cube.
[STRING(547)]	A value that specifies the link source for the cube.
[STRING(548)]	A value that specifies the link cube for the cube.
[MEASURE_GROUP_ARRAY]	An array of data structures that contain information about the measure groups in the cube.
[AUX_ARRAY]	An array of data structures that contain information about the auxiliary elements in the cube.
[INT32(386)]	A value that specifies the count of **partitions** in the cube.
CLOSE	A common data structure that indicates the end of the element.

2.2.5.2.2.1 INT32(530) – SAFETY_OPTIONS

This value defines default safety options of the cube.

Note For more information, see [MSDN-SafetyOptions].

Symbolic name	Value	Description
DBPROPVAL_MSMD_SAFETY_OPTIONS_DEFAULT	0	For connections through HTTP or HTTPS, this value is the same as DBPROPVAL_MSMD_SAFETY_OPTIONS_ALLOW_SAFE. For all other connections, this value is the same as DBPROPVAL_MSMD_SAFETY_OPTIONS_ALLOW_ALL.
DBPROPVAL_MSMD_SAFETY_OPTIONS_ALLOW_ALL	1	This value enables all user-defined function libraries without verifying that they are safe for initialization and scripting.
DBPROPVAL_MSMD_SAFETY_OPTIONS_ALLOW_SAFE	2	This value indicates that all classes for a particular user-defined function library are checked to make sure that they are safe for initialization and scripting.
DBPROPVAL_MSMD_SAFETY_OPTIONS_ALLOW_NONE	3	This value prevents user-defined functions from being used during the session.

2.2.5.2.2.2 CUBE_SECURITY

The syntax is defined as follows.

```
CUBE_SECURITY=
  OPEN(489)
  INT16(495);
  CUBE_SECURITY_DETAILS
  CLOSE
```

Element	Description
OPEN(489)	A common data structure that indicates the beginning of the element.
INT16(495)	A value that specifies whether cube security is enabled. A value of 1 indicates that cube security is enabled.
CUBE_SECURITY_DETAILS	A data structure that describes the security features that are enabled on the cube.
CLOSE	A common data structure that indicates the end of the element.

2.2.5.2.2.2.1 CUBE SECURITY DETAILS

The syntax is defined as follows.

```
CUBE_SECURITY_DETAILS=
  OPEN(259)
  INT32(498);
  INT32(260);
  INT32(285);
  INT32(353);
  INT32(499);
  INT32(500);
  INT32(359);
  INT32(261);
  CELL_SECURITY_ARRAY
  CUBE_DIMENSION_SECURITY
  CLOSE
```

Element	Description
OPEN(259)	A common data structure that indicates the beginning of the element.
INT32(498)	A Boolean that specifies whether cube security is enabled.
INT32(260)	A value that specifies the level of access (read or read-write) that is allowed to the cube, as follows: 0 – None 1 – Read 3 – Read/Write
INT32(285)	A value that specifies whether security is to be enforced on the server or on the client, as follows: 0 – None (client enforces) 1 – Server enforces
INT32(353)	A value that specifies whether drillthrough is allowed on the cube, as follows: 1 – Drillthrough is allowed 0 – Drillthrough is not allowed
INT32(499)	A Boolean that specifies whether linking to the cube is allowed.
INT32(500)	A value that specifies whether SQL queries to this cube are allowed, as follows: 1 – SQL queries are allowed 0 – SQL queries are not allowed
INT32(359)	This value is a constant. It MUST be 0.
INT32(261)	A value that specifies the count of cell security statements that are defined in the cube.
CELL_SECURITY_ARRAY	An array of data structures that contains information about the cell security statements that are defined in the cube.
CUBE_DIMENSION_SECURITY	An array of data structures that contains information about the security features that are enabled on the dimensions that are defined in the cube.
CLOSE	A common data structure that indicates the end of the element.

2.2.5.2.2.2.2 ROLE_COMMAND_ARRAY

The syntax is defined as follows.

```
ROLE_COMMAND_ARRAY=
   *STRING(360) ;
```

Element	Description
STRING(360)	An array of strings that contain role commands.

2.2.5.2.2.2.3 CUBE_SECURITY_ARRAY

The syntax of **CUBE_SECURITY_ARRAY** is defined as follows.

```
CUBE_SECURITY_ARRAY=
  OPEN(265)
  INT32(266);
  *ARRAY_CUBE_SECURITY
  CLOSE
```

Element	Description
OPEN(265)	A common data structure that indicates the beginning of the element.
INT32(266)	A value that specifies the count of cube security elements.
ARRAY_CUBE_SECURITY	An array of data structures that contain information about cube security elements.
CLOSE	A common data structure that indicates the end of the element.

2.2.5.2.2.3 CUBE_DIMENSION_SECURITY

The syntax of **CUBE_DIMENSION_SECURITY** is defined as follows.

```
CUBE_DIMENSION_SECURITY=
  OPEN(280)
  READ_CUBE_DIMENSION_ARRAY
  WRITE_CUBE_DIMENSION_ARRAY
  MEASURE_SECURITY_ARRAY
  CLOSE
```

Element	Description
OPEN(280)	A common data structure that indicates the beginning of the element.
READ_CUBE_DIMENSION_ARRAY	An array of data structures that contain information about the read-only dimensions that are defined in the cube.
WRITE_CUBE_DIMENSION_ARRAY	An array of data structures that contain information about the read/write dimensions that are defined in the cube.

Element	Description
MEASURE_SECURITY_ARRAY	An array of data structures that contain information about the security features that are enabled on measures that are defined in the cube.
CLOSE	A common data structure that indicates the end of the element.

2.2.5.2.2.4 READ_CUBE_DIMENSION_ARRAY

The syntax of **READ_CUBE_DIMENSION_ARRAY** is defined as follows.

```
READ_CUBE_DIMENSION_ARRAY=
  OPEN(281)
  INT32(282)
  *DIMENSION_SECURITY
  CLOSE
```

Element	Description
OPEN(281)	A common data structure that indicates the beginning of the element.
INT32(282)	A value that specifies the count of read-security dimension roles that are defined in the cube.
DIMENSION_SECURITY	A data structure that specifies dimension security for a specific dimension. This structure is repeated INT32(282) times.
CLOSE	A common data structure that indicates the end of the element.

2.2.5.2.2.5 WRITE_CUBE_DIMENSION_ARRAY

The syntax **WRITE_CUBE_DIMENSION_ARRAY** is defined as follows.

```
WRITE_CUBE_DIMENSION_ARRAY=
  OPEN(283)
  INT32(284); the count of the number of write dimension security
 *DIMENSION_SECURITY
  CLOSE
```

Element	Description
OPEN(283)	A common data structure that indicates the beginning of the element.
INT32(284)	A value that specifies the count of read/write dimensions that are defined in the cube.
DIMENSION_SECURITY	A data structure that specifies dimension security for a specific dimension. This structure is repeated INT32(282) times.
CLOSE	A common data structure that indicates the end of the element.

2.2.5.2.2.6 DIMENSION_SECURITY

The syntax of **DIMENSION_SECURITY** is as follows.

```
DIMENSION_SECURITY =
   OPEN(268)
    INT32(324)
    STRING(269)
    STRING(270)
    STRING(271)
    STRING(272)
    STRING(273)
    STRING(274)
    Int32(275)
    Int32(276)
    Int32(277)
    Int32(278)
    Int32(279)
    Int32(488)
    Int32(323)
    DIMENSION_SECURITY_REGION_ARRAY
  CLOSE
```

Element	Description
OPEN(268)	A common data structure that indicates the beginning of the element.
INT32(324)	A bitmask value that defines how the server is treating cube security roles, as follows: 0x80000000 – The server has computed a role based on all the role membership for the user. 0x40000000 – This value is the original role for the user as declared within **Decision Support Objects (DSO)**.
STRING(269)	This value is the default member condition. It is returned if **INT32(324)** satisfies one of the following conditions: 0x80000000 = 1 0x40000000 = 1
STRING(270)	This value is the high level of dimension security. It is returned if **INT32(324)** satisfies the following condition: 0x40000000 = 1
STRING(271)	This value is the low level of dimension security. It is returned if **INT32(324)** satisfies the following condition: 0x40000000 = 1
STRING(272)	This value is the visual level of dimension security. It is returned if **INT32(324)** satisfies the following condition: 0x40000000 = 1
STRING(273)	This value is the allowed set condition. It is returned if **INT32(324)** satisfies the following condition: 0x40000000 = 1

Element	Description
STRING(274)	This value is the deny set condition. It is returned if **INT32(324)** satisfies the following condition: 0x40000000 = 1
Int32(275)	This value is the hidden condition. It is returned if **INT32(324)** satisfies the following condition: 0x40000000 = 1
Int32(276)	This value is the security type, as follows: 0x00000000 – None 0x00000001 – Read 0x00000002 - Write
Int32(277)	This value is the high level of dimension security. It is returned if **INT32(324)** satisfies the following condition: 0x80000000 = 1
Int32(278)	This value is the low level of dimension security. It is returned if **INT32(324)** satisfies the following condition: 0x80000000 = 1
Int32(279)	This value is the visual level of dimension security. It is returned if **INT32(324)** satisfies the following condition: 0x80000000 = 1
Int32(488)	This value is the deepest level of any allow or deny. It is returned if **INT32(324)** satisfies the following condition: 0x80000000 = 1
Int32(323)	This value is the offset of this DPath within the path of DataID. It is returned if **INT32(324)** satisfies the following condition: 0x80000000 = 1
DIMENSION_SECURITY_REGION_ARRAY	An array of data structures that contain information about dimension security regions. It is returned if **INT32(324)** satisfies the following condition: 0x80000000 = 1
CLOSE	A common data structure that indicates the end of the element.

2.2.5.2.2.7 DIMENSION_SECURITY_REGION_ARRAY

The syntax of **DIMENSION_SECURITY_REGION_ARRAY** is defined as follows.

```
DIMENSION_SECURITY_REGION_ARRAY =
    OPEN(316)
    INT32(317) ;
    *ARRAY_DIMENSION_SECURITY_REGION
    CLOSE
```

Element	Description
OPEN(316)	A common data structure that indicates the beginning of the element.
INT32(317)	A value that specifies the count of dimension security regions.
ARRAY_DIMENSION_SECURITY_REGION	An array of data structures that contain information about dimension security regions.
CLOSE	A common data structure that indicates the end of the element.

2.2.5.2.2.7.1 ARRAY_DIMENSION_SECURITY_REGION

The syntax of **ARRAY_DIMENSION_SECURITY_REGION** is defined as follows.

```
ARRAY_DIMENSION_SECURITY_REGION=
   INT32(318)
   INT32(319)
   ALLOW_AXIS
   ALLOW_AXIS_TUPLES
   DENY_AXIS    DENY_AXIS_TUPLES
```

Element	Description
INT32(318)	If this value is not 0, an **allow axis** is defined. If the value is non-0, it is an internal pointer of no interpretation to the client.
INT32(319)	When this value is not 0, a deny axis is defined. If the value is non-0, it is an internal pointer of no interpretation to the client.
ALLOW_AXIS	This element has a value when the value of the **INT32(318)** element is not 0.
ALLOW_AXIS_TUPLES	This element has a value when the value of the **INT32(318)** element is not 0.
DENY_AXIS	This element has a value when the value of the **INT32(319)** element is not 0.
DENY_AXIS_TUPLES	This element has a value when the value of the **INT32(319)** element is not 0.

2.2.5.2.2.8 ALLOW_AXIS

The syntax of **ALLOW_AXIS** is defined as follows.

```
ALLOW_AXIS=
    AXIS
```

Element	Description
AXIS	A data structure that contains information about an axis.

2.2.5.2.2.9 AXIS

The syntax of **AXIS** is defined as follows.

```
AXIS=
   OPEN(156)
   INT32(157) ;
   *AXIS_DIMENSION
   CLOSE
```

Element	Description
OPEN(156)	A common data structure that indicates the beginning of the element.
INT32(157)	A value that specifies the count of dimensions on this axis.
AXIS_DIMENSION	An array of data structures that contain information about a dimension on this axis.
CLOSE	A common data structure that indicates the end of the element.

2.2.5.2.2.10 AXIS_DIMENSION

The syntax of **AXIS_DIMENSION** is defined as follows.

```
AXIS_DIMENSION=
   OPEN(158)
   INT16(159)
   CLOSE
```

Element	Description
OPEN(158)	A common data structure that indicates the beginning of the element.
INT16(159)	An index of the dimension within the cube.
CLOSE	A common data structure that indicates the end of the element.

2.2.5.2.2.11 ALLOW_AXIS_TUPLES

The syntax of **ALLOW_AXIS_TUPLES** is defined as follows.

```
ALLOW_AXIS_TUPLES=
    AXIS_TUPLES
```

Element	Description
AXIS_TUPLES	An array of data structures that contain information about the tuples on the allow axis.

2.2.5.2.2.12 AXIS_TUPLES

The syntax of **AXIS_TUPLES** is defined as follows.

```
AXIS_TUPLES=
    OPEN(160)
    INT32(161) ;
    INT32(162) ;
    INT32(163) ;
    INT32(164) ;
    INT32(169) ;
     *TuplePage
    CLOSE
```

Element	Description
OPEN(160)	A common data structure that indicates the beginning of the element.
INT32(161)	A value that specifies the count of tuples that will follow in the **TuplePage**.
INT32(162)	Actual size of the page, in bytes. This value is adjusted by the server from a base default value.
INT32(163)	Size of **TPath** in bytes.
INT32(164)	Number of actual **records**, based on the adjusted page size in **INT32(162)**.
INT32(169)	This value is always 0.
TuplePage	A binary blob that contains the tuples.
CLOSE	A common data structure that indicates the end of the element.

2.2.5.2.2.13 TuplePage

The **TPath** elements follow one after the other in this object.

```
TuplePage =
  *TPath
```

Element	Description
TPath	The set of DPATHs that form a tuple.

2.2.5.2.2.14 DENY_AXIS

The syntax of **DENY_AXIS** is defined as follows.

```
DENY_AXIS=
    AXIS
```

Element	Description
AXIS	A data structure that contains information about an axis.

2.2.5.2.2.15 DENY_AXIS_TUPLES

The syntax of **DENY_AXIS_TUPLES** is defined as follows.

```
DENY_AXIS_TUPLES=
   AXIS_TUPLES
```

Element	Description
AXIS_TUPLES	An array of data structures that contain information about the tuples on the allow axis.

2.2.5.2.2.16 MEASURE_SECURITY_ARRAY

The syntax of **MEASURE_SECURITY_ARRAY** is defined as follows.

```
MEASURE_SECURITY_ARRAY=
  OPEN(341)
  INT8(342)
  CLOSE
```

Element	Description
OPEN(341)	A common data structure that indicates the beginning of the element.
INT8(342)	A value that specifies whether measure security is defined in the cube.
CLOSE	A common data structure that indicates the end of the element.

2.2.5.2.2.17 CUBE_DIMENSION_ARRAY_DETAILS

The syntax of **CUBE_DIMENSION_ARRAY_DETAILS** is defined as follows.

```
CUBE_DIMENSION_ARRAY_DETAILS=
  OPEN(95)
  INT32(96)
*CUBE_DIMENSION
  CLOSE
```

Element	Description
OPEN(95)	A common data structure that indicates the beginning of the element.
INT32(96)	For a database, a value that specifies the number of dimensions in the database. For a cube. a value that specifies the number of dimensions in the cube.
CUBE_DIMENSION	An array of data structures that contain information about dimensions in the cube. This data structure is returned if **INT32REQFLAGS** satisfies the following condition: Bit 0x20000000 is set
CLOSE	A common data structure that indicates the end of the element.

2.2.5.2.2.18 CUBE_SQL_ARRAY

The syntax of **CUBE_SQL_ARRAY** is defined as follows.

```
CUBE_SQL_ARRAY=;
  OPEN(98)
  SQL_SOURCE_ARRAY
  CLOSE
```

Element	Description
OPEN(98)	A common data structure that indicates the beginning of the element.
SQL_SOURCE_ARRAY	An array of data structures that contain information about SQL data sources.
CLOSE	A common data structure that indicates the end of the element.

2.2.5.2.2.18.1 SQL_SOURCE_ARRAY

The syntax of **SQL_SOURCE_ARRAY** is defined as follows.

```
SQL_SOURCE_ARRAY=
  OPEN(81)
  INT32(82)
  *ARRAY_SQL
  CLOSE
```

Element	Description
OPEN(81)	A common data structure that indicates the beginning of the element.
INT32(82)	A value that specifies the number of data sources that are defined in the cube.
ARRAY_SQL	An array of data structures that contain information about the data sources that are defined in the cube.
CLOSE	A common data structure that indicates the end of the element.

2.2.5.2.2.18.2 ARRAY_SQL

The syntax of **ARRAY_SQL** is defined as follows.

```
ARRAY_SQL=
  OPEN(80)
  OBJECT
  SQL_DATA_SOURCE
  CLOSE
```

Element	Description
OPEN(80)	A common data structure that indicates the beginning of the element.

Element	Description
OBJECT	A common data structure that contains metadata that describes the object.
SQL_DATA_SOURCE	An array of data structures that contain the data sources that are defined in the cube.
CLOSE	A common data structure that indicates the end of the element.

2.2.5.2.2.18.3 SQL_DATA_SOURCE

The syntax of **SQL_DATA_SOURCE** is defined as follows.

```
SQL_DATA_SOURCE=
  OPEN(55)
  STRING(56)
  STRING(57)
  STRING(58)
  STRING(59)
  INT32(60)
  INT32(61)
  REAL64(62)
  CLOSE
```

Element	Description
OPEN(55)	A common data structure that indicates the beginning of the element.
STRING(56)	A value that specifies the connection string such as the following: " Provider=MSDASQL.1;Persist Security Info=False; Data Source=FoodMart 2000"
STRING(57)	A value that specifies the data source name (DSN) for the data source.
STRING(58)	A value that specifies the User ID to be used to connect to the data source.
STRING(59)	A value that specifies the password to be used to connect to the data source.
INT32(60)	A value that specifies the state of the data source.
INT32(61)	A value that specifies the time-out interval (in seconds) to be used when connecting to the data source.
REAL64(62)	A value that specifies the timestamp for the data source. See section 2.2.1.5.1.1 for documentation of the storage format for the **Date** data type.
CLOSE	A common data structure that indicates the end of the element.

The **INT32(60)** bitmask field MUST have a combination of zero or more of the following values.

Element	Description
0x10000000	Initialize OLE when the data source is initialized.
0x20000000	Reset OLE when the data source is reset.
0x40000000	Initialize the connection from the provider string.

Element	Description
0x80000000	Reset the connection if the date changes.
0x01000000	Refresh the data source on error.
0x02000000	The data source has a password string.
0x00100000	The data source is connected.
0x00400000	Connected to external **rowset**.
0x00800000	The data source is queried.
0x00010000	Read-only connection.
0x00020000	Prepare-only connection.
0x00040000	The connection is safe (unsafe connect properties are ignored).
0x00080000	Do not use pooled data sources.
0x00001000	Fast OLEDB loader.

2.2.5.2.3 AUX_ARRAY

The syntax for **AUX_ARRAY** is defined as follows.

```
AUX_ARRAY=
  OPEN(78)
  INT32(79)
  *CUBE_AUXILIARY
  CLOSE
```

Element	Description
OPEN(78)	A common data structure that indicates the beginning of the element.
INT32(79)	A value that specifies the count of the auxiliary objects in the cube.
*CUBE_AUXILIARY	An array of data structures that provide auxiliary information about the cube.
CLOSE	A common data structure that indicates the end of the element.

2.2.5.2.3.1 CUBE_AUXILIARY

The syntax of **CUBE_AUXILIARY** is defined as follows.

```
CUBE_AUXILIARY=
  OPEN(77)
  OBJECT
  [AUX]
  CLOSE
```

Element	Description
OPEN(77)	A common data structure that indicates the beginning of the element.
OBJECT	A common data structure that contains metadata that describes the auxiliary element.
[AUX]	A data structure that contains information about the auxiliary element. This data structure is returned if **INT32REQFLAGS** satisfies the following condition: Bit 0x00000001 is set.
CLOSE	A common data structure that indicates the end of the element.

2.2.5.2.4 CUBE_DIMENSION_ARRAY

The syntax is defined as follows:

```
CUBE_DIMENSION_ARRAY=
  OPEN(70)
  INT32(71)
  *ARRAY_DIMENSION
  CLOSE
```

Element	Description
OPEN(70)	A common data structure that indicates the beginning of the element.
INT32(71)	A value that specifies the number of dimensions in the cube.
ARRAY_DIMENSION	An array of data structures that contain information about the dimensions defined in the cube.
CLOSE	A common data structure that indicates the end of the element.

2.2.5.2.4.1 ARRAY_DIMENSION

The syntax is defined as follows.

```
ARRAY_DIMENSION=
  OPEN(68)
  OBJECT
  LOCKOBJECT
  DIMENSION
  INT32(69)
  *LEVEL_ARRAY
  [CUBE_TREE]
  [CustomMemberFormulas]
  CLOSE
```

Element	Description
OPEN(68)	A common data structure that indicates the beginning of the element.
OBJECT	A common data structure containing metadata that describes the object.

Element	Description
LOCKOBJECT	A common data structure for use by the Lock Manager.
DIMENSION	A data structure that contains information about a dimension in the cube.
INT32(69)	A value that specifies the number of levels defined in the dimension.
LEVEL_ARRAY	An array of data structures that contain information about the levels in the dimension. This data structure is returned if INT32REQFLAGS satisfies the following condition: Bit 0x00000001 is set
[CUBE_TREE]	Structure that contains information about dimensions defined in the cube. This data structure is returned if INT32REQFLAGS satisfies the following condition: Bit 0x00000001 is set Bit 0x40000000 is set
[CustomMemberFormulas]	Structure that contains information about custom member formulas defined in the dimension. This data structure is returned if INT32REQFLAGS satisfies the following condition: Bit 0x00000001 is set
CLOSE	A common data structure that indicates the end of the element.

2.2.5.2.4.1.1 LEVEL_ARRAY

The syntax is defined as follows:

```
LEVEL_ARRAY=
  *ARRAY_LEVEL
```

Element	Description
ARRAY_LEVEL	An array of data structures that contain information about the levels defined in the dimension.

2.2.5.2.4.1.2 ARRAY_LEVEL

The syntax is defined as follows:

```
ARRAY_LEVEL=
  OBJECT
  [LEVEL]
```

Element	Description
OBJECT	A common data structure containing metadata that describes the object.

Element	Description
[LEVEL]	An array of data structures that contain information about the levels defined in the dimension. This data structure is returned if INT32REQFLAGS satisfies the following condition: Bit 0x00000001 is set

2.2.5.2.4.2 CUBE_TREE

This section specifies the data structure of a dimension tree. The dimension tree data structure includes a single prefix used to identify members of the same dimension tree, a repeating data structure for each member in the dimension tree, and a mark to indicate the last member in the tree. Note that the object called CUBE_TREE in this ABNF in fact represents what is, to the OLAP end user, a dimension tree.

The message is a binary data structure that is transmitted from the server to the client.

The syntax is defined as follows:

```
CUBE_TREE =
     OPEN(126)
     [DVertex]
     *EVertex
     INT32(105)
     CLOSE
```

Value	Meaning
Open(126)	A common data structure that indicates the beginning of the element.
[DVertex]	Present if Cube_Tree response is part of GetDimensionMembers Response defined in section 2.2.7, it is absent if Cube_Tree is part of GetDatabase Response as defined in section 2.2.5.
EVertex	A dimension member vertex.
INT32(105)	This value MUST be 0.
Close	A common data structure that indicates the end of the element.

2.2.5.2.4.3 DVertex

The DVertex data structure specifies a prefix for dimension members.

The syntax is defined as follows:

```
DVertex:=
INT32(105)
INT32(106)
INT8(107)
INT32(108)
```

Value	Meaning
INT32(105)	A value that specifies the index of the database dimension. Will be the same as INT32(106).

Value	Meaning
INT32(106)	A value that specifies the index of the database dimension. Will be the same as INT32(105).
INT8(107)	A constant ASCII value of 'D' (68). This value MUST be specified.
INT32(108)	A count of the members in the dimension.

2.2.5.2.4.4 EVertex

The **EVertex** data structure specifies a single dimension member. This data structure repeats for each member in the dimension tree.

The syntax is defined as follows.

```
EVertex:=
INT32(105)
INT32(106)
INT8(107)
INT16(112)
INT32(114)
DPATH(115)
STRING(116)
STRING(117)
INT8(404)
INT8(118)
INT8(407)
INT8(119)
INT16(120)
[ARRAY(121)]
INT16(122)
INT16(123)
INT16(124)
INT16(125)
INT32(418)
```

Value	Meaning
INT32(105)	This is the unique index of the member inside the memory store which reflects the creation order of the member during the processing operation.
INT32(106)	Vertex index. Starting from 1 in the tree of EVertexes, it is the index in a preorder traversal of the tree.
INT8(107)	A constant that specifies the ASCII value of 'E' (69). This value MUST be specified.
INT16(112)	A value that specifies the level of the member in the **hierarchy**.
INT32(114)	A value that specifies the DataID.
DPATH(115)	A value that specifies the DPath.
STRING(116)	A value that specifies the member name. This value is empty if the member name is the same as the member key.
STRING(117)	This value MUST be NULL.

Value	Meaning
INT8(404)	A value that specifies whether the member is hidden. If the value is 1, the member is hidden. If the value is 0, the member is visible.
INT8(118)	A bitmask that describes several fields of the dimension member: bit1(0x01). This value MUST be FALSE. bit2(0x02) if this value is set to true, it means the member was sorted by value (key) inside the tree, and not by name. bit4(0x08)if set, this bit specifies whether the member is a hole in a **ragged hierarchy** for **type A hole** (where "A" is the inner side of the hierarchy). bit5(0x10)if set, the member is part of an automatically generated level. Decision Support Objects (DSO) allows the definition of a level that extends the number of children under one parent by introducing another level. The new level is added in between existing levels in an automatic fashion. If a member has this bit set, it means that it is a member of an auto-created level. bit6(0x20)if set, this bit specifies that the dimension member has a custom rollup formula or a custom rollup option associated with it.. bit7(0x40) if set, this bit specifies whether the member is a leaf node in the member tree. If the value is 1, the member is a leaf node in the parent-child hierarchy. If the value is 0, the leaf state is not specified.
INT8(407)	A bitmask that describes additional information about the dimension member: bit1(0x01) if set, this bit specifies whether the member is a terminal hole in a ragged hierarchy. bit2(0x02) if set, this bit specifies whether the member is a data member in a parent-child hierarchy. bit3(0x03) if set, the member has only additive (+) custom rollup operators below or no unary operators. bit4(0x04) if set, this flag indicates that in the subtree rooted by this node, there are no tilde characters set.
INT8(119)	A value that specifies the type of member key. Valid values include: 0x00-NULL 0x01-String 0x02–4-byte integer 0x03–8-byte integer 0x04-FLOAT 0x05-DOUBLE 0x06–64-bit REAL which represents a DATE. See section 2.2.1.5.1.1 which describes storage format of the date. 0x07–64-bit integer which represents Currency. For the storage format of Currency see section 2.2.5.2.4.4.1.
INT16(120)	A value that specifies the size of the member key. Valid values include: 0 to indicate NULL Any length up to maximum integer if the member key type is STRING. If the member key type is not STRING, this value is the size of its primitive type.
[ARRAY(121)]	An array that contains member data. IF INT8(119) = 0 – Nothing is sent. INT8(119) = 0x01 – Byte array is sent.

Value	Meaning
	INT8(119) – all others, this value is a binary representation of the primitive type.
INT16(122)	If the children are sorted by the natural order within the parent structure, this value represents the index this member would get in the natural order.
INT16(123)	If the children are sorted by the natural order within the parent structure, this value represents the current index of this member. The index value reflects the exact position of the child within the sort order.
INT16(124)	If the children are sorted by the key, this value represents the index this member would get given a key sort order.
INT16(125)	If the children are sorted by the key, this value represents the current index. The index value reflects the exact position of the child within the sort order.
INT32(418)	This value MUST be zero.

2.2.5.2.4.4.1 Format for representation of Currency

The CURRENCY data type is an 8-byte signed integer scaled by 10,000. This allows a variable of the CURRENCY type to have a range of:

(2 ^ 63 -1) / 10,000 = +922337203685477.5807

To

(2 ^ 63) / 10,000 = -922337203685477.5808

Up to 19 digits are allowed, with no more than 4 digits to the right of the decimal point.

For more information, see [MSFT-KB51414].

2.2.5.2.5 CustomMemberFormulas

This section specifies the data structure of **CustomMemberFormulas**, a complex data structure that summarizes and defines all custom formulas that are associated with a particular dimension member.

The syntax is defined as follows.

```
CustomMemberFormulas =
     OPEN(333)
     INT32(334)
     INT16(340)
    *CustomMemberFormula
     CLOSE
```

Value	Meaning
Open(333)	Indicates the start of the response.
INT32(334)	A value that specifies the total number of custom member formulas.
INT16(340)	A value that specifies the number of levels in the dimension (excluding disabled

Value	Meaning
	levels).
CustomMemberFormula	A data structure that specifies the custom formula. This data structure repeats for each custom formula used by the member.
Close	Indicates the end of the response.

2.2.5.2.5.1 CustomMemberFormula Data Structure

The **CustomMemberFormula** data structure specifies a formula used by the dimension member.

The syntax is defined as follows.

```
CustomMemberFormula=
INT32(339)
INT32(335)
STRING(336)
INT16(337)
DPATH(338)
STRING(475)
```

Value	Meaning
Open	Indicates the start of the response.
INT32(339)	This value is always 2.
INT32(335)	A value that specifies the **solve order** of the formula.
STRING(336)	A text string that specifies the MDX expression of the formula.
INT16(337)	A value that specifies the level of the dimension member that uses the formula.
DPATH(338)	The DPath of the dimension member that uses the formula.
STRING(475)	A text string that specifies the MDX text for custom rollup options.
Close	Indicates the end of the response.

2.2.5.2.6 ROLE_ARRAY

The syntax is defined as follows:

```
ROLE_ARRAY=
  OPEN(263)
  INT32(264)
  *ARRAY_ROLE
  CLOSE
```

Element	Description
OPEN(263)	A common data structure that indicates the beginning of the element.

Element	Description
INT32(264)	A value that specifies the count of the roles defined in the cube.
*ARRAY_ROLE	An array of data structures that contain information about the roles defined in the cube.
CLOSE	A common data structure that indicates the end of the element.

2.2.5.2.6.1 ARRAY_ROLE

The syntax is defined as follows:

```
ARRAY_ROLE=
  OBJECT
  ROLE
```

Element	Description
OBJECT	A common data structure containing metadata that describes the object.
ROLE	An array of data structures that contain information about the roles defined in the database.

2.2.5.2.6.2 ROLE

The syntax is defined as follows:

```
ROLE=
  OPEN(254)
  INT32(255)
  INT32(256)
  INT32(257)
  ARRAY(258)
  CLOSE
```

Element	Description
OPEN(254)	A common data structure that indicates the beginning of the element.
INT32(255)	A value that specifies the number of Security Identifiers (SIDs) defined in the database. For more information about SIDs, see [MSDN-SID].
INT32(256)	A value that specifies the total size of all SIDs defined in the database in bytes.
INT32(257)	A value that specifies the size of an individual SID defined in the database.
ARRAY(258)	An array of the SIDs defined in the database. Each is of size INT32(257).
CLOSE	A common data structure that indicates the end of the element.

2.2.5.2.7 DB_SECURITY

The syntax is defined as follows:

```
DB_SECURITY=
  OPEN(482)
  INT32(544)
  INT32(546)
  INT32(483)
  *ROLE_NAME_ARRAY
CLOSE
```

Element	Description
OPEN(482)	A common data structure that indicates the beginning of the element.
INT32(544)	Boolean, TRUE if the user has been authenticated as an OLAP administrator for the database.
INT32(546)	A value that specifies whether security is dynamic.
INT32(483)	A value that specifies the number of roles defined in the database.
ROLE_NAME_ARRAY	An array that contains the names of the roles defined in the database.
CLOSE	A common data structure that indicates the end of the element.

2.2.5.2.7.1 ROLE_NAME_ARRAY

The syntax is defined as follows:

```
ROLE_NAME_ARRAY=
  STRING(484);
```

Element	Description
STRING(484)	The name of the role.

2.2.5.2.8 CUBE_ARRAY

The syntax is defined as follows:

```
CUBE_ARRAY=
  OPEN(99)
  INT32(100);
  *CUBE;
  CLOSE
```

Element	Description
OPEN(99)	A common data structure that indicates the beginning of the element.
INT32(100)	A value that specifies the number of cubes defined in the database.
CUBE	An array of data structures that contain information about the cubes defined in the database.

Element	Description
CLOSE	A common data structure that indicates the end of the element.

2.2.6 Get Cube

The client can request information about a specific cube. This section describes the structure of the client request and the server response.

2.2.6.1 Get Cube Request

The client issues a Get Cube Request to retrieve information about the specified cube from the server. The Get Cube request is a request for information about an individual OLAP cube in the current OLAP database on the server.

2.2.6.1.1 Request Code

```
REQCODE   = 'G'
```

2.2.6.1.2 PARAM_STRING

The following is the PARAM_STRING specific for this request.

```
PARAM_STRING =
  ""REQUEST=G;STATE="" FLAGS "";"" CUBEREF
```

2.2.6.2 Get Cube Response

The **Get Cube** response describes an individual OLAP cube available in the current OLAP database on the server. It contains a series of elements, each of which describes particular features of the specified cube.

```
RESPONSEDATA =
OPEN(94)
ARRAY_CUBE
INT32(234)
[CUBE_DIMENSION_ARRAY_DETAILS]
[CUBE_SQL_ARRAY]
 CLOSE
```

Element	Description
OPEN(94)	A common data structure that indicates the beginning of the element.
ARRAY_CUBE	An array of data structures that describe the cube.
INT32(234)	A bitmask value that specifies the state of the cube. If REQSPEC contains ""Type=B"" and ""Type=m"", this value is

Element	Description
	0x60000001. Otherwise, this value is 0.
[CUBE_DIMENSION_ARRAY_DETAILS]	An array of data structures that describe dimensions defined in the cube. This is returned only if INT32(234) = 0x60000001.
[CUBE_SQL_ARRAY]	An array of data structures that describe data sources defined in the cube. This data structure is returned if INT32REQFLAGS satisfies the following condition: Bit 0x00000001 is set Bit 0x00008000 is not set
CLOSE	A common data structure that indicates the end of the element.

2.2.6.2.1 ARRAY_CUBE

The syntax of **ARRAY_CUBE** is defined as follows:

```
ARRAY_CUBE=
  OPEN(85)
  OBJECT
  LOCKOBJECT
  INT32(235)
  INT32(490)
  INT32(530)
  INT64(237)
  INT32(577)
```

THE_REST_OF_ARRAY_CUBE ; This data structure is returned if INT32REQFLAGS satisfies the following condition:

Bit 0x00000001 is set

```
THE_REST_OF_ARRAY_CUBE =
INT32(86)
INT32(87)
INT32(88)
INT32(89)
INT32(90)
INT32(390)
INT32(395)
INT32(396)
INT32(91)
INT32(92)
INT32(93)
STRING(547)
STRING(548)
[MEASURE_GROUP_ARRAY]
[AUX_ARRAY]
INT32(386)
```

CLOSE

Element	Description
OPEN(85)	A common data structure that indicates the beginning of the element.
OBJECT	A common data structure containing metadata that describes the cube.
LOCKOBJECT	A common data structure for use by the Lock Manager.
INT32(235)	A value that specifies the state of the cube.
INT32(490)	A value that specifies whether the cube is write-enabled.
INT32(530)	A value that specifies the safety options enabled on the cube.
INT64(237)	A value that specifies the size of the cube on disk.
INT32(577)	A value that specifies the estimate of the size of the cube.
[INT32(86)]	A value that specifies the count of the number of dimensions in the cube.
[INT32(87)]	A value that specifies the count of the number of levels in the cube.
[INT32(88)]	A value that specifies the count of the number of measure groups in the cube.
[INT32(89)]	A value that specifies the count of the number of measures in the cube.
[INT32(90)]	A value that specifies the count of the number of auxiliary elements in the cube.
[INT32(390)]	A value that specifies whether unique name generation is enabled on the cube.
[INT32(395)]	A value that specifies MDX compatibility for the cube.
[INT32(396)]	A value that specifies SQL compatibility for the cube.
[INT32(91)]	A value that specifies an internal version of the cube.
[INT32(92)]	A value that specifies an internal dimension version.
[INT32(93)]	A value that specifies an internal cache version.
[STRING(547)]	A value that specifies the link source for the cube.
[STRING(548)]	A value that specifies the link cube for the cube.
[MEASURE_GROUP_ARRAY]	An array of data structures that contain information about the measure groups in the cube.
[AUX_ARRAY]	An array of data structures that contain information about the auxiliary elements in the cube.
[INT32(386)]	A value that specifies the count of partitions in the cube.
CLOSE	A common data structure that indicates the end of the element.

2.2.6.2.1.1 MEASURE_GROUP_ARRAY

The syntax of **MEASURE_GROUP_ARRAY** is defined as follows:

```
MEASURE_GROUP_ARRAY=
  OPEN(73)
  STRING(74)
  STRING(75)
  INT32(76)
  *ARRAY_MEASURE_GROUP
  CLOSE
```

Element	Description
OPEN(73)	A common data structure that indicates the beginning of the element.
STRING(74)	A value that specifies the name of the dimension to which the measure group belongs.
STRING(75)	A value that specifies the name of the level to which the measure group belongs.
INT32(76)	A value that specifies the count of the number of measure groups.
*ARRAY_MEASURE_GROUP	An array of data structures that contain information about measure groups.
CLOSE	A common data structure that indicates the end of the element.

2.2.6.2.1.2 ARRAY_MEASURE_GROUP

The syntax of **ARRAY_MEASURE_GROUP** is defined as follows:

```
ARRAY_MEASURE_GROUP=
  OPEN(72)
  OBJECT
  [MEASURE_GROUP]
  [MEASURE_ARRAY]
  CLOSE
```

Element	Description
OPEN(72)	A common data structure that indicates the beginning of the element.
OBJECT	A common data structure containing metadata that describes the measure group.
[MEASURE_GROUP]	A of data structure that contains information about a measure group. This data structure is returned if INT32REQFLAGS satisfies the following condition: Bit 0x00000001 is set
[MEASURE_ARRAY]	An array of data structures that contain information about the measures in a measure group. This data structure is returned if INT32REQFLAGS satisfies the following condition: Bit 0x00000001 is set

Element	Description
CLOSE	A common data structure that indicates the end of the element.

2.2.6.2.1.3 MEASURE_GROUP

The syntax of **MEASURE_GROUP** is defined as follows:

```
MEASURE_GROUP=
  OPEN(46)
  INT32(47)
  INT16(48)
  INT32(49)
  INT32(50)
  INT32(51)
  INT32(589)
  MEASURE_GROUP_VIRTUAL
  CLOSE
```

Element	Description
OPEN(46)	A common data structure that indicates the beginning of the element.
INT32(47)	A value from the DETTYPE enumeration that indicates whether the cube is regular or virtual.
INT16(48)	A value that specifies the number of measures per measure group.
INT32(49)	A value that specifies the version of the cube.
INT32(50)	A value that specifies the version of the dimension.
INT32(51)	A value that specifies the version of the cache.
INT32(589)	A value that specifies the count of partitions in the cube.
MEASURE_GROUP_VIRTUAL	A data structure that contains information about a virtual measure group. Only present if INT32(47) = 1.
CLOSE	A common data structure that indicates the end of the element.

The INT32(47) field uses a value from the following enumeration:

REGULAR CUBE	**1**
VIRTUAL CUBE	3

2.2.6.2.1.4 MEASURE_GROUP_VIRTUAL

The syntax of **MEASURE_GROUP_VIRTUAL** is defined as follows:

```
MEASURE_GROUP_VIRTUAL =
  INT32(52)
  INT32(53)
```

```
VIRTUAL_MEASURE_GROUP_DIMENSION_ARRAY
ARRAY_INT16(54)
```

Element	Description
INT32(52)	A value that specifies the cube id for the virtual measure group.
INT32(53)	A value that specifies the count of dimensions in the virtual measure group.
VIRTUAL_MEASURE_GROUP_DIMENSION_ARRAY	An array of data structures that contain information about the dimensions in the virtual measure group.
ARRAY_INT16(54)	An array of values that specify the indices of the dimensions in the cube.

2.2.6.2.1.5 VIRTUAL_MEASURE_GROUP_DIMENSION_ARRAY

The syntax of **VIRTUAL_MEASURE_GROUP_DIMENSION_ARRAY** is defined as follows:

```
VIRTUAL_MEASURE_GROUP_DIMENSION_ARRAY=
  ARRAY_VIRTUAL_MEASURE_GROUP_DIMENSION
  VIRTUAL_MEASURE_GROUP_LEVEL_ARRAY
```

Element	Description
ARRAY_VIRTUAL_MEASURE_GROUP_DIMENSION	A data structure that contains information about a dimension in a virtual measure group.
VIRTUAL_MEASURE_GROUP_LEVEL_ARRAY	An array of data structures that contain information about the levels in a virtual measure group.

2.2.6.2.1.6 VIRTUAL_MEASURE_GROUP_LEVEL_ARRAY

The syntax of **VIRTUAL_MEASURE_GROUP_LEVEL_ARRAY** is defined as follows:

```
VIRTUAL_MEASURE_GROUP_LEVEL_ARRAY=
  ARRAY_LEVEL
```

Element	Description
ARRAY_LEVEL	A data structure that contains information about a level.

2.2.6.2.1.7 ARRAY_VIRTUAL_MEASURE_GROUP_DIMENSION

The syntax of **ARRAY_VIRTUAL_MEASURE_GROUP_DIMENSION** is defined as follows:

```
ARRAY_VIRTUAL_MEASURE_GROUP_DIMENSION=
  OBJECT
  LOCKOBJECT
```

```
DIMENSION
```

Element	Description
OBJECT	A common data structure containing metadata that describes the dimension.
LOCKOBJECT	A common data structure for use by the Lock Manager.
DIMENSION	A data structure that contains information about the dimension.

2.2.6.2.1.8 MEASURE_ARRAY

The syntax of **MEASURE_ARRAY** is defined as follows:

```
MEASURE_ARRAY=
  *ARRAY_MEASURE
```

Element	Description
ARRAY_MEASURE	An array of data structures that contain information about measures.

2.2.6.2.1.9 ARRAY_MEASURE

The syntax of **ARRAY_MEASURE** is defined as follows:

```
ARRAY_MEASURE=
  OPEN(42)
  ARRAY_MEASURE_BASE
  ARRAY_MEASURE_DIMENSION
  ARRAY_MEASURE_LEVEL
  [MEASURE]
  CLOSE
```

Element	Description
OPEN(42)	A common data structure that indicates the beginning of the element.
ARRAY_MEASURE_BASE	A data structure that contains standard object information about a measure.
ARRAY_MEASURE_DIMENSION	A data structure that contains information about the dimension associated with a measure.
ARRAY_MEASURE_LEVEL	A data structure that contains information about the level associated with a measure.
[MEASURE]	A data structure that contains information about the measure. This data structure is returned if INT32REQFLAGS satisfies the following condition: Bit 0x00000001 is set
CLOSE	A common data structure that indicates the end of the element.

2.2.6.2.1.10 ARRAY_MEASURE_BASE

The syntax of **ARRAY_MEASURE_BASE** is defined as follows:

```
ARRAY_MEASURE_BASE=
OBJECT
```

Element	Description
OBJECT	A common data structure containing metadata that describes the measure.

2.2.6.2.1.11 ARRAY_MEASURE_DIMENSION

The syntax of **ARRAY_MEASURE_DIMENSION** is defined as follows:

```
ARRAY_MEASURE_DIMENSION=
OBJECT
```

Element	Description
OBJECT	A common data structure containing metadata that describes the object.

2.2.6.2.1.12 ARRAY_MEASURE_LEVEL

The syntax of **ARRAY_MEASURE_LEVEL** is defined as follows:

```
ARRAY_MEASURE_LEVEL=
OBJECT
```

Element	Description
OBJECT	A common data structure containing metadata that describes the level.

2.2.6.2.1.13 MEASURE

The syntax of **MEASURE** is defined as follows:

```
MEASURE=
OPEN(41)
INT16 (28)
INT32 (29)
INT32 (30)
INT32 (31)
INT32 (32)
STRING (33)
STRING (459)
INT32 (357)
CLOSE
```

Element	Description
OPEN(41)	A common data structure that indicates the beginning of the element.
INT16 (28)	An index of the measure within the measure group.
INT32 (29)	A value from an enumeration that indicates whether the CUBE is regular or virtual.
INT32 (30)	An enumeration value that specifies the data type for the measure.
INT32 (31)	An enumeration that specifies the aggregation type for the measure.
INT32 (32)	A value that specifies the size of the measure in bytes.
STRING (33)	A value that specifies the format of the measure; for example, "Currency".
STRING (459)	A value that specifies the source of the measure.
INT32 (357)	Boolean. If dimension security is specified for the measure, the value is 1; otherwise, the value is 0.
CLOSE	A common data structure that indicates the end of the element.

The INT32(29) field uses a value from the DETTYPE enumeration:

Element	Value
REGULAR CUBE	1
COMMENT	2
VIRTUAL CUBE	3

The Measure Type enumeration for INT32(30)

Element	Value
NULL	0x00
BYTE	0x01
INT	0x02
LONG	0x03
FLOAT	0x04
REAL	0x05
DATE	0x06
CURRENCY	0x07
SHORT	0x09

The Aggregation Type enumeration for INT32(31)

Element	Value
SUM or COUNT	1
MAX	2
MIN	3
DISTINCT COUNT	4

2.2.6.2.2 DIMENSION

The syntax of **DIMENSION** is defined as follows:

```
DIMENSION=
OPEN(36)
INT32(8)
INT32(267)
INT32(409)
STRING(410)
INT16(9)
STRING(10)
STRING(449)
INT32(331)
CLOSE
```

Element	Description
OPEN(36)	A common data structure that indicates the beginning of the element.
INT32(8)	A value from the DIMTYPE enumeration that specifies the type of the dimension.
INT32(267)	A value from the DIMSTRUCTTYPE enumeration that specifies the structural type of the dimension.
INT32(409)	A value from the DIMDATAFOR enumeration that specifies what the dimension data is for.
STRING(410)	A value that specifies data for captions.
INT16(9)	A value that specifies the count of levels in the dimension.
STRING(10)	A value that specifies the cube name for dimension data.
STRING(449)	A value that specifies the original cube for a private dimension.
INT32(331)	A value that specifies the version of the dimension.
CLOSE	A common data structure that indicates the end of the element.

The INT32(8) element uses a value from the DIMTYPE enumeration:

Value	Description
0x00	UNKNOWN
0x01	TIME

Value	Description
0x02	MEASURE
0x03	OTHER
0x05	QUANTITATIVE
0x06	ACCOUNTS
0x07	CUSTOMERS
0x08	PRODUCTS
0x09	SCENARIO
0x0A	UTILIY
0x0B	CURRENCY
0x0C	RATES
0x0D	CHANNEL
0x0E	PROMOTION
0x0F	ORGANIZATION
0x10	BILL_OF_MATER
0x11	GEOGRAPHY

The INT32(267) element uses a value from the DIMSTRUCTTYPE enumeration, which gives the structural type of the dimension.

Value	Description
1	**rigid dimension**
2	**flexible dimension**
3	virtual dimension

The INT32(409) element uses a value from the DIMDATAFOR enumeration which applies to parent-child dimensions.

Value	Description
1	Only LEAF members may have data.
2	**Nonleaf** members may have data; the data is aggregated among the member's descendants.
3	Nonleaf members may have data; the data is NOT aggregated among the member's descendants.

2.2.6.2.3 LEVEL

The syntax of **LEVEL** is defined as follows:

```
LEVEL=
OPEN(44)
INT16(24)
INT32(25)
INT16(26)
INT32(27)
STRING(301)
STRING(302)
STRING(303)
STRING(347)
INT16(355)
STRING(356)
INT32(402)
INT32(417)
CLOSE
```

Element	Description
OPEN(44)	A common data structure that indicates the beginning of the element.
INT16(24)	A value that specifies the level number.
INT32(25)	A value from the LEVELTYPE enumeration that specifies the type of the level.
INT16(26)	A value that specifies the maximum index of the level.
INT32(27)	A value that specifies the number of members for the level.
STRING(301)	A value that specifies the **master dimension name** for the level.
STRING(302)	A value that specifies the **master level name** for the level.
STRING(303)	A value that specifies the **master property name** for the level.
STRING(347)	A value that specifies the name of the member property to sort the level.
INT16(355)	Boolean to indicate if a level has custom rollup.
STRING(356)	A value that specifies the formula text for the level. Only present if INT16(255) = TRUE.
INT32(402)	An enumeration that specifies the key type for the level.
INT32(417)	The count of Ntab structures for this level that are in memory.
CLOSE	A common data structure that indicates the end of the element.

The INT32(25) field uses the values from the LEVELTYPE enumeration:

REGULAR	**0x0000**
ALL	0x0001
CALCULATED	0x0002
TIME	0x0004
RESERVED1	0x0008

REGULAR	**0x0000**
TIME_YEARS	0x0014
TIME_HALF_YEAR	0x0024
TIME_QUARTERS	0x0044
TIME_MONTHS	0x0084
TIME_WEEKS	0x0104
TIME_DAYS	0x0204
TIME_HOURS	0x0304
TIME_MINUTES	0x0404
TIME_SECONDS	0x0804
TIME_UNDEFINED	0x1004
GEO_CONTINENT	0x2001
GEO_REGION	0x2002
GEO_COUNTRY	0x2003
GEO_STATE_OR_PROVINCE	0x2004
GEO_COUNTY	0x2005
GEO_CITY	0x2006
GEO_POSTALCODE	0x2007
GEO_POINT	0x2008
ORG_UNIT	0x1011
BOM_RESOURCE	0x1012
QUANTITATIVE	0x1013
ACCOUNT	0x1014
CUSTOMER	0x1021
CUSTOMER_GROUP	0x1022
CUSTOMER_HOUSEHOLD	0x1023
PRODUCT	0x1031
PRODUCT_GROUP	0x1032
SCENARIO	0x1015
UTILITY	0x1016
PERSON	0x1041

REGULAR	0x0000
COMPANY	0x1042
CURRENCY_SOURCE	0x1051
CURRENCY_DESTINATION	0x1052
CHANNEL	0x1061
REPRESENTATIVE	0x1062
PROMOTION	0x1071

The Level Key Type enumeration for INT32(402)

Description	Value
NULL	0x00
BYTE	0x01
INT	0x02
LONG	0x03
FLOAT	0x04
REAL	0x05
DATE	0x06
CURRENCY	0x07
SHORT	0x09

2.2.6.2.3.1 AUX

The syntax of AUX is defined as follows:

```
AUX=
OPEN(65)
INT32(66)
STRING(67)
INT32(190)
INT32(582)
CLOSE
```

Element	Description
OPEN(65)	A common data structure that indicates the beginning of the element.
INT32(66)	A value from the AUXTYPE enumeration that specifies the type of the auxiliary element.
STRING(67)	A value that specifies the text of the auxiliary element;

Element	Description
INT32(190)	A value that specifies the key length of the auxiliary element
INT32(582)	Globally unique cookie. Client passes it if sending the query context to the server for calculation.
CLOSE	A common data structure that indicates the end of the element.

The INT32(66) field uses a value from the AUXTYPE enumeration:

PN_AUX_FORMULA	**1**
PN_AUX_LOOKUP	2
PN_AUX_PROPERTIES	3

2.2.6.2.3.2 ARRAY_CUBE_SECURITY

The syntax of **ARRAY_CUBE_SECURITY** is defined as follows:

```
ARRAY_CUBE_SECURITY=
OBJECT
CUBE_SECURITY
```

Element	Description
OBJECT	A common data structure containing metadata that describes the cube security object.
CUBE_SECURITY	A data structure that contains information about cube security.

2.2.6.2.3.3 CUBE_SECURITY

The syntax of **CUBE_SECURITY** is defined as follows.

```
CUBE_SECURITY=
  OPEN(259)
  INT32(498)
  INT32(285)
  INT32(353)
  INT32(499)
  INT32(500)
  INT32(359)
  ROLECOMMAND_ARRAY
  INT32(261)
  CELL_SECURITY_ARRAY
  CUBE_DIMENSION_SECURITY
  CLOSE
```

Element	Description
OPEN(259)	A common data structure that indicates the beginning of the element.

Element	Description
INT32(498)	A value that specifies whether cube security is enabled.
INT32(260)	A value that specifies the level of access to the cube: 0 = None 1 = Read 3 = Read/Write
INT32(285)	A value that specifies whether security is enforced on the server or on the client.
INT32(353)	A value that specifies whether drillthrough is allowed.
INT32(499)	A value that specifies whether linking to this cube is allowed.
INT32(500)	A value that specifies whether SQL queries are allowed to this cube.
INT32(359)	A value that specifies the count of role commands for the cube.
ROLECOMMAND_ARRAY	An array of data structures that contain information about role commands associated with cube security.
INT32(261)	A value that specifies the count of cell security statements in the cube.
CELL_SECURITY_ARRAY	An array of data structures that contain information about cell security statements in the cube.
CUBE_DIMENSION_SECURITY	A data structure that contains information about dimension security in the cube.
CLOSE	A common data structure that indicates the end of the element.

2.2.6.2.3.3.1 CELL_SECURITY_ARRAY

The syntax of **CELL_SECURITY_ARRAY** is defined as follows:

```
CELL_SECURITY_ARRAY=
  *STRING(262)
```

Element	Description
*STRING(262)	An array of cell security statements.

2.2.6.2.3.4 CUBE_DIMENSION

The syntax of **CUBE_DIMENSION** is defined as follows.

```
CUBE_DIMENSION=
OPEN(97)
ARRAY_DIMENSION
CLOSE
```

Element	Description
OPEN(97)	A common data structure that indicates the beginning of the element.
ARRAY_DIMENSION	A data structure that contains information about a dimension in the cube.
CLOSE	A common data structure that indicates the end of the element.

2.2.6.2.3.4.1 ARRAY_DIMENSION

2.2.6.2.3.5 ARRAY_DIMENSION

The syntax is defined as follows:

```
ARRAY_DIMENSION=
OPEN(68)
OBJECT
LOCKOBJECT
DIMENSION
INT32(69)
*LEVEL_ARRAY
[CUBE_TREE]
[CustomMemberFormulas]
CLOSE
```

Element	Description
OPEN(68)	A common data structure that indicates the beginning of the element.
OBJECT	A common data structure containing metadata that describes the object.
LOCKOBJECT	A common data structure for use by the Lock Manager.
DIMENSION	A data structure that contains information about a dimension in the cube.
INT32(69)	A value that specifies the number of levels defined in the dimension.
LEVEL_ARRAY	An array of data structures that contain information about the levels in the dimension. This data structure is returned if INT32REQFLAGS satisfies the following condition: Bit 0x00000001 is set
[CUBE_TREE]	Structure that contains information about dimensions defined in the cube. This data structure is returned if INT32REQFLAGS satisfies the following condition: Bit 0x00000001 is set Bit 0x40000000 is set
[CustomMemberFormulas]	Structure that contains information about custom member formulas defined in the dimension. This data structure is returned if INT32REQFLAGS satisfies the following condition: Bit 0x00000001 is set

Element	Description
CLOSE	A common data structure that indicates the end of the element.

2.2.6.2.3.5.1 ARRAY_SQL

The syntax of **ARRAY_SQL** is defined as follows:

```
ARRAY_SQL=
OPEN(80)
OBJECT
SQL_DATA_SOURCE
CLOSE
```

Element	Description
OPEN(80)	A common data structure that indicates the beginning of the element.
OBJECT	A common data structure containing metadata that describes the object.
SQL_DATA_SOURCE	A data structure that contains information about a data source.
CLOSE	A common data structure that indicates the end of the element.

2.2.6.2.3.5.2 SQL_DATA_SOURCE

The syntax of **SQL_DATA_SOURCE** is defined as follows:

```
SQL_DATA_SOURCE=
OPEN(55)
STRING(56)
STRING(57)
STRING(58)
STRING(59)
INT32(60)
INT32(61)
REAL64(62)
CLOSE
```

Element	Description
OPEN(55)	A common data structure that indicates the beginning of the element.
STRING(56)	A value that specifies the connection string for the data source.
STRING(57)	A value that specifies the data source name (DSN) that is used to connect to the data source.
STRING(58)	A value that specifies the user ID used to connect to the data source.
STRING(59)	A value that specifies the password used to connect to the data source.
INT32(60)	A value that specifies the state of the data source.

Element	Description
INT32(61)	A value that specifies the timeout value for connecting to the data source.
REAL64(62)	A value that specifies the date stamp for the data source. See section 2.2.1.5.1.1 for the storage format for dates.
CLOSE	A common data structure that indicates the end of the element.

2.2.7 Get Dimension Members

The client can request information about the members of a specific dimension. This section describes the structure of the client request and the server response.

2.2.7.1 Get Dimension Members Request

The client issues a Get Dimension Members Request to retrieve information about dimension members and their siblings for a given dimension.

2.2.7.1.1 Request Code

```
REQCODE   = 'X'
```

2.2.7.1.2 PARAM_STRING

The following is the PARAM_STRING that is specific for this request.

```
PARAM_STRING =
""REQUEST=X;STATE="" FLAGS "";"" CUBEREF
               "";DIM="" DIMNUMBER "";LEVEL="" LEVELNUMBER "";SLEVEL="" SLEVELNUMBER
DIMNUMBER = STRING
```

DIMNUMBER is a string representation of an integer that specifies a dimension number inside the cube from which the members are to be retrieved.

LEVELNUMBER = STRING

LEVELNUMBER is a string representation of an integer that specifies the level number inside the dimension from which the members are to be retrieved.

SLEVELNUMBER = STRING

SLEVELNUMBER is a string representation of an integer that specifies the starting level number inside the dimension from which the members are to be retrieved.

The OTHER_PARAM parameters that are not encoded as Unicode string for this request are as follows:

```
OTHER_PARAM = [DPATH]
DPATH = *DATAID
```

```
DATAID = INT16
```

DPATH is a sequence of DataIDs that specify the parent member, which children are to be retrieved. If this parameter is not specified, all the members from the levels that are specified above are to be retrieved.

In the OTHER _PARAM for this request, the string ""OTHER_PARAM="" is Unicode-encoded. That string is immediately followed by an array of INT16 bytes as described.

2.2.7.2 Get Dimension Members Response

```
RESPONSEDATA =
STATUS
Cube_Tree
```

Value	Meaning
STATUS	The RESPONSESTATUS component MUST be set to 1.
Cube_Tree	Specifies the data structure of a dimension tree.
STATUS	Valid STATUS report for this object.

2.2.8 Get Member Properties

The client can request information about the properties of a specific member. This section describes the structure of the client request and the server response.

2.2.8.1 Get Member Properties Request

The client issues a Get Member Properties request to retrieve dimension members and their siblings for the specified dimension.

2.2.8.1.1 Request Code

REQCODE = 'T'

2.2.8.1.2 PARAM_STRING

The following is the PARAM_STRING specific for this request.

```
PARAM_STRING =
""REQUEST=T;STATE="" FLAGS "";"" CUBEREF
            "";DIM="" DIMNUMBER "";LEVEL="" LEVELNUMBER "";START="" STARTNUMBER
"";FINISH="" FINISHNUMBER
DIMNUMBER = STRING
```

DIMNUMBER is a string representation of an integer which specifies the dimension number inside the cube from which member properties are to be retrieved.

LEVELNUMBER = STRING

LEVELNUMBER is a string representation of an integer which specifies the level number inside the dimension from which member properties are to be retrieved.

START = STRING

START is a string representation of an integer which specifies the starting level number inside the dimension from which member properties are to be retrieved.

FINISH = STRING

FINISH is a string representation of an integer which specifies the last level number inside the dimension from which member properties are to be retrieved.

2.2.8.2 Get Member Properties Response

```
RESPONSEDATA =
   OPEN(63)
   INT32(198)
   *DimensionMemberPropertyList
   CLOSE
```

Value	Meaning
Open(63)	A common data structure that indicates the beginning of the element.
INT32(198)	A value that specifies the number of records.
DimensionMemberPropertyList	A data structure that repeats once for each dimension member.
Close	A common data structure that indicates the end of the element.

2.2.8.2.1 DimensionMemberPropertyList

```
DimensionMemberPropertyList=
*DimensionMemberProperty    member
```

Value	Meaning
DimensionMemberProperty	Specifies the data structure of a property for a dimension member. This data structure repeats one time for each property.

2.2.8.2.1.1 DimensionMemberProperty

```
DimensionMemberProperty =
STRING(64)
INT32(64)
INT64(64)
REAL32(64)
REAL64(64)
```

Value	Meaning
Variant(64)	The value of the member property, expressed as the appropriate data type.

2.2.9 Get RecordSet

The client can request a recordset. This section describes the structure of the client request and the server response.

2.2.9.1 Get RecordSet Request

The client issues a Get RecordSet request to retrieve cell data from the cube for the specified subcube. See section 4.5.1 for an example.

2.2.9.1.1 Request Code

```
REQCODE   = '@'
```

2.2.9.1.2 PARAM_STRING

The following is the PARAM_STRING specific for this request.

```
PARAM_STRING =
  ""REQUEST=@;STATE="" FLAGS "";"" CUBEREF
```

2.2.9.1.3 OTHER_PARAM

The OTHER_PARAM parameters, which are not encoded as Unicode string for this request, are as follows:

OTHER_PARAM = DATASET SLICE

DATASET = *DATASETLEVEL

DATASETLEVEL = CHAR ; This is ASCII character

DataSet is encoded as a sequence of DATASETLEVEL (which are ASCII characters), one per dimension in the cube. The integer value of the ASCII character DATASETLEVEL minus constant 48 is the number of the level in the corresponding dimension, representing the granularity of the data to be returned by this request. For more information, see section 7.

```
SLICE = *DATAID
DATAID = INT16
```

Slice is encoded as sequence of DATAID (which are 2 byte integers). For more information, see section 7.

The OTHER_PARAM for this request will look as follows, with the parts in quotation marks being Unicode characters:

""OTHER_PARAM=DATASET=""[ASCII characters]""SLICE=""[INT16 BYTES]

and where the ASCII characters and INT16 BYTES will be as described above.

2.2.9.2 Get RecordSet Response

See section 4.5.2 for an example.

```
RESPONSEDATA =
OPEN (127)
INT32 (128)
INT32 (129)
[RecordsDescription]
CLOSE
*RecordPage
```

Element	Description
OPEN(127)	Indicates the start of the response.
INT32(128)	Always 0.
INT32(129)	A value that indicates the number of records.
RecordsDescription	An optional property that is present only when the record count is greater than 0.
CLOSE	Indicates the end of the response.
RecordPage	Contains one or more Records. A RecordSet contains zero or more RecordPage elements.

2.2.9.2.1 RecordsDescription

Describes the size and layout of records that are common to both client and server memory.

The syntax is defined as follows:

```
RecordsDescription =
INT32(130)
INT32(131)
INT16(132)
FilterOnClient
```

Element	Description
INT32 (130)	A value that indicates the number of pages.
INT32 (131)	A value that indicates the number records per page.
INT16 (132)	A value that indicates the size of a record, in bytes.
FilterOnClient	Boolean value directing client as to whether it must remove secured records or if server has already done it.

2.2.9.2.1.1 FilterOnClient

The syntax is defined as follows.

```
FilterOnClient =
INT32(320)
```

Element	Description
INT32 (320)	A value that specifies dimension security. A value of 1 indicates that the client MUST apply dimension security to remove secured records. A value of 0 indicates that the server already applied dimension security.

2.2.9.2.2 RecordPage

The syntax is defined as follows.

```
RecordPage :=
*Record
```

See section 2.2.1.4.5 for more information.

Element	Description
Record	Actual field values of a record. See section 7 for more information.

2.2.10 Calculate MDX fragment

The client can request a calculation for a specific MDX expression. This section describes the structure of the client request and the server response for Calculate MDX fragment Request.

2.2.10.1 Calculate MDX Fragment Request

The client issues a Calculate MDX Fragment request to get a calculation of a specific MDX expression in the context of a user MDX session and MDX query. See section 4.3.1 for an example.

2.2.10.1.1 Request Code

```
REQCODE   = 'Q'
```

2.2.10.1.2 PARAM_STRING

The following is the PARAM_STRING specific for this request.

```
PARAM_STRING =
""REQUEST=Q;STATE="" FLAGS "";"" CUBEREF
```

2.2.10.1.3 REQDATA

The REQDATA part of the request is as follows.

```
REQDATA =
OPEN(166)
Int32(389)
Int32(397)
Int32(398)
Int32(399)
*DirtyDimensionLevels
Int32(167)
QueryContextQueryDefinition
[QueryContextQueryDefinition]
CLOSE
```

Element	Definition
OPEN(166)	A common data structure that indicates the beginning of the element.
Int32(389)	Secured Cell Value Property Value. See section 2.2.10.1.3.1 for more information.
Int32(397)	MDX Unique Name Style property. See section 2.2.10.1.3.2 and [MSDN-MDXUniqueNameStyle] for more information.
Int32(398)	MDX Compatibility property value. See section 2.2.10.1.3.3 and [MSDN-MDXCompatibility] for more information.
Int32(399)	MUST be the same value as the user sets for the SQL Compatibility property. See [MSDN-SQLCompatibility] for more information.
*DirtyDimensionLevels	Repeat for each dimension.
Int32(167)	Indicates Session Query status. Always set. 0x00000001 Is set if session query. 0x00000002 Is set if performance monitor count has been incremented.0x00000004
[QueryContextQueryDefinition]	MUST be sent only when INT32(167) x00000002 bit is not set.
CLOSE	A common data structure that indicates the end of the element.

2.2.10.1.3.1 INT32(389) – Secured Cell Value

See [MSDN-SecuredCellValue] for more information.

The table below specifies the values in the cells secured by cell security for the different values of Secured Cell Value property

Value	Definition
0	Default. For compatibility with earlier versions, this means the same as Value 1. The meaning of this default value is subject to change in future versions.

Value	Definition
1	Returns: HRESULT = NO_ERROR The Value property of the cell contains the result as a variant data type. The string "#N/A" is returned in the Formatted Value property.
2	An error is returned as the value of HRESULT.
3	NULL is returned in both the Value and Formatted Value properties.
4	A numerical zero (0) is returned in the Value property, and a formatted zero is returned in the Formatted Value property. For example, 0.00 is returned in the Formatted Value property for a cell whose format property is "#.##".
5	The string "#SEC" is returned in both the Value and Formatted Value properties.

2.2.10.1.3.2 INT32(397) Unique Name Style Property

See [MSDN-MDXUniqueNameStyle] for more information.

Use this property to specify the algorithm for generating unique names according to the values below.

Value	Description
0	Default. For compatibility with earlier versions, this is the same as Value 2. The meaning of this default value is subject to change in future versions.
1	Key path algorithm: [dim].&[k1].&[k2]
2	Compatible with version 7.0, name path algorithm: [dim].[n1].[n2]
3	Compatible with SQL Server 2000 Analysis Services. The algorithm uses guaranteed unique names, which are stable over time.

2.2.10.1.3.3 INT32(398) MDX Compatibility Property

See [MSDN-MDXCompatibility] for more information.

This property determines how placeholder members in a ragged hierarchy or unbalanced hierarchy are treated. The following table describes possible values for this property.

Value	Description
0	Default. For compatibility with earlier versions, this is the same as Value 1. The meaning of this default value is subject to change in future versions.
1	Default value. Placeholder members are exposed.
2	Placeholder members are not exposed.

2.2.10.1.3.4 DirtyDimensionLevels

The syntax is defined as follows:

```
DirtyDimensionLevels =
  Int16(403)
```

Element	Description
Int16(403)	Dirty Level number

2.2.10.1.3.5 QueryContextQueryDefinition

The syntax is defined as follows:

```
QueryContextQueryDefinition=
  OPEN(133)
  Int32(221)
Int32(223)
Int32(567)
Int32(231)
Int32(501)
Real64(502)
Int32(542)
Int32(571)
Int32(572)
Real64(586)
Int32(587)
Int16(134)
Int16(135)
Int16(227)
Int16(248)
CellFormulas
*AxisInformation
*CalculatedMembers
SkipComingObject
[SessionQueryContext]
Int8(552)
[MPATH(551)]
[SessionQueryContextLevelCustomRollups]
[StatefulVisualTotals]
CLOSE
```

Element	Description
OPEN(133)	A common data structure that indicates the beginning of the element.
Int32(221)	This is a bitmask that represents the internal state of the query context, as to how calculation is shared between the client and the server. SESSION QUERYCONTEXT: 0x00000001 QUERYCONTEXT EMULATED MODE: 0x00000002 QUERYCONTEXT ISOLATED MODE: 0x00000004 QUERYCONTEXT ZOMBIE: 0x00000008 QUERYCONTEXT OWNS MPATH: 0x00000010 PN_QC_READONLY: 0x00000020

Element	Description
	Query context is in read-only mode (no commit). Cancel was requested by user: 0x00000100 There are **Stateful Visual Totals** on the query context: 0x00000200 Any non-server items should be treated as not existent, such as ignore client calculation definitions: 0x00000400 Secure middle-tier setting: 0x00000800 Do not use formula cache: 0x00001000 Debug mode for the Query Context: 0x00002000 Query context is initialized: 0x80000000
Int32(223)	This value indicates the Execution Location. For more information, see [MSDN-ExecutionLocation]. Default: 0x00000000 Optimal performance (application chosen): 0x00000001 Client: 0x00000002 Server: 0x00000003
Int32(567)	This value indicates the Default MDX visual mode. For more information, see [MSDN-DefaultMDXVisualMode].
Int32(231)	Mask indicates which optional properties are to be returned in the query. FORMAT 0x00000001 BACKCOLOR 0x00000002 FORECOLOR 0x00000004 FONT_NAME 0x00000008 FONT_SIZE 0x00000010 FONT_ATTR 0x00000020 ALIGNMENT 0x00000040
Int32(501)	Bitmask that indicates which **SONAR** algorithm to use. More information about the Analysis Services SONAR algorithm can be found in [MSDN-ASPIA]. SONAR_POLICY_DEFAULT 0 SONAR_POLICY_WINDOWING 1 SONAR_POLICY_ALLSCAN 2 SONAR_POLICY_ISOLATED 3 SONAR_POLICY_IGNORE_VP 4 SONAR_POLICY_INFINITE_VP 5 SONAR_POLICY_LARGEST_AREA_FIRST 6 SONAR_POLICY_OR_QUERIES 7
Real64(502)	This value indicates a **cache ratio** (default: 0.5). For more information see [MSDN-CacheRatio].
Int32(542)	Bitmask indicates the Query Context CacheMode (Default: 0xFFFFFFFF). This is for communication between Storage

Element	Description
	Engine code component and Cache Registry code component. CACHE_MODE_NOTUSED 0xFFFFFFFF CACHE_MODE_DEFAULT 0 CACHE_MODE_REGONLY 1 CACHE_MODE_SCAN 2
Int32(571)	This bitmask provides additional information as to which variant of the SONAR algorithm to use. DEEP_DATA_SET 10 DEEP_REMOTE_DATA_SET 50 DEEP_REMOTE_DATA_SET_INFINITE 0
Int32(572)	This value indicates the depth to be used in the SonarDeepRemoteDataSet.
Real64(586)	This value indicates the SONAR Cache Ratio (default: 0.5).
Int32(587)	This value indicates the number of remaining overall seconds for query time out. Since query calculation is shared between client and server, they must coordinate on remaining time within user-set timeout parameter.
Int16(134)	This value indicates the number of sets. This could be **named sets**, client defined sets, or an axis.
Int16(135)	This value indicates number of calculated members.
Int16(227)	This value indicates number of user defined functions.
Int16(248)	This value indicates number of cell security restrictions.
*AxisInformation	Repeated for each axis definition.
*CalculatedMembers	Repeated for each calculated member (server side or client side).
SkipComingObject	This data (user-defined function) is ignored.
[SessionQueryContext]	Only when INT32(221) has set SESSION_QUERYCONTEXT bit.
Int8(552)	This value indicates the number of user-defined default members.
MPATH(551)	This value indicates the MPATH including user-defined default members.
[SessionQueryContextLevelCustomRollups]	Only when INT32(221) has set the SESSION_QUERYCONTEXT bit.
[StatefulVisualTotals]	Only when INT32(223) is FORCE_SERVER.
CLOSE	A common data structure that indicates the end of the element.

2.2.10.1.3.5.1 CellFormulas

The syntax is defined as follows:

```
CellFormulas =
OPEN(503)
  Int32(504)
*CellFormula
  CLOSE
```

Element	Description
OPEN(503)	A common data structure that indicates the beginning of the element.
Int32(504)	This value indicates the number of cell formulas.
*CellFormula	Repeat for each entry.
CLOSE	A common data structure that indicates the end of the element.

2.2.10.1.3.5.1.1 CellFormula

The syntax is defined as follows.

```
CellFormula=
String(505)
String(506)
Int32(507)
DATASET(508)
DATASET(509)
SLICE(510)
Array(529, length)
String(511)
String(512)
Int32(513)
Int32(514)
Int32(515)
Int32(516)
Int32(528)
[String(517)]
[String(518)]
[UInt16(519)]
[UInt16(520)]
[UInt32(521)]
[UInt32(522)]
String(523)
String(524)
String(525)
String(526)
String(527)
```

Element	Description
String(505)	This value indicates name of the CELL CALCULATION.

Element	Description
String(506)	This value provides a description.
Int32(507)	This value indicates whether the calculation utilizes an internal optimization code pathway.
DATASET(508)	Only if INT32(507)=TRUE. The Top space of the optimization calculation.
DATASET(509)	Only if INT32(507)=TRUE. The Bottom space of the optimization calculation.
SLICE(510)	This value indicates Slicers (number of levels in the cube + 1 (measure)).
Array(529, length)	An array of masks, one for each dimension. The length of this array in bytes will be the number of dimensions times the size of an INT32 on the hardware. Include calculated members. 0x00000001 Include specified member only, no descendants: 0x00000002 Verify range, leaves only. 0x00000004 Verify range, non-leaves only. 0x00000008
String(511)	This value is MDX that represents the CONDITON part of the expression.
String(512)	This value is MDX that represents the calculation part of the expression.
Int32(513)	This value indicates a **calculation pass**.
Int32(514)	This value indicates the depth of the calculation pass.
Int32(515)	This value indicates a solve order.
Int32(516)	This value indicates a Protocol State Mask defined on server: 0x00000001 Disabled; client will ignore: 0x00000002 HIDDEN in schema rowsets: 0x00000004
Int32(528)	This value indicates a Protocol Attribute Mask Has format property (user setting): 0x00000001 Has font name property: 0x00000002 Has font size property: 0x00000004 Has font attribute property: 0x00000008 Has back color property: 0x00000010 Has fore color property: 0x00000020
[String(517)]	Format property value, optional only if the Format Attribute Mask is set (INT32(528) has 0x00000001 set).
[String(518)]	FontName property value, optional only if attribute mask is set (INT32(528) has 0x00000002 set).
[UInt16(519)]	FontSize property value, optional only if attribute mask is set. (INT32(528) has 0x00000004 set).
[UInt16(520)]	FontAttribute property value, optional only if attribute mask is set (INT32(528) has 0x00000008 set).

Element	Description
[UInt32(521)]	Background Color property value, optional only if attribute mask is set (INT32(528) has 0x00000010 set).
[UInt32(522)]	Foreground Color property value, optional only if attribute mask is set (INT32(528) has 0x00000020 set).
String(523)	Background color property value.
String(524)	Foreground color property value.
String(525)	Font size property value.
String(526)	Font attribute property value.
String(527)	Font name property value.

2.2.10.1.3.5.2 AxisInformation

The syntax is defined as follows:

```
AxisInformation =
  OPEN(136)
  Int32(241)
  String(137)
  String(138)
  Int32(139)
  Int32(486)
  Int32(555)
  [AxisDetails]
  CLOSE
```

Element	Description
OPEN(136)	A common data structure that indicates the beginning of the element.
Int32(241)	This value indicates the order of the axis within the client's memory.
String(137)	This value indicates an axis name.
String(138)	Contains MDX that represents an axis description.
Int32(139)	Bitmask that determines how **SETs** are calculated. SET calculated on SERVER 0x00000001 SET is UNNAMED 0x00000002 SET is NONEMPTY 0x00000004 SET is FILTERED on AXIS 0x00000008 SET is DEFFERED 0x00000010 SET is NONREMOTABLE 0x00000020 SET: NEED_RESULT 0x00000040 SET is in CACHE 0x00000080 SET: request REMOTEADVISE 0x00000100 SET: HARD_NONREMOTABLE 0x00000200

Element	Description
Int32(486)	Mask for internal calculation pathways of Vertices.
Int32(555)	This value indicates whether an axis definition comes from the client and therefore follows this integer.
[AxisDetails]	Only when INT32(555) is not 0.
CLOSE	A common data structure that indicates the end of the element.

2.2.10.1.3.5.2.1 Mask for Internal Calculation Handling of Vertex Status/Type

The client and server use this mask to share calculation of vertex values between each other.

Value	Description
0x00000001	Vertex has been deleted.
0x00000002	Vertex defined by server.
0x00000004	Execution deferred to server.
0x00000008	Vertex cannot be remoted.
0x00000020	Vertex used for Visual Totals.
0x00000040	Request remotable status for Vertex.
0x00000080	Either server or client can state it is not the owner of this DPATH.
0x00000100	Vertex is disabled.
0x00000200	Vertex does not own DPATH point.
0x00000400	Vertex From Cell.
0x00000800	Vertex from data definition language (DDL).
0x00001000	Vertex from custom rollup.
0x00002000	Vertex from Visual Totals (alternate method).
0x00004000	Vertex hard not remotable to server.

2.2.10.1.3.5.2.2 AxisDetails

The syntax is defined as follows.

```
AxisDetails =
  OPEN(153)
  AxisStructure (PN_OBJECT_SET)
  AxisTupleData
  CLOSE
```

Element	Description
OPEN(153)	A common element which indicates the beginning of the structure.
AxisStructure	Structure of the axis.
AxisTupleData	Tuple data for the axis.
CLOSE	A common element which indicates the end of the structure.

2.2.10.1.3.5.2.2.1 AxisStructure

The syntax is defined as follows.

```
AxisStructure =
  OPEN (156)
  Int32(157)
  *DimensionInfo
CLOSE
```

Element	Description
OPEN(156)	A common element which indicates the beginning of the structure.
Int32(157)	This value indicates the number of dimensions on the axis.
DimensionInfo	Repeated for each dimension.
CLOSE	A common element which indicates the end of the structure.

2.2.10.1.3.5.2.2.2 DimensionInfo

The syntax is defined as follows:

```
DimensionInfo =
  OPEN(158)
  Int16 (159)
  CLOSE
```

Element	Description
OPEN(158)	A common element which indicates the beginning of the structure.
Int16 (159)	This value indicates the current dimension number.
CLOSE	A common element which indicates the end of the structure.

2.2.10.1.3.5.2.2.3 AxisTupleData

The syntax is defined as follows.

```
AxisTupleData =
```

```
OPEN(160)
Int32(161)
[TupleRecord]
CLOSE
*TupleData
```

Element	Description
OPEN(160)	A common element which indicates the beginning of the structure.
Int32(161)	This value indicates the number of tuples for the axis.
[TupleRecord]	Only when number of tuples is greater than 0.
CLOSE	A common element which indicates the end of the structure.
TupleData	Repeated for each tuple.

2.2.10.1.3.5.2.2.4 TupleRecord

The syntax is defined as follows.

```
TupleRecord =
  Int32(162)
  Int32(163)
  Int32(164)
  Int32(169)
```

Element	Description
Int32(162)	Actual size of page, in bytes. This value is adjusted from a base default value by the server.
Int32(163)	Size of TPath in bytes.
Int32(164)	Number of actual records, based on the adjusted page size in INT32(162).
Int32(169)	Always 0.

2.2.10.1.3.5.2.2.5 TupleData

The syntax is defined as follows.

```
The TPath elements follow one after the other in this object.

TuplePage =
  *TPath
```

Element	Description
TPath	The set of DPATHs that form a tuple.

2.2.10.1.3.5.3 CalculatedMembers

The syntax is defined as follows.

```
CalculatedMembers =
OPEN(140)
  Int16 (141)
  Int16(142)
  DPATH(143)
  String(144)
  Int32(145)
  Int32(241)
  Int32(583)
  [String(146)]
  String(252)
  Int32(147)
  Int32(406)
  Int32(485)
  Int32(531)
  [String(532)]
  [UInt16(533)]
  [UInt16(534)]
  [UInt32(535)]
  [UInt32(536)]
  String(537)
  String(538)
  String(539)
  String(540)
  String(541)
  UInt16(554)
CLOSE
```

Element	Description
OPEN(140)	A common data structure that indicates the beginning of the element.
Int16 (141)	This value indicates the current level.
Int16(142)	This value indicates the number of levels.
DPATH(143)	This value indicates the DPATH to a calculated member.
String(144)	This value indicates a member name.
Int32(145)	This value indicates the ID of the Vertex assigned by the query engine (for internal calculation).
Int32(241)	This value indicates the current number of calculated members.
Int32(583)	This value indicates a Cookie. The same value is passed back and forth between the client and server. A value of 0 specifies that it comes from client; otherwise it is an internal cookie that is used by the server.
[String(146)]	Returned only when INT32(583)=0. Contains MDX formula text for the CalculatedMember.
String(252)	This value indicates a format string. For example, Standard.

Element	Description
Int32(147)	This value indicates a solve order.
Int32(406)	This value indicates whether formula is hidden.
Int32(485)	Mask for internal calculation pathways of Vertices. See section 2.2.10.1.3.5.2.1 for more information. Used in calculated members.
Int32(531)	This value indicates whether Calculated Member has the following attributes. Has format property 0x00000001 Has font name property 0x00000002 Has font size property 0x00000004 Has font attribute property 0x00000008 Has back color property 0x00000010 Has fore color property 0x00000020
[String(532)]	FontName property value, optional.
[UInt16(533)]	FontSize property value, optional.
[UInt16(534)]	FontAttribute property value, optional.
[UInt32(535)]	Background Color property value, optional.
[UInt32(536)]	Foreground Color property value, optional.
String(537)	MDX Background Color property value.
String(538)	MDX Foreground Color property value.
String(539)	MDX Font size property value.
String(540)	MDX Font attribute property value.
String(541)	MDX Font Name property value.
UInt16(554)	If NonEmptyBehavior not set by user, this is 0. If NonEmptyBehavior is set, this value is the ID of the Measure used for NonEmptyBehavior.
CLOSE	A common data structure that indicates the end of the element.

2.2.10.1.3.5.4 SessionQueryContext

The syntax is defined as follows:

```
SessionQueryContext=
  OPEN(245)
  *CellSecurityRestriction
  CLOSE
```

Element	Description
OPEN(245)	A common element which indicates the beginning of the structure.

Element	Description
CellSecurityRestriction	Repeated for each restriction.
CLOSE	A common element which indicates the end of the structure.

2.2.10.1.3.5.4.1 CellSecurityRestriction

The syntax is defined as follows:

```
CellSecurityRestriction  =
  Byte(246)
  Int32(249)
  Int32(250)
  [String(247)]
```

Element	Description
Byte(246)	This value indicates cell security type. A single ASCII character is transmitted. 'W' GRANT CELL WRITE 'R' GRANT FULL CELL READ 'C' GRANT CONTINGENT CELL READ
Int32(249)	0 = There is no formula text. Nonzero = Formula text is present.
Int32(250)	This value indicates a solve order.
[String(247)]	Only when INT32(249)is not equal to 0 and not equal to (0x00000001).

2.2.10.1.3.5.5 SessionQueryContextLevelCustomRollups

The syntax is defined as follows:

```
SessionQueryContextLevelCustomRollups =
  OPEN(553)
  *LevelCustomRollup
  CLOSE
```

Element	Description
OPEN(553)	A common element which indicates the beginning of the structure.
LevelCustomRollup	Repeated for each level.
CLOSE	A common element which indicates the end of the structure.

2.2.10.1.3.5.5.1 LevelCustomRollup

The syntax is defined as follows:

```
LevelCustomRollup =
  INT16(355)
  String(356)
```

Element	Description
INT16(355)	This value indicates whether it has custom rollup (custom level formula).
String(356)	Only present if INT16(355) is TRUE.

2.2.10.1.3.5.6 StatefulVisualTotals

The syntax is defined as follows.

```
StatefulVisualTotals =
  Int32(565)
  *DimensionMemberStatefulVisualTotal
```

Element	Description
Int32(565)	This value indicates the number of calculated members.
DimensionMemberStatefulVisualTotal	CalculatedMember content, repeated for each Calculated member.

2.2.10.1.3.5.6.1 DimensionMemberStatefulVisualTotal

The syntax is defined as follows.

```
DimensionMemberStatefulVisualTotal =
  Int32(564)
  [DimensionMemberStatefulVisualTotalDetail]
```

Element	Description
Int32(564)	This value indicates whether there are there any Stateful Visual Totals.
[DimensionMemberStatefulVisualTotalDetail]	Contents of the Member's Stateful Visual Totals, if INT32(564) = TRUE.

2.2.10.1.3.5.6.1.1 DimensionMemberStatefulVisualTotalDetail

The syntax is defined as follows.

```
DimensionMemberStatefulVisualTotalDetail =
  OPEN(556)
   Int16(557)
   Int16(558)
   DPATH(559)
   Int32(560)
   String(561)
```

```
 Int32(562)
 Int32(563)
 AxisDetails
CLOSE
```

Element	Description
OPEN(556)	A common element which indicates the beginning of the structure.
Int16(557)	This value indicates the current level.
Int16(558)	This value indicates the number of levels.
DPATH(559)	This value indicates the DPath of an internally created member that serves for the visual total calculation.
Int32(560)	This value indicates an ID of an internally created member that serves for the visual total calculation.
String(561)	This value indicates a Name for an internally created member that serves for the visual total calculation.
Int32(562)	This value indicates a solve order.
Int32(563)	Mask for internal calculation pathways of Verteces. See section 2.2.10.1.3.5.2.1 for more information.
AxisDetails	Structure and tuple data of an axis.
CLOSE	A common element which indicates the end of the structure.

2.2.10.1.3.5.6.1.2 DimensionMemberFormula

The syntax is defined as follows.

```
DimensionMemberFormula =
  OPEN(153)
  AxisStructure
  AxisTupleData
  CLOSE
```

Element	Description
OPEN(153)	A common element which indicates the beginning of the structure.
AxisStructure	Structure of the axis, including dimensions along the axis.
AxisTupleData	Data records for the tuples.
CLOSE	A common element which indicates the end of the structure.

2.2.10.2 Calculate MDX Fragment Response

The Calc Query response is sent in response to the Calc Query request. See section 4.3.2 for an example.

The syntax is defined as follows:

```
RESPONSEDATA =
  [STATUS]
  QueryResult
```

Element	Description
STATUS	Only sent over TCP/IP, and not over HTTP. The RESPONSESTATUS MUST have value of 1.
QueryResult	The detailed contents of the query result.

2.2.10.2.1 QueryResult

The syntax is defined as follows:

```
QueryResult =
  OPEN(168)
  OPEN(151)
  Int32(152)
  *AxisDetails
  [StatefulVisualTotals]
  CLOSE
  [ISOLATEDCache]
  CLOSE
```

Element	Description
OPEN(168)	A common element which indicates the beginning of the structure.
OPEN(151)	A common element which indicates the beginning of the structure.
Int32 (152)	This is the same value that was passed in as INT32REQFLAGS.
AxisDetails	This is repeated for each axis where INT32(486) has 0x00000040 bit set.
[StatefulVisualTotals]	Only when INT32(223) = 3 Execution location is server.
CLOSE	A common element which indicates the end of the structure.
[ISOLATEDCache]	Only when INT32(221) has 0x00000004 set and 0x00000001 not set.
CLOSE	A common element which indicates the end of the structure.

2.2.10.2.1.1 ISOLATEDCache

The syntax is defined as follows:

```
ISOLATEDCache =
  CacheData
  Int32(584)
  *[String(585)]
```

```
[CacheTable]
```

Element	Description
Int32(584)	Count of number of cache strings.
String(585)	Contains the cache string values when client has asked for calculations in Isolation mode.
[CacheTable]	Only when cache has a table.

2.2.10.2.1.1.1 CacheData

The syntax is defined as follows:

```
CacheData = RecordSet
```

Element	Description
RecordSet	Array of pages of Records.

2.2.10.2.1.1.2 RecordSet

```
RecordSet =
  *RecordPage
```

Element	Description
RecordPage	Array of **RecordPages**. Each **RecordPage** is an array of Records.

2.2.10.2.1.1.3 CacheTable

The syntax is defined as follows:

```
CacheTable =
  CacheTableStructure
  CacheTableData
```

Element	Description
CacheTableStructure	Header block for the structure plus array of CacheTableFields.
CacheTableData	Cache data values.

2.2.10.2.1.1.3.1 CacheTableData

The syntax is defined as follows.

```
CacheTableData =
  OPEN(63)
```

```
Int32(198)
*CacheTableRecord
CLOSE
```

Element	Description
OPEN(63)	A common element that indicates the beginning of the structure.
Int32(198)	A value that indicates the number of records. The value is 0 if the number of IO Fields is 0.
CacheTableRecord	An array of CacheTableField. This value is repeated for each record.
CLOSE	A common element that indicates the end of the structure.

2.2.10.2.1.1.3.1.1 CacheTableRecord

The syntax is defined as follows:

```
CacheTableRecord =
  *CacheTableField
```

Element	Description
CacheTableField	This value is repeated for each visible field.

2.2.10.2.1.1.3.1.2 CacheTableField

The syntax is defined as follows:

```
CacheTableField =
  String(64) / Int32(64) / Int64(64)/ Real32(64)/Real64(64)
```

Element	Description
String(64) / Int32(64) / Int64(64)/ Real32(64)/Real64(64)	One of the following values: String value type is String. Int32 value when the type is VT_14 Int64 value when the type is either VT_I8 or VT_CY Real32 value when the type is VT_R4 Real64 value when the type is either VT_R8 or VT_DATE

2.2.10.2.1.1.3.2 CacheTableStructure

The syntax is defined as follows:

```
CacheTableStructure =
  OPEN(191)
```

```
Int32(192)
Int32(232)
Int32(233)
*Field
CLOSE
```

Element	Description
OPEN(191)	A common element which indicates the beginning of the structure.
Int32(192)	A value that indicates the number of IO fields.
Int32(232)	A value that indicates the number of records.
Int32(233)	A value that indicates the number of keys.
*Field	This value is repeated for each field that is visible.
CLOSE	A common element which indicates the end of the structure.

2.2.10.2.1.1.3.2.1 Field

The syntax is defined as follows:

```
Field =
  INT32(364)
  Int16(365)
  String (366)
  Int32 (393)
  Int16(193)
  Int16(194)
  Int16(195)
  Int32(196)
```

Element	Description
INT32(364)	This value indicates a property type. For more information, see PROPERTY_CONTENT_TYPE at [MSDN-MDSCHEMA_PROPERTIES].
Int16(365)	This value indicates a language ID.
String(366)	This value indicates a caption.
Int32(393)	A mask set to 0x00000001 if field is hidden.
Int16(193)	This value indicates a database type. For more information, see [MSDN-DTMRP].
Int16(194)	This value indicates a field variant type. For more information, see [MSDN-Variant.VariantType].
Int16(195)	This value indicates field size in bytes.
Int32(196)	This value indicates Field ID.

2.2.11 Member Name Resolution

The client can request name resolution for a member of a dimension. This section describes the structure of the client request and the server response.

2.2.11.1 Member Name Resolution Request

The client issues a Member Name Resolution request to resolve the member name, the **member qualified name**, or the **member unique name** to its dimension coordinate.

2.2.11.1.1 Request Code

```
REQCODE    = 'N'
```

2.2.11.1.2 PARAM_STRING

The following is the PARAM_STRING specific for this request.

```
PARAM_STRING =
""REQUEST=N;STATE="" FLAGS "";"" CUBEREF
```

2.2.11.1.3 REQDATA

The REQDATA part of the request is as follows.

```
REQDATA =
OPEN(313)
INT32(400)
STRING(314)
CLOSE
```

Value	Meaning
OPEN(313)	Indicates the start of the response.
INT32(400)	The value of the MDX Compatibility property, which specifies how placeholder members in a ragged hierarchy or unbalanced hierarchy are treated. In the context of a NameResolution data structure, if the client issues a name resolution request for a placeholder member, this property instructs the server how to process the request: 0= Default. For compatibility with earlier versions, this is the same as Value 1. The meaning of this default value is subject to change in future versions. 1= Default value. Placeholder members are exposed. 2= Placeholder members are not exposed.
STRING(314)	A string that specifies the name specification.
CLOSE	Indicates the end of the response.

2.2.11.2 Member Name Resolution Response

This section specifies the optional data structure of a NameResolution response for a dimension member. The message is a binary data structure that is transmitted from the server to the client. It specifies the optional response data element RESPONSEDATA that is part of the basic response structure.

The syntax is defined as follows:

```
RESPONSEDATA =
STATUS /
(OPEN(313)
STRING(315)
CLOSE)
```

Value	Meaning
Status	A value that specifies error information if the NameResolution request fails. This value will be the STATUS of the basic response data structure.
OPEN(313)	Indicates the start of the response.
STRING(315)	A string that specifies the unique name of the of dimension member. This value is returned if the NameResolution request succeeds.
CLOSE	Indicates the end of the response.

2.2.12 Refresh Database

The client can request updated information about a specific database. This section describes the structure of the client request and the server response.

2.2.12.1 Refresh Database Request

The client issues a Refresh Database request to check whether it has up-to-date metadata about the specified database.

2.2.12.1.1 Request Code

```
REQCODE   = 'H'
```

2.2.12.1.2 PARAM_STRING

The following is the PARAM_STRING specific for this request.

```
PARAM_STRING =
""REQUEST=G;STATE="" FLAGS "";"" DBREF
```

2.2.12.2 Refresh Database Response

The Refresh Database response is a server response that returns current information about the database metadata on the server.

```
RESPONSE =
OPEN(224)
INT32(225)
REAL64(496)
CLOSE
```

Value	Meaning
Open(224)	Indicates the start of the response.
INT32(225)	A value that specifies the version of the OLAP database.
REAL64(496)	A value that specifies the date and timestamp of the last change to the database.
Close	Indicates the end of the response.

2.2.13 Refresh Cube

The client can request updated information about a specific cube. This section describes the structure of the client request and the server response.

2.2.13.1 Refresh Cube Request

The client issues a Refresh Database request to check whether it has up-to-date metadata about the specified cube.

2.2.13.1.1 Request Code

```
REQCODE   = 'H'
```

2.2.13.1.2 PARAM_STRING

The following is the PARAM_STRING specific for this request.

```
PARAM_STRING =
""REQUEST=G;STATE="" FLAGS "";"" CUBEREF
```

2.2.13.2 Refresh Cube Response

The data structure of the reply varies based on data about the cube. When the request is received, the server gets version information for the cube's dimensions, followed by version information about the cube's structure. If the version information on the server matches the version information cached on the client, the server sends STATUS with RESPONSESTATUS of 1. Otherwise, the server sends STATUS with RESPONSESTATUS of 8.

```
RESPONSEDATA=
STATUS/
NeedRefreshCubeReply
```

Value	Meaning
STATUS	RESPONSESTATUS of 1 indicates that the cube is up to date. RESPONSESTATUS of 8 indicates that the cube is not up to date.
NeedRefreshCubeReply	If the version information does not match (RESPONSESTATUS=8), the server replies by sending updated version information. If RESPONSESTATUS=1, no information is sent.

2.2.13.2.1 NeedRefreshCubeReply

This response data structure specifies the version of the dimension and the number of members in the dimension.

```
NeedRefreshCubeReply=
STATUS
OPEN(182)
INT32(183)
INT32(184)
REAL64(185)
INT32(186)
CLOSE
```

Value	Meaning
STATUS	RESPONSESTATUS MUST be 8.
Open(182)	Indicates the start of the response.
INT32(183)	A value that specifies the dimension version within the cube.
INT32(184)	A value that specifies the version of the cube structure.
REAL64(185)	A value that specifies the date and timestamp of the last change to the cube or dimension.
INT32(186)	A value that specifies the number of dimensions.
Close	Indicates the end of the response.

2.2.13.2.1.1 DimensionVersionReply

This response data structure specifies the version of the dimension and the number of members in the dimension.

```
DimensionVersionReply=
OPEN(187)NT32(188)
INT32(330)
CLOSE
```

Value	Meaning
Open(187)	Indicates the start of the response.
INT32(188)	A count of the number of dimension members.
INT32(330)	A value that specifies the version of the dimension.
Close	Indicates the end of the response.

3 Protocol Details

3.1 Client Details

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves.

The server SHOULD organize its data as per the OLAP data model:

- The server SHOULD contain a collection of databases.
- Each database SHOULD contain a collection of cubes.
- Each cube SHOULD contain a collection of dimensions, measure groups, measures, and sets.
- Each dimension SHOULD contain a collection of hierarchies.
- Each hierarchy SHOULD contain a collection of levels.
- Each level SHOULD contain a collection of members.
- Each member SHOULD contain a collection of properties.
- Each measure group SHOULD contain a collection of measure group dimensions and measures.

For more information about the OLAP data model, see [MSDN-SSAS].

3.1.2 Timers

Connect Timeout: This timer is used during establishment of the session. After establishing a network connection, the Connect Timeout value is used as a timeout value for each network send and receive operation, during the Authentication and Handshake messages in the login sequence. Note that the user perception of connecting includes many network messages, so the usage might not match the user expectations in some situations.<10>

Query Timeout: This timer is used during query request and response. The query timeout is applied to user queries, which can result in none or many request /response pairs. Thus it MUST be applied in the client application layer. For each request the client makes it MUST set the network timeout for the network receive operation. A client observing a network timeout error MUST close the network connection, return an error to the calling application, and later MUST reconnect. Note that the network cannot be used until a response has been completed. The default query timeout value is 60 minutes. The query timeout may be set on the OLE DB connection string or programmatically set for each query.

Ping Period: This is an optional background timer used by the client to check for changes on the server. Often user queries are answered from cached data without communication to the server. This mechanism allows limiting the time lag between possible server data updates and user visibility of those changes. Note that all request/response pairs are serialized on a single connection; therefore it is possible for a user request to occur when a background ping refresh request is in progress, and therefore must wait for completion. By default the ping period is 10 seconds. The ping period may be set on the OLE DB connection string, with a minimum of 250 ms.

3.1.3 Initialization

Initialization of the protocol occurs as the result of a higher layer on the client asking for connection to the server. The connection establishes a session on the server, with state being associated with the session. The state contains security information, which is used to decide what information is allowed to be sent to the client.

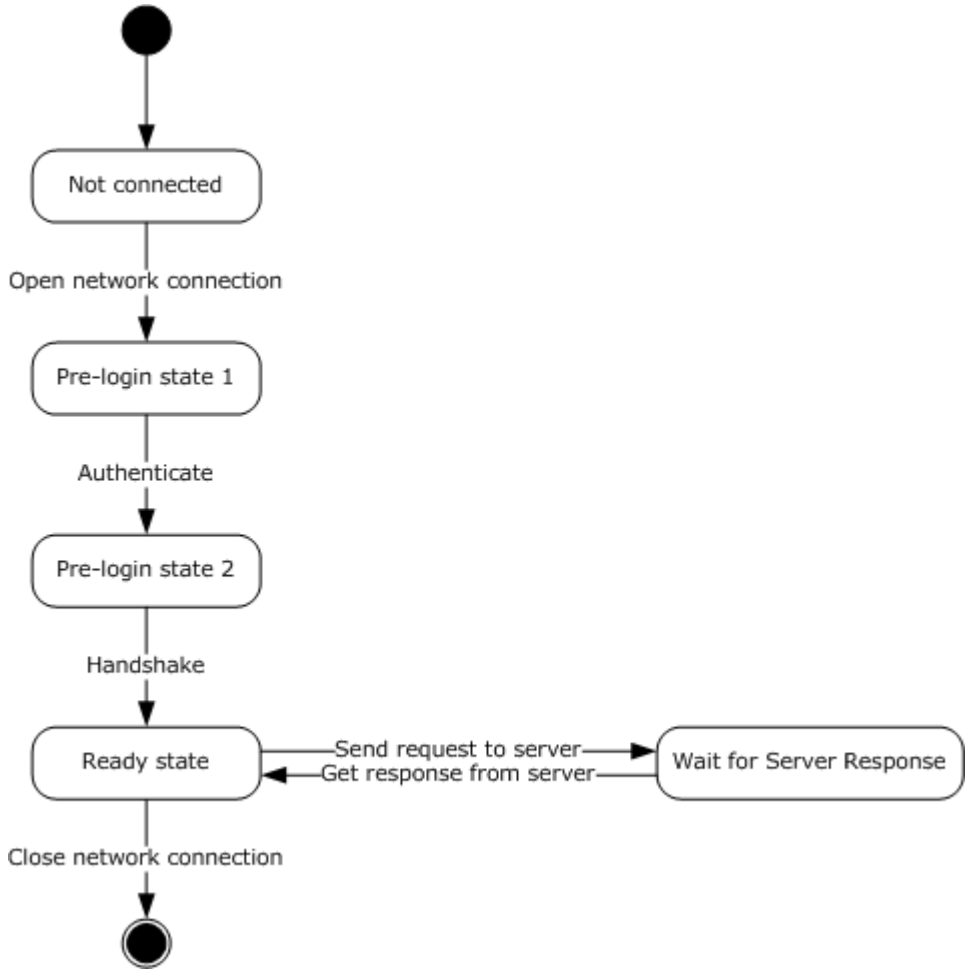


Figure 4: Client connection state diagram (client perspective)

The state machine for clients is depicted in figure 1 above. Note that error handling and timeout logic are not illustrated. Errors or timeout during the login phase MUST be propagated to the higher level caller and MUST close the network connection.

A network connection MUST be established. From the Ready state, the client can send any request message to the server. To reach the ready state, the client MUST first make several requests in the sequence shown in figure 2 below. Note that Authenticate and Handshake actually require several network request/response message pairs, as described in section 2.

The Connect Timeout value is used as a network timeout value for each send and receive operation during the login sequence.

MUST Request to Authenticate
MUST Request to Handshake
SHOULD Request to Get Database Collection

The client network connection is done in a loop using the following pseudocode:

```
Set network timeout to user-specified ConnectTimeout
Repeat
Try to open network connection
Until ( time spent in loop > 100 milliseconds || status == WSAECONNREFUSED || status ==
WSAECONNRESET || status == WSAETIMEDOUT )
```

Note that when using TCP, an implementation MAY set TCP_NODELAY and the receive buffer size, but this is an implementation decision and not part of the protocol.<11>

3.1.3.1 Termination

A session is terminated by the client by closing the network connection. The client does not send a request message to the server.

3.1.4 Higher-Layer Triggered Events

3.1.4.1 Request to Authenticate

As described in section 2.2.2, this event is part of the login sequence. It MUST be followed by a Handshake request. Note that this event requires several client/server round trips.

3.1.4.2 Request to Handshake

As described in section 2.2.3, this event is part of the login sequence. It MUST be preceded by an Authenticate request. Note that this event requires several client/server round trips.

3.1.4.3 Request to Get Database Collection

This event MAY occur at any time after the login sequence.<12>

3.1.4.4 Request to Get Database

This event MAY occur at any time after the login sequence.<13>

3.1.4.5 Request to Get Cube

This event MAY occur at any time after the login sequence.<14>

This request is typically made by the client on behalf of a user query which uses a specific cube. The information is typically cached by the client so subsequent user queries need not send additional requests to the server. The response from the server will include information about the cube, including dimension information. The dimension information might not be complete, subject to a size threshold. The client may need to send a later request to obtain a vertex range by sending the request to Get Dimension Members.

3.1.4.6 Request to Get Dimension Members

This event MAY occur at any time after the login sequence.<15>

3.1.4.7 Request to Get Member Properties

This event MAY occur at any time after the login sequence.<16>

3.1.4.8 Request to Get RecordSet

This event MAY occur at any time after the login sequence.<17>

This request is typically made during query execution. The default client implementation caches this information and future user queries might be answered entirely from cache and not require any server contact.

3.1.4.9 Request to Calc Query

This event MAY occur at any time after the login sequence.<18>

3.1.4.10 Request to Member Name Resolution

This event MAY occur at any time after the login sequence.<19>

3.1.4.11 Request to Refresh Database

This event MAY occur at any time after the login sequence.<20>

3.1.4.12 Request to Refresh Cube

This event MAY occur at any time after the login sequence.<21>

3.1.5 Message Processing Events and Sequencing Rules

3.1.5.1 Sending a Request (All Request Types)

Requests are handled by the server one at a time, in the order received. Therefore, the client SHOULD wait to receive a response to a previous request before sending a new request.<22>

3.1.5.2 Receiving a Response (All Request Types)

This section specifies common steps that MUST be performed whenever the client receives the response to a request that it has sent.

The client MUST associate the response with the request sent. The client MUST only send one request at a time. Multiplexing of multiple requests is not supported, and intermixing more than one request would result in nonsensical stream of bytes (octets) observed by the server.

The client SHOULD implement its own timeout mechanism, for the case of the server not responding within the specified connection timeout or query timeout. Note that if a response to a first request has not been received, it will not be possible to send a second request. In this case, it MAY be necessary to close the connection and establish a new connection.<23>

The client MUST check the status code in the response to determine if the request succeeded. The sections dealing with responses for specified request types can have rules for how to handle certain

failed requests, and any such rules MUST be followed. Errors SHOULD be propagated to the higher level application.<24>

The client MUST check the status code for PN_REFRESH. This indicates the server version is different from the client version. The response might not include all the information requested, depending on which request it is, as specified in section 2. For some application scenarios server updates will be done with high frequency and it is vital that updates be handled correctly. Note that a single user request can result in 0 or many server request/response pairs, so to ensure consistency of the answer to the user, the client SHOULD restart the user request.

3.1.5.3 Receiving a Response for Authenticate

As described in section 2.2.2, this event is part of the login sequence. It MUST be followed by a Handshake request. Note that this event requires several client/server round trips.

3.1.5.4 Receiving a Response for Handshake

As described in section 2.2.3, this event is part of the login sequence. It MUST be preceded by an Authenticate request. Note that this event requires several client/server round trips. It SHOULD be followed by a Get Database Collection request.<25>

3.1.5.5 Receiving a Response for Get Database Collection

This event SHOULD be part of the login sequence, and MAY be done at any later time.<26>

3.1.5.6 Receiving a Response for Get Database

The client MUST first follow the steps as specified in section 3.1.5.2.

3.1.5.7 Receiving a Response for Get Cube

The client MUST first follow the steps as specified in section 3.1.5.2.

The client MUST first follow the steps as specified in section 3.1.5.2.

The client MUST first follow the steps as specified in section 3.1.5.2.

The client MUST first follow the steps as specified in section 3.1.5.2.

The client MUST first follow the steps as specified in section 3.1.5.2.

The client MUST first follow the steps as specified in section 3.1.5.2.

The client MUST first follow the steps as specified in section 3.1.5.2.

The client MUST first follow the steps as specified in section 3.1.5.2.

3.1.6 Timer Events

3.1.6.1 Connect Timer Expires

When the connect timer expires, the client MUST close the network connection and propagate the error to the higher application layer.

3.1.6.2 Query Timer Expires

The query timer is implemented above the protocol, and is used to specify a timeout for the network receive. When a network timeout occurs, the client MUST propagate the error to the higher application layer and close the network connection.

3.1.6.3 Ping Timer Expires

When the ping period expires, the client SHOULD MAKE a request for Refresh Database and Refresh Cube for all open cubes. The client SHOULD use locking to serialize request / response of foreground and background activities over the network connection.<27>

3.1.7 Other Local Events

3.1.7.1 Network Connection Is Disconnected or Encounters Error

A client terminates the session by closing the network connection, without sending a request to the server. The server will then detect that the TCP connection has been dropped and will clean the state associated with this connection.

If the client discovers any network errors, it SHOULD propagate the error to the higher level application and close the network connection. If time remains for the high-level operation the client MAY try reconnecting and retrying the failed operation.<28>

3.2 Server Details

3.2.1 Abstract Data Model

Unless otherwise specified, the protocol MUST report the occurrence of an error to the higher layer, stop all timers, and stop processing the current request. The message details in section 2 discuss error codes and the format which MUST be used for including them in the response to the client.

The server MUST maintain state for each connected client. The protocol does not dictate how the state is stored by the server. The state MUST include security information, used to govern what information may be sent to the client. The state MUST includes version information, used to identify if the client version is different and requires a refresh.

Client server communication MUST first follow a login sequence as described in section 2.2.2 and illustrated in figure 2 in section 3.1.3.

3.2.2 Timers

Default Send Timeout: This timer is used by the server during sending responses to a client. In the OLAP Server implementation, the default value is 60 seconds, unless specified in the Windows Registry for OLAP Server, which is initially created by Setup with the value 60 seconds.

3.2.3 Initialization

The protocol MUST first follow a login sequence as described in section 2.2.2 and figure 2 in section 3.1.3. Following a successful login sequence, other requests MAY be serviced.<29>

3.2.4 Higher-Layer Triggered Events

3.2.4.1 Request to Authenticate

As described in section 3.1.3, this event is part of the initialization sequence. It MUST be followed by a Handshake request. Note that this event requires several client/server round trips.

3.2.4.2 Request to Handshake

As described in section 2.2.3, this event is part of the login sequence. It MUST be preceded by an Authenticate request. Note that this event requires several client/server round trips.

3.2.4.3 Request to Get Database Collection

This event MAY occur at any time after the login sequence.<30>

3.2.4.4 Request to Get Database

This event MAY occur at any time after the login sequence.<31>

3.2.4.5 Request to Get Cube

This event MAY occur at any time after the login sequence.<32>

3.2.4.6 Request to Get Dimension Members

This event MAY occur at any time after the login sequence.<33>

3.2.4.7 Request to Get Member Properties

This event MAY occur at any time after the login sequence.<34>

3.2.4.8 Request to Get RecordSet

This event MAY occur at any time after the login sequence.<35>

3.2.4.9 Request to Calc Query

This event MAY occur at any time after the login sequence.<36>

3.2.4.10 Request to Member Name Resolution

This event MAY occur at any time after the login sequence.<37>

3.2.4.11 Request to Refresh Database

This event MAY occur at any time after the login sequence.<38>

3.2.4.12 Request to Refresh Cube

This event MAY occur at any time after the login sequence.<39>

3.2.5 Message Processing Events and Sequencing Rules

3.2.5.1 Receiving a Request (All Request Types)

This section specifies common steps that MUST be performed whenever the client receives the response to a request that it has sent.

The server MUST complete the response before reading or waiting for subsequent requests. The server MUST only respond to one request at a time. Multiplexing of multiple requests is not supported, and intermixing more than one response would result in nonsensical stream of bytes (octets) observed by the client.

3.2.5.2 Sending a Response (All Request Types)

The server MUST return the correct status code in the response message as defined in section 2. The sections dealing with responses for specified request types can have rules for how to handle certain failed requests, and any such rules MUST be followed. Errors SHOULD be propagated to the higher level application.

The server MUST store the client's last used version of each object and compare it with the current version. If the versions are mismatched, the server MUST return status code for PN_REFRESH.

3.2.6 Timer Events

3.2.6.1 Connect Timer Expires

When the server detects the connect timer has expired, it MUST close the network connection.

3.2.7 Other Local Events

3.2.7.1 Network Connection Is Disconnected or Encounters Error

When the server detects the network connection has been closed, it MUST delete all client state.

If the server encounters any network errors, it MAY propagate the error to the higher level application and close the network connection.<40>

4 Protocol Examples

4.1 HTTP transport examples

4.1.1 Example 1: Response

This example shows a response for a binary package with no length specified as described in section 2.2.1.6.2, Case 1. The communication between the server and client is completed when this response is received.

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.1
Date: Thu, 21 Feb 2008 22:02:32 GMT
Content-Type: text/html
Expires: Thu, 21 Feb 2008 22:02:32 GMT
Cache-control: private
Connection: close
X-Powered-By: ASP.NET
… Encoding of the prefix + Binary Response here …
```

4.1.2 Example 2: Request-Response Pair

This example shows a set of requests and responses that demonstrates scenario of binary package of defined length as described in section 2.2.1.6.2, Case 2. The first request establishes a session and sets a session cookie, which is used for subsequent communication. Chunked encoding is the most common type of request and response.

4.1.2.1 Request 1

```
POST /msolap80/msolap.asp HTTP/1.1
User-Agent: Light DCube
Host: localhost
Content-Length: 70
Proxy-Connection: Keep-Alive
Pragma: no-cache

…Binary request content here…
```

4.1.2.2 Response 1

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.1
Date: Thu, 21 Feb 2008 22:02:32 GMT
Content-Type: text/html
Expires: Thu, 21 Feb 2008 22:02:32 GMT
Cache-control: private
X-Powered-By: ASP.NET
Transfer-Encoding: chunked
Set-Cookie: ASPSESSIONIDQSSAASAC=JKFLOGPCEPJMHEEIKIKMPPCO; path=/
```

```
… Encoding of the prefix + Binary Response here …
```

4.1.2.3 Request 2

```
POST /msolap80/msolap.asp HTTP/1.1
User-Agent: Light DCube
Host: localhost
Content-Length: 70
Proxy-Connection: Keep-Alive
Pragma: no-cache
Cookie: ASPSESSIONIDQSSAASAC=JKFLOGPCEPJMHEEIKIKMPPCO

…Binary request content here…
```

4.1.2.4 Response 2

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.1
Date: Thu, 21 Feb 2008 22:02:32 GMT
Content-Type: text/html
Expires: Thu, 21 Feb 2008 22:02:32 GMT
Cache-control: private
X-Powered-By: ASP.NET
Transfer-Encoding: chunked

… Encoding of the prefix + Binary Response here …
```

4.2 Handshake Examples

4.2.1 Handshake Request Example

See section 2.2.3.1.

```
52 00 45 00 51 00 55 00 45 00 53 00 54 00 3d 00  R.E.Q.U.E.S.T.=.
7c 00 3b 00 53 00 54 00 41 00 54 00 45 00 3d 00  |.;.S.T.A.T.E.=.
30 00 3b 00                                      0.;.
```

4.2.2 Handshake Response Example

See section 2.2.3.

```
STATUS
aa 40 aa 00 00 00 b0 00 04 ff ff 00 00 ab 40 ab
00 00 00 ac 00 04 01 00 00 00 ad 00 04 00 00 00
00 ae 00 04 00 00 00 00 af 00 02 00 00 01 00 00
01 00 00

RESPONSESTATE
```

```
OPEN(202)
ca 40 ca 00 00 00

Array (203, 11)
cb 00 0b 53 63 68 6f 6f 6c 20 32 33 39 00

cb 00: PID 203 (0x00cb)
0b:   size of array 11 bytes
53 63 68 6f 6f 6c 20 32 33 39 00 : ""School 239"" PN_ID_String

Int32(204) ; client Major version

cc 00 04 01 01 00 00
major version = 257

Int32(205) ; client Minor Version
cd 00 04 82 00 00 00

minor version = 130

Int32(549) ; whether client is 64 bit.
25 02 04 00 00 00 00

Int32(251) ; number of members in huge level
fb 00 04 00 00 00 00

Int32(253) ; current Agent index
fd 00 04 00 00 00 00

Int32(419) ; whether Agent is an internal agent;
a3 01 04 00 00 00 00

Int32(369) ; LCID, language setting
71 01 04 09 04 00 00
LCD = 1033 (0x0409), English

Int32(325) ; Client transport type
45 01 04 05 00 00 00
Transporttype = 5 (TCP/IP)

String (287) ; User roles
1f 01 00
string size=0. It is an empty string.

Int32(425) ;Authentication status
a9 01 04 00 00 00 00

Int32(569) ; whether it is DSO
39 02 04 00 00 00 00

Int32(570) ; whether client is SP3 or later
3a 02 04 01 00 00 00
1 mean yes

CLOSE
01 00 00

WriteHandShake
OPEN(202)
```

```
ca 40 ca 00 00 00

OPEN(206)
ce 40 ce 00 00 00
Int32(207) ; proof id, always 0x0239
cf 00 04 39 02 00 00

Int32(208) ; cookie, always 1
d0 00 04 01 00 00 00

Int32(209) ; server major version
d1 00 04 01 01 00 00
SP4: 257(0x0101)

Int32(210) ; server minor version
d2 00 04 82 00 00 00
SP4: 130 (0x0082)

Int32(211) ; Major build number 1, always 0
d3 00 04 00 00 00 00

Int32(212) ; Major build number 2, always 0
D4 00 04 00 00 00 00

Int32(213) ; Minor build number 1, always 0
D5 00 04 00 00 00 00

Int32(214) ; Minor build number 2, always 0
D6 00 04 00 00 00 00

Int32(550) ; whether server is 64 bit
26 02 04 00 00 00 00
0 means No

Int32(566) ; whether server supports EL4Remoting (Force calculation location to be server)
36 02 04 01 00 00 00
1 means yes

Int32(573) ; whether Sockets use select
3d 02 04 01 00 00 00

Int32(574) ; size of sockets buffer
3e 02 04 b4 05 00 00
Buffer size: 1460 (0x05b4)

Int32(576) ; whether sockets use buffer for send
40 02 04 00 00 00 00

Int32(575) ; whether sockets use nagle
3f 02 04 00 00 00 00

Int32(588) ; whether it is a secure drill through
4c 02 04 01 00 00 00

String(422); product version string (for example, "8.0.2544")
a6 01 14 38 00 2e 00 30 00 30 00 2e 00 32 00 32 00 35 00 34 00 00 00
14: string length is 20 bytes

Int32(215) ; LCID used when server does String Compare
```

```
d7 00 04 09 04 00 00

Int32(216) ; flags used for case sensitive string compare on the server side (default:0)
d8 00 04 00 00 00 00

Int32(217) ; flags used for case insensitive string compare on the server side (default
0x30001)
d9 00 04 01 00 03 00

Int32(239) ; server's flavor
ef 00 04 03 00 00 00
03: ENGINE_ENTERPRISE

Int32(424) ; Authentication status
a8 01 04 02 00 00 00
authentication is done

String(240) ; user name
f0 00 22 52 00 6d 00 6d 00 6d 00 6d 00 6d 00 6d 00 5c 00 6d 00 75 00
6d 00 6d 00 6d 00 6d 00 6d 00 6d 00 00 00
String size: 32 (0x22)

CLOSE
01 00 00
```

4.3 Calculate MDX Fragments Query Examples

4.3.1 Calculate MDX Fragments Request Example

See section 2.2.10.1.

```
52 00 45 00 51 00 55 00 45 00 53 00 54 00 3d 00  R.E.Q.U.E.S.T.=.
51 00 3b 00 53 00 54 00 41 00 54 00 45 00 3d 00  Q.;.S.T.A.T.E.=.
30 00 3b 00 54 00 59 00 50 00 45 00 3d 00 62 00  0.;.T.Y.P.E.=.b.
3b 00 4e 00 41 00 4d 00 45 00 3d 00 46 00 6f 00  ;.N.A.M.E.=.F.o.
6f 00 64 00 4d 00 61 00 72 00 74 00 20 00 32 00  o.d.M.a.r.t. .2.
30 00 30 00 30 00 3b 00 56 00 45 00 52 00 3d 00  0.0.0.;.V.E.R.=.
32 00 3b 00 4c 00 41 00 53 00 54 00 3d 00 4e 00  2.;.L.A.S.T.=.N.
3b 00 54 00 59 00 50 00 45 00 3d 00 6d 00 3b 00  ;.T.Y.P.E.=.m.;.
4e 00 41 00 4d 00 45 00 3d 00 53 00 61 00 6c 00  N.A.M.E.=.S.a.l.
65 00 73 00 3b 00 56 00 45 00 52 00 3d 00 32 00  e.s.;.V.E.R.=.2.
36 00 3b 00 4c 00 41 00 53 00 54 00 3d 00 59 00  6.;.L.A.S.T.=.Y.
3b 00 44 00 56 00 45 00 52 00 3d 00 32 00 36 00  ;.D.V.E.R.=.2.6.
3b 00 43 00 56 00 45 00 52 00 3d 00 32 00 36 00  ;.C.V.E.R.=.2.6.
3b 00 00 00
STATUS

aa 40 aa 00 00 00 b0 00 04 ff ff 00 00 ab 40 ab
00 00 00 ac 00 04 01 00 00 00 ad 00 04 00 00 00
00 ae 00 04 00 00 00 00 af 00 02 00 00 01 00 00
01 00 00

RESPONSEDATA

OPEN(166)
A6 40 a6 00 00 00
```

```
Int32(389) ;Presentation Of Secured Cell Value
85 01 04 00 00 00 00

Int32(397) ; Unique Name Style, 8d 01 04 03 00 00 00
3: Compatible with SQL Server 2000 Analysis Services. The algorithm uses guaranteed unique
names, which are stable over time.

Int32(398) ; MDX Compatibility, 8e 01 04 01 00 00 00
1: Default value. Placeholder members are exposed.

Int32(399) ; SQL Compatibility, 8f 01 04 01 00 00 00

 *DirtyDimensionLevels ; Repeat for each dimension
Int16(403) ; Dirty Level number
93 01 02 ff ff 93 01 02 ff ff 93 01 02 ff ff 93 01 02 ff ff 93 01 02 ff ff 93 01 02 ff ff
93 01 02 ff ff 93 01 02 ff ff 93 01 02 ff ff 93 01 02 ff ff 93 01 02 ff ff 93 01 02 ff ff
93 01 02 ff ff

ff ff: there's no dirty level in the dimension

Int32(167) ; state to tell whether it is a session query only
a7 00 04 00 00 00 00

QueryContextQueryDefinition
OPEN(133)
85 40 85 00 00 00

Int32(221) ;
dd 00 04 01 00 00 80

0x80000001 = PN_SESSION_QUERYCONTEXT | PN_QC_INITIALIZED

Int32(223) ;, df 00 04 04 00 00 00
PN_QC_EXEC_DEFAULT         = 0x00000000
PN_QC_EXEC_SMART           = 0x00000001
PN_QC_EXEC_CLIENT          = 0x00000002
PN_QC_EXEC_SERVER          = 0x00000003
PN_QC_EXEC_FORCE_SERVER    = 0x00000004

Int32(567) ;37 02 04 02 00 00 00

Int32(231)
e7 00 04 00 00 00 00

Int32(501
f5 01 04 00 00 00 00

Real64(502) ; Cache Ratio (default: 0.5),
f6 01 08 00 00 00 00 00 00 e0 3f
0.5

Int32(542)
1e 02 04 01 00 00 00

Int32(571)
3b 02 04 00 00 00 00

Int32(572))
```

```
3c 02 04 32 00 00 00

Real64(586) ; Sonar Cache Ratio (default: 0.5)
4a 02 08 00 00 00 00 00 00 e0 3f

Int32(587) ; number of Remaining Seconds (For time out purpose)
4b 02 04 10 0e 00 00
Time out: 3600 seconds

Int16(134) ; number Of sets
86 00 02 00 00

Int16(135) ; number Of calculated members
87 00 02 02 00

Int16(227) ; number of user defined functions
e3 00 02 00 00

Int16(248) ; number of cell security restrictions
f8 00 02 03 00

CellFormulas
OPEN(503)
f7 41 f7 01 00 00

Int32(504) ;Number of Cell Formulas
f8 01 04 00 00 00 00

CLOSE
01 00 00

OPEN(136)
88 40 88 00 00 00

Close
01 00 00

OPEN(140)
8c 40 8c 00 00 00
*CalculatedMembers; Repeated for each calculated member (server side and client side)
Int16 (141); current Level
8d 00 02 02 00

Int16(142); number of Levels
8e 00 02 02 0

Array(143, size(Int16) * number of Levels); read level path
8f 00 04 00 00 02 08

String(144); name
90 00 0e 50 00 72 00 6f 00 66 00 69 00 74 00 00 00
String size: 0e
String: Profit

Int32(145)
91 00 04 01 00 00 f0

Int32(241)
f1 00 04 01 00 00 00
```

```
Int32(583)
47 02 04 10 00 00 00

String(252); format string
fc 00 12 53 00 74 00 61 00 6e 00 64 00 61 00 72 00 64 00 00 00
String size: 12
String: Standard

Int32(147); Solve Order
93 00 04 00 00 00 00

Int32(406); Is Formula Hidden
96 01 04 00 00 00 00

Int32(485); Special Flags
 e5 01 04 06 00 00 00

Int32(531); Protocol Attribute Mask13 02 04 00 00 00 00

String(537) ; MDX Background Color property
19 02 00

String(538) ; MDX Foreground Color property
1a 02 00

String(539) ; MDX Font size property
1b 02 00

String(540) ; MDX Font attribute property
1c 02 00

String(541) ; MDX Font Name property
1d 02 00

UInt16(554) ;NonEmptyBehavior, default 0
2a 02 02 00 00

CalculatedMember
 8d 00 02 02 00 8e 00 02 02 00 8f 00 04 00 00 03 08 90 00 1c 53 00 61 00 6c 00 65 00
73 00 20 00 41 00 76 00 65 00 72 00 61 00 67 00 65 00 00 00 91 00 04 02 00 00 f0 f1
00 04 02 00 00 00 47 02 04 12 00 00 00 fc 00 12 53 00 74 00 61 00 6e 00 64 00 61 00
72 00 64 00 00 00 93 00 04 00 00 00 00 96 01 04 00 00 00 00 e5 01 04 06 00 00 00 13
02 04 00 00 00 00 19 02 00 1a 02 00 1b 02 00 1c 02 00 1d 02 00 2a 02 02 00 00
For calculated member ""Sales Average""

CLOSE
01 00 00
SkipComingObject; this data is ignored
e4 40 e4 00 00 00 01 00 00

[SessionQueryContext] ; only when m_Start has set PN_SESSION_QUERYCONTEXT bit
OPEN(245)
f5 40 f5 00 00 00

 *Restriction  ;repeated for each restriction
Byte(246); type
f6 00 01 57
'W' GRANT CELL WRITE
```

```
Int32(249); hasFormulaText
f9 00 04 00 00 00 00

Int32(250); Solve order
fa 00 04 00 00 00 00

Restriction
f6 00 01 52 f9 00 04 00 00 00 00 fa 00 04 00 00 00 00 f6 00 01 43 f9 00 04 00 00 00 00
fa 00 04 00 00 00 00
repeated for the other 2 cell security restrictions

CLOSE
01 00 00

Int8(552) ; number of User defined default members
28 02 01 00

[SessionQueryContextLevelCustomRollups] OPEN(553)
29 42 29 02 00 00

 *LevelCustomRollup ;Repeated for each level
INT16(355) ; Whether it has Custom Rollup (custom level formula)
63 01 02 00 00 63 01 02 00 00 63 01 02 00 00 63 01 02 00 00 63 01 02 00 00 63 01 02
00 00 63 01 02 00 00 63 01 02 00 00 63 01 02 00 00 63 01 02 00 00 63
01 02 00 00 63 01 02 00 00 63 01 02 00 00 63 01 02 00 00 63 01 02 00
00 63 01 02 00 00 63 01 02 00 00 63 01 02 00 00 63 01 02 00 00 63 01
02 00 00 63 01 02 00 00 63 01 02 00 00 63 01 02 00 00 63 01 02 00 00
63 01 02 00 00 63 01 02 00 00 63 01 02 00 00 63 01 02 00 00 63 01 02
00 00 63 01 02 00 00 63 01 02 00 00 63 01 02 00 00 63 01 02 00 00 63
01 02 00 00 63 01 02 00 00 63 01 02 00 00

CLOSE
01 00 00

[StatefulVisualTotals]
Int32(565) ; number of Calculated Members
35 02 04 02 00 00 00

 *DimensionMemberStatefulVisualTotal  ; repeated for each Dimension Member
 Int32(564)   ; Are there any Stateful Visual Total?
34 02 04 00 00 00 00 34 02 04 00 00 00 00

CLOSE
 01 00 00
```

4.3.2 Calculate MDX Fragment Response Example

See section 2.2.10.2.

```
STATUS
aa 40 aa 00 00 00 b0 00 04 ff ff 00 00 ab 40 ab
00 00 00 ac 00 04 01 00 00 00 ad 00 04 00 00 00
00 ae 00 04 00 00 00 00 af 00 02 00 00 01 00 00
01 00 00
```

```
OPEN(168)
a8 40 a8 00 00 00

OPEN(151)
97 40 97 00 00 00

 Int32(152) ;state
98 00 04 00 00 00 00

 *AxisDetails ;
OPEN(153)
99 40 99 00 00 00

AxisStructure 9c 40 9c 00 00 00 9d 00 04 01 00 00 00 9e 40 9e 00 00 00 9f 00 02 06 00 01 00
00 01 00 00

AxisTupleData
a0 40 a0 00 00 00 a1 00 04 02 00 00 00 a2 00 04 00 01 00 00 a3 00 04 0a 00 00 00 a4 00 04
00 00 00 00 a9 00 04 00 00 00 00 01 00 00

CLOSE
01 00 00

AxisDetails
99 40 99 00 00 00 9c 40 9c 00 00 00 9d 00 04 01 00 00 00 9e 40 9e 00 00 00 9f 00 02 00
00 01 00 00 01 00 00 a0 40 a0 00 00 00 a1 00 04 08 00 00 00 a2 00 04 00 01 00 00 a3 00
04 02 00 00 00 a4 00 04 00 00 00 00 a9 00 04 00 00 00 00 01 00 00 01 00 00

AxisDetails
99 40 99 00 00 00 9c 40 9c 00 00 00 9d 00 04 02 00 00 00 9e 40 9e 00 00 00 9f 00 02 08
00 01 00 00 9e 40 9e 00 00 00 9f 00 02 06 00 01 00 00 01 00 00 a0 40 a0 00 00 00 a1 00
04 02 00 00 00 a2 00 04 49 12 00 00 a3 00 04 0e 00 00 00 a4 00 04 00 00 00 00 a9 00 04
00 00 00 00 01 00 00 01 00 00

[StatefulVisualTotals]
35 02 04 02 00 00 00 34 02 04 00 00 00 00 34 02 04 00 00 00 00

CLOSE
01 00 00

[ISOLATEDCache] CacheData
7f 40 7f 00 00 00 80 00 04 00 00 00 00 81 00 04 10 00 00 00 82 00 04 00 00 00 00 83 00 04
f1 02 00 00 84 00 02 57 00 40 01 04 00 00 00 00 01 00 00

Int32(584) ; Cache strings
48 02 04 00 00 00 00

[CacheTable] ;Only when cache has table
CacheTableStructure
bf 40 bf 00 00 00 c0 00 04 02 00 00 00 e8 00 04 03 00 00 00 e9 00 04 00 00
00 00 6c 01 04 00 00 00 00 6d 01 02 00 00 6e 01 00 89 01 04 00 00 00 00 c1
00 02 03 00 c2 00 02 03 00 c3 00 02 04 00 c4 00 04 00 00 00 00 c5 00 00 6c
01 04 00 00 00 00 6d 01 02 00 00 6e 01 00 89 01 04 00 00 00 00 c1 00 02 82
00 c2 00 02 1f 40 c3 00 02 04 00 c4 00 04 01 00 00 00 c5 00 00 01 00 00

CacheTableData
3f 40 3f 00 00 00 c6 00 04 03 00 00 00 40 00 04 01 00 00 00 40 00 12 53 00 74 00 61 00
6e 00 64 00 61 00 72 00 64 00 00 00 40 00 04 01 00 00 00 40 00 12 43 00 75 00 72 00 72 00
65 00 6e 00 63 00 79 00 00 00 40 00 04 01 00 00 00 40 00 04 23 00 00 00 01 00 00
```

```
CLOSE
01 00 00
```

4.4 STATUS Example

See section 2.2.1.3.

```
OPEN(170)
aa 40 aa 00 00 00
aa 40
id: 170 (0x00aa).   idObj = 0x40aa
aa 00
id: 170
00
State: 0
00
Length field, must be 0
Int32 (176) ;
b0 00 04 ff ff 00 00
b0 00
id: 176 (0x00b0)
04
Size: 4 bytes for Int 32
ff ff 00 00
OPEN (171)
ab 40 ab 00 00 00
Int32 (172);
ac 00 04 01 00 00 00
1
Int32(173);
ad 00 04 00 00 00 00
0
Int32(174);
ae 00 04 00 00 00 00
0
String (175):
af 00 02 00 00 00
string size =2. String is empty since it only contains the end character.
CLOSE
01 00 00
01 00
00
Size, must be 0
CLOSE
01 00 00
```

4.5 Get Record Set

4.5.1 Get RecordSet Request Example

See section 2.2.9.1.

```
52 00 45 00 51 00 55 00 45 00 53 00 54 00 3d 00  R.E.Q.U.E.S.T.=.
40 00 3b 00 53 00 54 00 41 00 54 00 45 00 3d 00  @.;.S.T.A.T.E.=.
61 00 30 00 30 00 30 00 30 00 3b 00 54 00 59 00  a.0.0.0.0.;.T.Y.
50 00 45 00 3d 00 62 00 3b 00 4e 00 41 00 4d 00  P.E.=.b.;.N.A.M.
45 00 3d 00 46 00 6f 00 6f 00 64 00 4d 00 61 00  E.=.F.o.o.d.M.a.
72 00 74 00 20 00 32 00 30 00 30 00 30 00 3b 00  r.t. .2.0.0.0.;.
56 00 45 00 52 00 3d 00 32 00 3b 00 4c 00 41 00  V.E.R.=.2.;.L.A.
53 00 54 00 3d 00 4e 00 3b 00 54 00 59 00 50 00  S.T.=.N.;.T.Y.P.
45 00 3d 00 6d 00 3b 00 4e 00 41 00 4d 00 45 00  E.=.m.;.N.A.M.E.
3d 00 53 00 61 00 6c 00 65 00 73 00 3b 00 56 00  =.S.a.l.e.s.;.V.
45 00 52 00 3d 00 32 00 36 00 3b 00 4c 00 41 00  E.R.=.2.6.;.L.A.
53 00 54 00 3d 00 59 00 3b 00 44 00 56 00 45 00  S.T.=.Y.;.D.V.E.
52 00 3d 00 32 00 36 00 3b 00 43 00 56 00 45 00  R.=.2.6.;.C.V.E.
52 00 3d 00 32 00 36 00 3b 00 31 31 31 31 31 32  R.=.2.6.;.111112
31 32 32 31 31 31                                122111

31 31 31 31 31 32 31 32 32 31 31 31: DATASET
```

4.5.2 Get RecordSet Response Example

See section 2.2.9.2.

```
OPEN (127)
7f 40 7f 00 00 00

INT32 (128)     ; number of Segments
80 00 04 00 00 00 00

 INT32(129)   ;  number of Records
81 00 04 02 00 00 00
2 records

RecordsDescription

INT32(130)  ;number of pages
82 00 04 00 00 00 00

INT32(131) ; number of records per page
83 00 04 05 05 00 00
1285 celles per page

INT16(132);  size of a record (in bytes)
84 00 02 33 00
51 bytes per cell

INT32(320)
40 01 04 00 00 00 00

CLOSE
01 00 00

Record
00 04 00 00 00 00 06 00 80 04 00 00 00 00 00 e0
2a f0 40 ef 5f 76 4f 44 82 eb 40 80 14 ae 47 33
37 01 41 2a 54 00 00 00 00 00 00 ec f8 a0 67 44
ad f4 40 00 04 00 00 00 00 06 00 00 05 00 00 00
```

```
00 00 30 c9 f0 40 d5 4b 37 89 95 40 ec 40 88 14
ae 47 15 ba 01 41 cf 57 00 00 00 00 00 00 22 83
c0 ca df 53 f5 40

00 04 00 00 00 00 06 00 80 04 00: Path for record 1
00 04 00 00 00 00 06 00 00 05 00: Path for record 2
For both records, there are 5 measure values. Each measure is a Double (8 bytes).
```

5 Security

5.1 Security Considerations for Implementers

The underlying server data which can be communicated by this protocol might be of a sensitive nature. In addition, the server might be returning potentially sensitive data in its responses. The server itself has security built in at many levels of the OLAP model. For instance, at the database, cube, dimension, and cell level, the server will only return responses with data to authenticated users who specifically have access to the part of the OLAP data model that the user is requesting access to, through this protocol.

This protocol offers an authentication sequence to assure that the server has established the user identity of the client, which is used by the server to determine which data on the server is accessible to the client. This is based on SSPI function calls with ""NTLM"" package. See section 2.2.2 for more information.

At the transport layer, if not using TCP, HTTPS SHOULD be used instead of HTTP to ensure the integrity of the protocol transmissions and to prevent tampering, unauthorized access to data in the protocol, and to prevent interception of data content while in transport.

5.2 Index of Security Parameters

None.

6 Appendix A: Error Messages

6.1 RESPONSEERRORCODE

This section lists the error codes for RESPONSEERRORCODE. In this protocol, only the numeric code is sent from the server to the client. However, the message text has been provided to enable the implementer to map the numeric error code to the message.

Numeric Value	Message Text
110	- resource access error
111	- resource ACL access error
153	- the user could not be authenticated
3042	- the aggregate function cannot operate on calculated measure ""%s""
3009	- bind request is not valid
3090	- <set>.Current cannot be called in this context
3125	- dimension '%s' does not belong to the specified cube
3012	- dimension count is not valid
3010	- dimension ID is not valid
3045	- usage is not valid: A nontuple formula was used as a tuple
3015	- level ID is not valid
*** value not found ***	
3011	- object type is not valid
3022	- StrToArray conversion usage is not valid
3029	- property name is not valid: ""%s""
3089	- context for requesting properties is not valid (you may have requested a property of a calculated member)
3021	- the string ""%s"" is not a valid VisualTotals string. Use the asterisk (*) character.
3119	- The subcube expression must be one of the following: decendants(...) function, <level>.members, <dimension>.members, or a single member.
0x80000020	- cannot convert dimension to %s
3033	- cannot bind: unknown dimension: ""%s""
3036	- cannot bind: unknown dimension or level: ""%s""
3039	- cannot bind: unknown name: ""%s""

Numeric Value	Message Text
3037	- cannot bind: unknown dimension or member: ""%s""
3038	- cannot bind: unknown level or member: ""%s""
3034	- cannot bind: unknown level: ""%s""
3035	- cannot bind: unknown member: ""%s""
3019	- cannot bind: the name: ""%s"" is unbindable in the current context
0x80000030	- cannot convert level to %s
0x80000040	- cannot convert member to %s
0x80000060	- cannot convert set to %s
0x80000050	- cannot convert tuple to %s
0x80000000	- cannot convert <unknown> type to %s
0x80000010	- cannot convert expression to %s
3159	- the DistinctCount function can only be used as a top-level (most outer) function in a calculated measure definition
3164	- the model ""%s"" is not a grouping cube. Only grouping cubes can be dropped.
3126	- the dimension ""%s"" is a version 7.0 virtual dimension. It cannot be used in grouping
3169	- failed load of sqldata.dll, which sqlsort.dll depends on.
3165	The parent member is at the highest level, so you cannot move or create child members under it. This would involve changing the parent member's key.
3157	- the dimension ""%s"" is a version 7.0 virtual dimension. It cannot be used in slicing.
3097	- cannot remove member ""%s"". This member has associated data in a cube. It is not possible to move members that have associated data.
2084	Attempt to create cube %1 from the cube %2. This cannot be done, because the source cube has cell security defined.
3057	- you are not authorized to access the cell. For more information, see your administrator.
5211	Infinite recursion detected during execution of calculated cell formula %1 at pass %2.
1023	- the underlying data structures are not valid
5204	The cube '%s' does not exist, or it is not processed

Numeric Value	Message Text
3100	- query to the source data is too long.
*** value not found ***	Cube data was changed during the cell allocation. The UPDATE operation was canceled and needs to be reapplied
3110	- too many levels specified for a DDL statement
3087	- he member ""%s"" has an invalid value for the number of skipped levels
3147	Formula error - the cube name of the dimension ""%1"" must match the source cube name ""%2"".
3155	- the dimension of the source member ""%s"" must match the source dimension.
3141	- invalid reference - the dimension ""%s"" does not exist.
3080	- the dimension ""%s"" is not writable
5207	Some dimension(s) were changed on the server. The query is no longer valid.
3008	- aggregations are not supported for the DISTINCT COUNT measure ""%s""
3027	- incompatible types in operation '/' (division)
3000	- a formula attempted to divide an integer by zero
3140	- a data mining dimension (""%s"") cannot declare its structure definition
3031	- drill specifier has more than one dimension; it must have only one
3030	- cannot match any dimension for drilling
3096	- the cube is not enabled for drillthrough
2998	- an object whose name you are searching for was dropped and is not available for this operation
5210	Dimension security on dimension %1 does not support this drillthrough
5209	Cannot execute drillthrough on calculated member %2 in dimension %1.
3023	- duplicate dimensions across (independent) axes
3076	- duplicate property
3121	- duplicate property name
5215	Detected two sibling members with the same name: %1.
5004	Error in the definition of the dimension security for the dimension

Numeric Value	Message Text
	'%1' and role '%2'. Cannot evaluate the expression '%3'
3088	- the effort for processing the query for the ROLAP dimension ""%s"" exceeded the maximum system customized limit
3046	- reference to the filter axis is not supported
3069	- cannot specify a calculated member by key (""%s"")
3003	Infinite recursion detected during execution of calculated member %1.
3072	- formula level is not valid (too deep) - ""%s""
3086	- the function does not support ragged hierarchies
3142	- a dimension (""%s"") of a global cube cannot declare groups of members.
3168	- global cubes do not support distinct count measures.
3145	- global cubes must specify valid storage.
3148	- dimension ""%s"" is not a regular dimension, and it is not supported in global cubes.
3146	- global cubes can only accept a single cube in the source cube list.
3161	- global cubes (slices) must have a valid STORAGE clause.
3139	- a level of type All (in grouping dimension ""%s"") must appear as the first level in the dimension
3136	- a grouping dimension (""%s"") cannot declare a member list for slicing
3132	- the grouping dimension ""%s"" must declare its levels
3153	- the grouping level from one group must have the same parent.
3137	- a grouping dimension (""%s"") must refer a valid number of levels from the source dimension
3135	- a level of a grouping dimension (""%s"") cannot have an alias
3134	- a level of a grouping dimension (""%s"") cannot be declared as hidden
3133	- a level of a grouping dimension (""%s"") cannot declare a property list
3138	- a grouping level (in dimension ""%s"") cannot be the last level in the dimension
3150	- the levels of grouping members must be grouping levels (""%s"" is not a grouping level).
3162	- the cube ""%s"" cannot be used as a grouping source because

Numeric Value	Message Text
	it is a grouping cube.
3082	- the member ""%s"" has infinite loops
5208	The HTTP or HTTPS connection feature is available only with Analysis Services for Microsoft SQL Server 2000 Enterprise Edition. The server you are attempting to connect to is not an Enterprise Edition server.
3028	- incompatible types
3007	- inconsistent cardinality for statistical function
3107	- invalid alias
3112	- unable to refer to a data member (""%s"") in this context
3081	- dimension is not valid (""%s"")
3004	- dimensions are not valid (they do not match)
3108	- invalid parsing entity (""%s"")
3043	- value is not valid for a range (it is less than zero)
3098	- the operator ""%s"" used in a RollupChildren function cannot appear on the first position
3120	- invalid property name
3122	- invalid property value
3099	- the operator ""%s"" used in a RollupChildren function is not valid
3163	- the syntax used to define the nongrouping dimension ""%s"" is not valid.
3078	- invalid dimension member resolution ""%s""
3059	- one or more MOVE statements are not valid (they do not affect any rows)
0x1111003c	- token is not valid: ""%s""
2999	- level name is not unique: ""%s""
3127	- invalid reference: the level ""%s"" does not exist
3085	- levels are not valid (they do not match)
PF_ERROR_LOST_CONNECTION	Connection to the server is lost
PF_ERROR_MACHINE_NOT_FOUND	Unable to connect to the Analysis server. The server name '%s' was not found. Please verify that the name you entered is correct, and then try again.
2091	Database %1 contains more than the maximum of %2 dimensions.

Numeric Value	Message Text
2092	Database %1 contains more than the maximum of %2 levels.
2089	Cube %1 contains more than the maximum of %2 measures.
2087	Virtual cube %1 contains more than the maximum of %2 cubes.
2090	Dimension %1 contains more than the maximum of %2 levels.
2085	Cube %1 contains more than the maximum of %2 dimensions.
2086	Cube %1 contains more than the maximum of %2 levels.
2088	Virtual cube %1 contains more than the maximum of %2 measures.
5214	More than the maximum of 64,000 dimension member children for a single parent (dimension '%1', level '%2', member '%3').
3048	- the maximum size was exceeded
3149	Formula error - the cube name of the measure ""%1"" must match the source cube name ""%2"".
3144	- invalid reference: the measure ""%s"" does not exist.
3156	- the member ""%s"" is already linked.
3131	Formula error - the member ""%1"" is not part of the level ""%2"".
3151	Formula error - the level of the members to be grouped must be below the grouping level (""%1"") -- ""%2"" is above or at the same level.
3025	- incompatible types in operation '-' (minus)
3026	- incompatible types in operation '*' (multiplication)
3158	- the dimension ""%s"" cannot be declared as a grouping dimension because its source dimension already has a grouping dimension assigned to it.
3084	- SKIPPED_LEVELS (%d)must be a positive value
3128	- grouping levels (""%s"") cannot be referred to in global cube creation
3130	- invalid reference: the member ""%s"" does not exist
3129	- invalid reference: the property ""%s"" does not exist
3016	- no set holder
3154	- no source member found for the DDL member (""%s"").
3083	- the member ""%s"" has no valid parent
5212	Access denied to the cube %1.

Numeric Value	Message Text
3160	- the NON_EMPTY_BEHAVIOR property can only be declared on calculated measures.
3152	Formula error - there should be no nongrouping levels between the level of a member that must be grouped (""%1"") and its grouping level (""%2"").
3167	- the multidimensional coordinate on the case dimension must be on the level that corresponds to the key column (%s) of the data mining model
3049	- the target level was expected to be of type CALCULATED
PF_ERROR_NOT_EXIST	- required entity missing (""%s"")
3124	- dimension '%s' is not a regular dimension
2093	A user-defined function from interface %1 is not safe to be called
*** value not found ***	- user-defined function cannot be supported:
5213	Write access denied to the cube %1. Unable to commit changes.
3073	- the name and key should match (no name column) - ""%s""
3077	- the key is NULL (this is not supported)
3040	- cannot calculate dimension for a <null> member
3041	- cannot calculate level for a <null> member
3044	- <null> WHERE clause not supported
3079	- only the SKIPPED_LEVELS property can be specified in MOVE statements
3050	- a virtual dimension can host only one virtual hierarchy
3051	- the functionality is supported only at the session level
3002	- overflow: ""%s""
3106	- error parsing the name of an OLAP mining model column name
3109	- error parsing XML : %s
3024	- incompatible types in operation '+' (plus)
0x1111003b	- unexpected end of expression
3058	- the Analysis server cannot process your request for cell access. For more information, see your administrator.
3111	- invalid cube name: ""%s""
3104	- invalid data mining model: ""%s""
3105	- error retrieving a data mining schema column

Numeric Value	Message Text
5002	Dimension security conflict for dimension '%1'. You belong to the role '%2' that allows levels from %3 to %4, and to another role that allows levels from %5 to %6
3103	- You tried to connect to a version of the Analysis server that does not support anonymous users
3102	- Because the server is running on a version of Windows that does not support anonymous users, you cannot connect as an anonymous user
3143	- slicing is possible only with members of the first level.
PF_ERROR_SQL_NOTIF_ERROR	A SQL Server notification operation failed (error code = %d, operation = '%s').
PF_ERROR_SSPIPACKAGE_NOT_EXIST	- This SSPI package is not installed on this system
3001	- syntax error
3005	- required Time dimension not found
3032	- specified time level type not found across dimension
3060	- too many levels encountered
*** value not found ***	User-defined functions are not allowed by security settings for current connection.
3017	- in user-defined function: expected %d parameters
3054	- in user-defined function: no more than %d parameters can be specified for this user-defined function
3053	- in user-defined function: expected at least %d parameters
3018	- in user-defined function: parameter #%d was expected to be of the type %s
3052	- a user-defined function could not be executed because some of its parameters are of unsupported types
3055	- in user-defined function: for user-defined functions that return variants, variants must be of numeric type
3061	- unbalanced delimiter in ""%s""
3006	- unbound dimension
3014	- unbound level
3013	- unbound member
3062	- unexpected end in ""%s""
3123	- invalid or unsupported user-defined function: ""%s""
3020	- unknown error

Numeric Value	Message Text
3093	Unable update cell because it is on virtual level %1
3091	Unable update cell because it contains calculated member %1
3095	Unable update N/A cell in the virtual cube
3092	Unable update cell because it contains a nonsum measure %1
3094	Unable update cell because it contains a member, %1, that is not at the lowest level
5217	Cannot update cell, because dimension %1 is not included in the virtual cube
3047	- cannot assign NULL value
3056	- user-defined error
2053	- dimension cannot be empty (%s)
3101	- the dimension ""%s"" has no valid member. For more information, see your administrator.
3063	- incorrect delimiter ('%c') in ""%s""
3071	- incorrect parent dimension member type ('%c') for ""%s""
3070	- incorrect dimension member type ('%c') for ""%s""

6.2 RESPONSEERRORCODEEX

This section lists the error codes for RESPONSEERRORCODEEX values when the value of RESPONSESTATUS is equal to -12. In this protocol, only the numeric code is sent from the server to the client. However, the message text has been provided to enable the implementer to map the numeric error code to the message.

Numeric Value	Message Text
0x1111002b	- in the AddCalculatedMembers function
0x11110005	- in the Aggregate function
0x11110068	- in the ALLMembers function
0x11110001	- in the Ancestor function
0x11110063	- in the Ancestors function
0x11110067	- in the Ascendants function
0x11110006	- in the VisualTotals function
0x11110002	- in the First/LastPeriods function
0x11110050	- when calculating a query axis
0x11110071	- in the CalculationCurrentPass function

Numeric Value	Message Text
0x11110070	- in the CalculationPassValue function
0x1111004f	- when calculating the WHERE clause
0x11110007	- in the Children function
0x11110008	- in the CoalesceEmpty function
0x11110009	- in the <constant> function
0x1111000a	- in the Correlation function
0x1111000c	- in the Count function
0x1111000b	- in the Cousin function
0x1111000d	- in the Covariance function
0x1111004e	- in the CreatePropertySet function
0x11110053	- in the CreateVirtualDimension function
0x11110072	- in a property from the custom rollup property list
0x1111000e	- in the Current function
0x11110060	- in an alias MDX language construction
0x11110061	- while creating a virtual cube for this session
0x11110049	- in the internal procedure for drilling down a member
0x1111000f	- in the Descendants function
0x11110077	- in the Difference function
0x11110010	- in a <dimension> base object
0x11110011	- in the Dimensions(<calc>) function
0x11110057	- in the Dimensions.Count function
0x11110046	- in the <object>.Dimension function
0x11110012	- in the Dimensions(<string>) function
0x11110013	- in a <dimension>.<property> function
0x11110079	- in the DistinctCount function
0x1111003f	- in the <dot> function
0x1111004a	- in the internal procedure for drilling up a member
0x1111004c	- in the internal procedure for drilling a level up or down
0x11110075	- in the Extract function
0x11110073	- in the Filter function

Numeric Value	Message Text
0x1111003d	- in base class <formula>
0x11110078	- in the Generate function
0x11110045	- in the <object>.Hierarchy function
0x11110014	- in the IIF function
0x11110076	- in the Intersect function
0x1111006c	- in the IsAncestor function
0x1111006e	- in the IsGeneration function
0x1111006f	- in the IsLeaf function
0x1111006d	- in the IsSibling function
0x1111007a	- in the IsGeneration function
0x1111007b	- in the IsLeaf function
0x1111005e	- in the MDX operator IS
0x11110015	- in the Item function
0x11110016	- in the Item(string,...) function
0x11110017	- in the LastPeriods function
0x11110018	- in a <level> base object
0x11110055	- in the <dimension>.Levels.Count function
0x11110042	- in the Levels(<string>) function
0x11110019	- in the <dimension>.Levels(...) function
0x1111001a	- in the <member>.Level function
0x11110040	- in the <level>.Ordinal function
0x11110066	- in the LinkMember(<object>, <object>) function
0x1111001b	- in the LinRegIntercept function
0x1111001c	- in the LinRegPoint function
0x1111001d	- in the LinRegR2 function
0x1111001e	- in the LinRegSlope function
0x1111001f	- in the LinRegVariance function
0x11110062	- in the LookupCube function
0x11110020	- in a <member> base object
0x11110041	- in the Members(<string>) function

Numeric Value	Message Text
0x11110044	- in the <tuple>.Item(<calc>) function
0x11110021	- in a <member>.<property> function
0x11110022	- in the <object>.Members function
0x11110064	- in the MemberToStr(<member>) function
0x11110023	- in the <object>.Name function
0x1111006a	- in the NameToSet function
0x11110069	- in the NonEmptyCrossjoin function
0x11110024	- in an <object> base class
0x11110025	- in the Lead/Lag function
0x11110026	- in an <operator> function
0x11110074	- in the Order function
0x11110027	- in the ParallelPeriod function
0x11110028	- in the PeriodsToDate function
0x11110003	- in a function operating on one set
0x11110004	- in a function operating on two sets
0x1111005f	- in the Predict function
0x11110029	- in the Range function
0x1111002d	- in the Rank function
0x11110038	- in function VisualTotals -- during run-time definitions
0x1111005d	- in the RollupChildren function
0x1111002a	- in a <set> base object
0x11110051	- in setting a user-defined cache
0x1111006b	- in the calculated cell condition
0x1111002c	- in the { ... } function
0x1111002e	- when referencing a named <set>
0x1111003e	- in the SetToArray function
0x11110030	- in the SetToStr function
0x1111005c	- in the Siblings function
0x11110032	- in a statistical set 1 function
0x11110033	- in a statistical set 2 function

Numeric Value	Message Text
0x1111002f	- in the StripCalculatedMembers function
0x11110065	- in the StrToMember(<string>) function
0x11110034	- in the StrToSet function
0x11110047	- in the StrToTuple(<string>) function
0x1111004b	- in the internal procedure for toggling the drill state of a member
0x1111004d	- in a Top or Bottom function
0x11110035	- in a <tuple> base object
0x11110056	- in the <tuple>.Count function
0x11110043	- in the TupleToStr(<tuple>) function
0x11110039	- in a user-defined function
0x11110048	- in a property of an object
0x11110036	- in a <unary_operator> function
0x1111003a	- in <object>.UniqueName function
0x11110037	- in a name-binding function
0x11110054	- in the UserName function
0x11110052	- in the ValidMeasure function
0x11110031	- in the VisualTotals function

7 Appendix B: Paths

SQL Server Analysis Services Version 8.0 is an application-level protocol that describes the transfer of requests and responses between client and server OLAP applications. More specifically, the protocol describes the binary data structures for authentication, requests, and responses for OLAP data constructs, and requests and responses for formulas and metadata. The data structures that are returned are self-describing.

The main data object that is used in the message exchange between a client and server is the dimension tree. A dimension tree is a multi-level representation of data that consists of a root node and succeeding layers of additional nodes that terminate at leaf nodes that represent the most detailed data. The dimension tree structure, dimension paths, and indexes are best explained through the following examples.

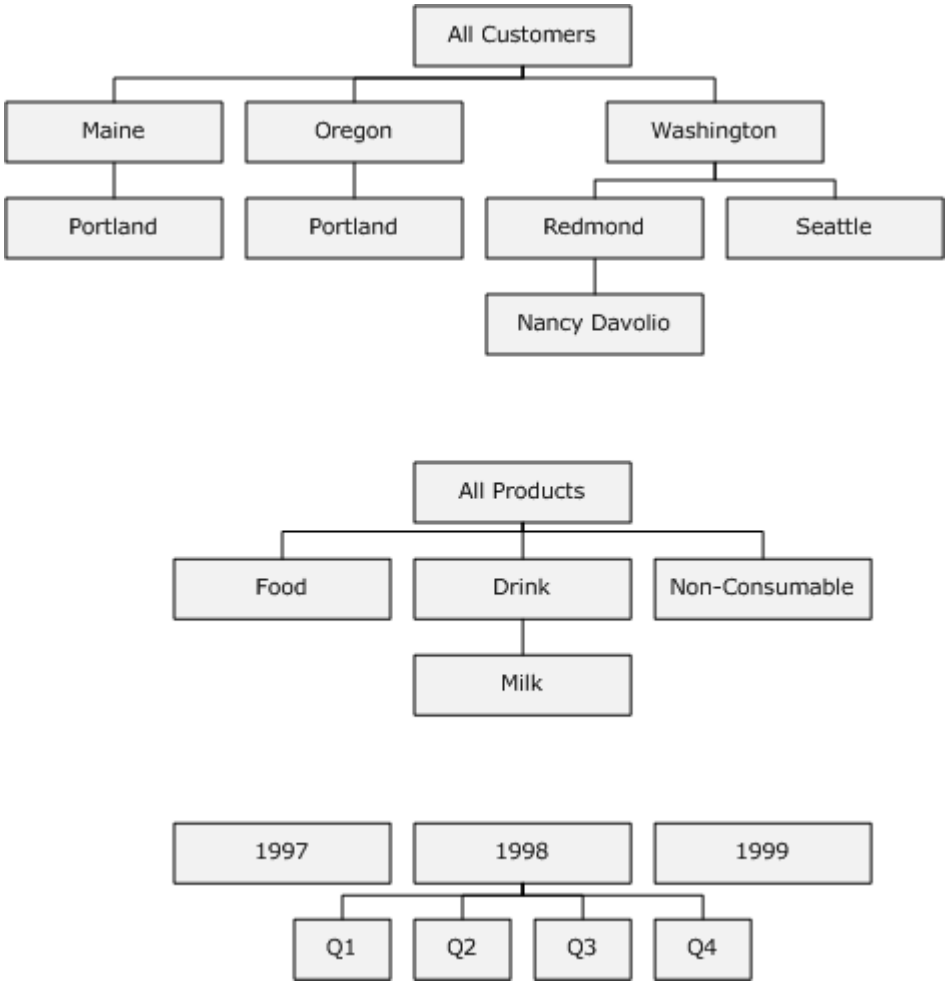


Figure 5: Example dimension trees and dimension members

In Figure 3, the [Customers] dimension tree consists of a root node, inner nodes that correspond to aggregated data for states and cities, and a leaf node that corresponds to the most detailed data, in this case data about a specific person. The root contains the aggregate of all leafs.

Dimension Path (DPath)

The dimension tree structure defines a fully-qualified dimension path to its members. Consider the city of [Seattle]. Using the path provided, the unique name for [Seattle] is [Customers].[All Customers].[Washington].[Seattle]. The path qualifies data constructs that might have the same name, as illustrated by the city of [Portland] which exists in both [Maine] and [Oregon].

Internally, fully qualified paths are represented by an array of numbers, where each position in the array corresponds to a level. The server converts the fully qualified name to a number array. Within the array, the number in a given position corresponds to the one-based index of an item among its siblings, in the natural order. The index of the root is 1. If the member is not a leaf, all positions below the level of the member get the value 0.

The internal name for the array is **DPath**. It represents the internal depiction of a coordinate of a data cell on one of the dimensions.

Using Figure 3 as an illustration, the **DPath** for each level in the [Customers] dimension are as follows:

[1-0-0-0] corresponds to [All Customers]. The zeros correspond to non-leaf levels that are unspecified.

[1-48-0-0] corresponds to [All Customers].[Washington]. In the natural (alphabetical) order of states in the United States, [Washington] is the forty-eighth state.

[1-48-2-0] corresponds to [All Customers].[Washington].[Redmond]. If the database contains customer data for only [Redmond] and [Seattle], and [Redmond] is the second member that was processed, then [Redmond] will be 2. This example emphasizes that **DPath** does not reflect the natural order, but the order in which members are processed.

[1-48-2-5] corresponds to [All Customers].[Washington].[Redmond].[Nancy Davolio]

Dimension Forest

A dimension tree is not required to have a root node. Consider the [Time] dimension in Figure 3. It does not have a root. If a dimension does not have a root, it is called a **dimension forest**. To calculate the **DPaths** of data cells in a dimension forest, the server calculates index of members at the first left by taking their corresponding index in the natural order (for example, the **DPath** of the first quarter in 1999 is [3-1-0].

System Path (Path)

Given that each data cell has a coordinate on each dimension, a single cell might have multiple **DPaths** if it is an intersection point for multiple dimensions. For data cells that correspond to intersection points on multiple dimension, the address scheme is called a **Path** data structure.

A **Path** (or system path) is an array that concatenates the collection of **DPaths** for that data cell. The order of the **DPaths** inside the **Path** depends on the internal order of the dimensions in the cube. The internal order is the order in which the dimensions were added to the cube. The cardinality of **Path** is the number of dimensions in the cube.

The internal order can be found by using OLAP Manager and checking the order of dimensions inside the cube. Programmatically, this may be done by using the Dimensions(<index>) function. For example to discover the first dimension name, use the expression Dimensions(1).UniqueName to return the full-qualified name for that dimension.

Using Figure 3 as an illustration, the Path for the cell ([Customers].[All Customers].[Washington].[Redmond].[Nancy Davolio], [Products].[All Products], [Time].[1998].[Q4]) is [1-48-2-5]-[1-0-0]-[2-4-0].

Measure Extended Path (MPath)

Although each multidimensional data cell holds all the measures, data retrieval as described for the SQL Server Analysis Services Protocol Version 8 is done for only one measure at a time. This explains why a tuple is used to uniquely identify the data cell. The tuple consists of a measure and a **Path**. Internally, this array is referred to as **MPath.**

The index of the measure corresponds to the internal order of the measures inside the cube (again, this may be seen in the OLAP Manager). For example, if total purchases is the second measure defined in the cube, the purchase measure for the above data cell is referred by the following **MPath**:

2-[1-48-2-5]-[1-0-0]-[2-4-0].

The index number that is assigned to a measure remains in effect even if preceding measures are deleted. For example, if the first measure is deleted, total purchases (which is measure 2) retains the index value of 2.

Given an **MPath** value, the system can quickly identify the referred value. However, there is still one question that remains unanswered: what if the data cell is not a value inside the physical cube (that is, it is a formula)? The system must be able to identify this. This is easily achieved by having special values for the elements of the **MPath**, dedicated to formulas. These values are:

- Measure indexes between 1 and 128 are reserved for regular measures (maximum 128 regular formulas).
- Measure indexes between 129 and 32831 are reserved for session and global measure formulas (maximum 32703 session and global measure formulas).
- Measure indexes between 32832 and 65534 are reserved for query measure formulas (maximum 32703 session and global measure formulas).
- Dimension indexes between 1 and 64000 are reserved for regular members (maximum 64000 children under the same parent).
- Dimension indexes between 64001 and 64760 are reserved for session and global regular formulas (maximum 760 session and global regular formulas under the same parent).
- Dimension indexes between 64761 and 65520 are reserved for session and global regular formulas (maximum 760 session and global regular formulas under the same parent).

The indexes of formulas are assigned in the order in which the formula was created. For example, given two session measure formulas, [Cost] and [Volume] (created in this order), and one query formula [Consumable] created under the parent [Products].[All Products], the **MPath** corresponding to the data cell ([Measures].[Volume], ([Customers].[All Customers].[WA].[Redmond].[Cristian Petculescu], [Products].[All Products].[Consumable], [Time].[1998].[Q4])) will be the following:

130-[1-48-2-5]-[1-64761-0]-[2-4-0].

Every cell has coordinates on each and every dimension. For client requests that specify an MDX query, the server formulates a response that includes dimension information in the following ways:

- Dimensions that are not specified on the horizontal and vertical axes are taken from the Where condition.
- If a dimension is neither on an axes nor in a Where condition, the server returns the **MPath** of the default member of that **dimension**. A default member must be explicitly defined as such in the underlying cube. Although the default member of a dimension is one of the members at the first level, you cannot infer which member in that level is the actual default. As such, SQL Server Analysis Services Protocol Version 8 does not imply or infer default status for any member.

Records

Data records are represented inside the internal caches and in the protocol. The data record contains all of the measures; therefore, it has just a **Path**, not an **MPath**. The **Path** is followed by the array of the measures, each of them in their original data types: 4 bytes for integer measures, 8 bytes for long integer measures, 4 bytes for float measures, 8 bytes for double measures, and 8 bytes for date measures. The format for the representation of measures conforms to the IEEE specifications for the above-mentioned data types.

Client Request Structure Using the (DataSet, Slice) Tuple

The server uses paths to get a very fast response to individual data cells. The path mechanism supported on the server allows that regular, physical, data cells can be accessed by the same mechanism by which formulas are referred to, even though the cells have a different underlying mechanism for data retrieval. This creates a level of abstraction that allows for uniform treatment of formulas and regular data cells. The level abstraction is the **DataSet**, **Slice** tuple that is defined for a specific dimension. It represents all descendants of the same common ancestor.

On each dimension:

- **Slice** specifies the **Path** of a common ancestor. Depending on the ancestor, the path can be to a single data cell or a group of data cells.
- **DataSet** specifies the target level (of the descendants).

The (**DataSet**, **Slice**) is the request format that the server supports. The client side (such as the Pivot Table Service, DR and SONAR) MUST organize client requests for data cells using the (**DataSet**, **Slice**) tuple . **DataSets** and **Slices** exist only for real (physical) cells. They do not exist for formulas.

A tuple (**DataSet**, **Slice**) corresponding to an **MPath** will have the **Slice** equal to **MPath** and a **DataSet** that contains the active levels of each member (from all **DPaths** that form the **MPath**, where the active level is the level of the member inside a **DPath**).

For example, given the cell ([Customers].[All Customers].[Washington].[Redmond].[Nancy Davolio], [Products].[All Products], [Time].[1998].[Q4], the **MPath** is:

(4-1-2, [1-48-2-5]-[1-0-0]-[2-4-0])

The (**DataSet**, **Slice**) for the cell will specify the active level of the target member (corresponds to the **DataSet**) and the **MPath** (corresponds to **Slice**):

([Customers].[All Customers].[Washington].[Redmond].[Nancy Davolio], Descendants([Products].[All Products], [Product].Family]), Descendants([Time].[1998], [Time].[Quarter])) is: (4-2-2, [1-48-2-5]-[1-0-0]-[2-0-0]).

8 Appendix C: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- 2007 Microsoft Office system Service Pack 1 (SP1)
- Microsoft SQL Server 2000
- Microsoft SQL Server 2012

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is 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 the product does not follow the prescription.

<1> Section 2.1: Microsoft supports both layer choices. It is the implementer's choice as to whether to use TCP, HTTP, or HTTPS.

<2> Section 2.2.1.1.11: Microsoft uses an array of bytes in places in the protocol, as documented. The data type of the array of bytes is specific to the place in the protocol in which it is used.

<3> Section 2.2.1.2: Microsoft uses the optional REQDATA element for those protocol objects in which it is defined as required.

<4> Section 2.2.1.2.3: This section defines objects which MAY be utilized within various requests. Microsoft uses those objects that are appropriate to each specific request, as defined in section 2.

<5> Section 2.2.1.6.1.2.1: Microsoft client emits "Light DCube". There is no behavioral difference in Microsoft products, if a different string is emitted by the client.

<6> Section 2.2.1.6.1.2.2: Microsoft products emit the string "Keep-alive". There is no difference in behavior of Microsoft products if a different string is emitted.

<7> Section 2.2.1.6.1.2.3: The Microsoft client uses cookies to establish a session between the client and server.

<8> Section 2.2.1.6.2.1.5.1: Microsoft emits "ASP.NET". There is no difference in behavior of the protocol if a different string is emitted.

<9> Section 2.2.1.6.2.1.5.2: Microsoft clients use a cookie to operate the interaction between client and server within the context of a session. Other clients could choose not to create a session.

<10> Section 3.1.2: In the Microsoft client, the default value is 15 seconds. In the Microsoft client, this default can be specified in the OLE DB connection string or programmatically set prior to connecting if a value other than 15 seconds is desired.

<11> Section 3.1.3: The Microsoft client sets TCP_NODELAY.

<12> Section 3.1.4.3: The Microsoft OLEDB provider includes this request as part of the login sequence. The reason to include this as part of the login sequence is due to high-level application logic, but it is not required by the protocol to precede other requests

<13> Section 3.1.4.4: The Microsoft OLEDB provider includes this request as part of the login sequence. For the Microsoft client, the default database can be specified on the OLE DB provider connection string. The Microsoft client implementation only maintains state for one database, so this request will result in any previous client-side state being discarded.

<14> Section 3.1.4.5: The Microsoft client makes this request, as needed, depending on characteristics of the specific user session.

<15> Section 3.1.4.6: The Microsoft client makes this request, as needed, depending on characteristics of the specific user session.

<16> Section 3.1.4.7: The Microsoft client makes this request, as needed, depending on characteristics of the specific user session.

<17> Section 3.1.4.8: The Microsoft client makes this request, as needed, depending on characteristics of the specific user session.

<18> Section 3.1.4.9: The Microsoft client makes this request, as needed, depending on characteristics of the specific user session.

<19> Section 3.1.4.10: The Microsoft client makes this request, as needed, depending on characteristics of the specific user session.

<20> Section 3.1.4.11: The Microsoft client makes this request, as needed, depending on characteristics of the specific user session. The Microsoft client makes this request in a background ping thread. The Microsoft client's application layer typically converts any current state associated with an obsolete database into a zombie state, per OLE DB specification.

<21> Section 3.1.4.12: The Microsoft client makes this request, as needed, depending on characteristics of the specific user session. The Microsoft client makes this request in a background ping thread. The Microsoft client's application layer typically converts any current state associated with an obsolete cube into a zombie state, per OLE DB specification.

<22> Section 3.1.5.1: The Microsoft client sends requests one at a time and waits for a response to each request before sending another request.

<23> Section 3.1.5.2: The Microsoft client has three separate timeout periods for different types of operations, as described in section 3.2.1.

<24> Section 3.1.5.2: The Microsoft client propagates errors to the higher level application.

<25> Section 3.1.5.4: The Microsoft client follows this request with a Get Database Collection request.

<26> Section 3.1.5.5: The Microsoft client OLE DB provider performs this operation as part of the login sequence. A property of the OLEDB interface allows the database to be specified.

<27> Section 3.1.6.3: The Microsoft OLEDB client uses a ping period, and makes a Refresh Database and Refresh Cube request for all open cubes when the ping period expires. The Microsoft client uses locking to serialize request/response for foreground and background activities over the network connection.

<28> Section 3.1.7.1: The Microsoft OLEDB client propagates network errors to the higher level application. In some cases, the Microsoft OLEDB client attempts to reconnect and to retry the failed operation.

<29> Section 3.2.3: The Microsoft server services requests after the login sequence as they arrive from the client, in the order that they arrive.

<30> Section 3.2.4.3: The sequence of events is dependent upon the requirement of the particular session. Requests are serviced as received.

<31> Section 3.2.4.4: The sequence of events is dependent upon the requirement of the particular session. Requests are serviced as received.

<32> Section 3.2.4.5: The sequence of events is dependent upon the requirement of the particular session. Requests are serviced as received.

<33> Section 3.2.4.6: The sequence of events is dependent upon the requirement of the particular session. Requests are serviced as received.

<34> Section 3.2.4.7: The sequence of events is dependent upon the requirement of the particular session. Requests are serviced as received.

<35> Section 3.2.4.8: The sequence of events is dependent upon the requirement of the particular session. Requests are serviced as received.

<36> Section 3.2.4.9: The sequence of events is dependent upon the requirement of the particular session. Requests are serviced as received.

<37> Section 3.2.4.10: The sequence of events is dependent upon the requirement of the particular session. Requests are serviced as received.

<38> Section 3.2.4.11: The sequence of events is dependent upon the requirement of the particular session. Requests are serviced as received.

<39> Section 3.2.4.12: The sequence of events is dependent upon the requirement of the particular session. Requests are serviced as received.

<40> Section 3.2.7.1: The Microsoft server propagates network errors to the higher level application and closes the network connection.

9 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

10 Index