Sedna Native XML Database Client/Server Protocol

Contents

1 Introduction

This document describes details of the message-based protocol Sedna XML Database server uses for communication with clients through the TCP/IP sockets. Higher level application programming interfaces are built over this protocol.

This document describes versions 1.0, 2.0, 3.0 and 4.0 of the protocol. It consists of three parts: section 2 describes the basic message structure, section 3 defines communication protocol and 4-th section describes the detailed format of each message.

1.1 Known Implementations

The known implementations of the protocol include:

- XQJ Driver by Charles Foster (version 4.0);

- XML:DB API Driver by Charles Foster (version 4.0);

- Java Driver included in distribution (version 2.0);

- Scheme Driver included in distribution (version 2.0);

- Terminal (`se_term`) included in distribution (version 3.0).

2   Message Structure

In messages values of the following three data types are used:

1. `byte` – one byte;

2. `int` – four bytes presented in the network byte order (most significant byte first);

3. `string` – has the following structure: the first byte identifies the string format, the next four bytes (int) specify the length of the string in bytes and the next 'length' bytes is the string. The only supported string format is C-string without trailing null character. The first byte equal zero identifies C-string.

The common message structure is as follows:

- the first four bytes (int) is instruction;
- the next four bytes (int) is the length of a body in bytes;
- the next 'length' bytes is the body.

The body of the message is determined by the instruction. In general the body of all messages is a sequence of values of the three types listed above. The position of a value in the sequence determines its meaning.

In the current version of Sedna the size of the message body must not exceed 10240 bytes.

3 Message Flow

3.1 Start-Up

To begin a session, a client creates a connection to the server and sends a startup message. The server launches a new process that is associated with the session. If launching a new process causes some errors, the server sends the `se_ErrorResponse` message, if not, it sends the `se_SendSessionParameters` message to the client. Then client sends the session parameters in the `se_SessionParameters` message. This message includes the particular protocol version to be used, the name of the user and of the database the user wants to connect to. The server then uses this information to determine whether the connection is acceptable. If not, it sends the error message (`se_ErrorResponse`). If the connection is acceptable the server then sends an authentication request message, to which the client must reply with an appropriate authentication response message. In principle the authentication request/response cycle could require multiple iterations, but the present authentication method uses exactly one request (`se_SendAuthParameters`) and response (`se_AuthenticationParameters`). The authentication cycle ends with the server either rejecting the connection attempt (`se_AuthenticationFailed`), or sending `se_AuthenticationOk`.

The possible instructions from the client in this phase are:

- `se_Start-Up`. Does not contain the body.
- `se_SessionParameters`. The body contains the protocol version, user name and db name.
- `se_AuthenticationParameters`. The body contains password.

The possible instructions from the server in this phase are:

- `se_SendSessionParameters`. Does not contain the body.
- `se_SendAuthParameters`. Does not contain the body.
- `se_AuthenticationOK`. Does not contain the body.
- `se_AuthenticationFailed`. Body contains info.
- `se_ErrorResponse`. Body contains info.

3.2 Transactions

After the start-up phase has succeeded and session is begun, client can run zero or more transactions in the session.

Transactions must be run sequentially, that is the client must commit a transaction before starting a new one.

To begin a transaction client sends the `se_BeginTransaction` message. If the transaction begins Ok, the server answers `se_BeginTransactionOk`. If the transaction fails to begin, the server answers `se_BeginTransactionFailed`.

To commit the transaction client sends the `se_CommitTransaction` message. If the transaction commits Ok, the server answers `se_CommitTransactionOk`. If the transaction fails to commit, the server does rollback for the transaction and answers `se_CommitTransactionFailed`.

To rollback the transaction client sends the `se_RollbackTransaction` message. If the transaction rollbacks Ok, the server answers `se_RollbackTransactionOk`. If the transaction failed to rollback, the server sends `se_RollbackTransactionFailed` and closes the session.

The possible instructions from the client in this phase are:

- `se_BeginTransaction`. Does not contain a body.
- `se_CommitTransaction`. Does not contain a body.
- `se_RollbackTransaction`. Does not contain a body.

The possible instructions from the server in this phase are:

- `se_BeginTransactionOk`. Does not contain a body.
- `se_BeginTransactionFailed`. The body contains the error code and error info.
- `se_CommitTransactionOk`. Does not contain a body.
- `se_CommitTransactionFailed`. The body contains the error code and error info.
- `se_RollbackTransactionOk`. Does not contain a body.
- `se_RollbackTransactionFailed`. The body contains the error code and error info.
- `se_ErrorResponse`. Body contains info.

3.3   Session Options

Since version 3.0 of the Sedna Client-Server protocol it is possible to set session options.

There are a number of session options. Session options can be set at any moment during the session except the period when session is in a query evaluation phase (executing or passing result data to a client).

To set one or more options client must send `se_SetSessionOptions` message. If options were set successfully server sends `se_SetSessionOptionsOk` message to the client. Otherwise server sends `se_ErrorResponse` to the client.

To reset options to their default values client must send `se_ResetSessionOptions`. If options were reset successfully server sends `se_ResetSessionOptionsOk` message to the client. Otherwise server sends `se_ErrorResponse` to the client.

The possible instructions from client are:

- `se_SetSessionOptions`. The body contains any number of pairs: option id followed by option value. Option id is int, option value is string.
- `se_ResetSessionOptions`. Does not contain a body.

The possible instructions from the server are:

- `se_SetSessionOptionsOk`. Does not contain a body.
- `se_ResetSessionOptionsOk`. Does not contain a body.
- `se_ErrorResponse`. Body contains info.

Possible option ids:

- `SEDNA_DEBUG_OFF` – turns off query debug mode. Query debug mode is off be default. (See "Debug features" section of the Sedna Programmer's Guide for details.) This option does not contain a value (there must be a string of zero length in a message body);
- `SEDNA_DEBUG_ON` – turns on query debug mode. This option does not contain a value (there must be a string of zero length in a message body).

3.4   Query Execution

Queries are executed via different subprotocols depending on the type of the query and the query length. There are three types of queries: query, update, bulk load.

If the query is not longer than the message body (10240 bytes) the client sends the Execute message that contains a query statement to the server. If the query is longer than the message body the client must send the query in parts: each part is in the body of `se_ExecuteLong` message. After all parts of the query are sent, client must send `se_LongQueryEnd` message (with the empty body), thus to make the server understand that the whole query is sent.

The server analyzes the query text to identify the type of the query and runs the corresponding subprotocol. The following sections describe these subprotocols.

3.4.1 Querying Data

The client sends a query in the `se_Execute` message, or in the `se_ExecuteLong` messages plus the `se_LongQueryEnd` message. The first byte of the query text is a format of the result to obtain - SXML [1] or XML. Use `0` if you want to get the result of the query in XML, `1` - to get the result in SXML.

Note 1 *Since Protocol Version 4.0 it is client's task to indent and pretty print output.*

The server processes the query. If the query succeeds the server sends the `se_QuerySucceeded` message to the client and then sends a number of messages that contain the first item of the result sequence and query debug information (if any) to the client in the way described below.

When the client needs to get the next item of the result sequence it sends the `se_GetNextItem` message. The server then sends the next item of the result sequence.

The way server sends items depends on the protocol version:

Protocol Versions 1.0, 2.0, 3.0: Server may send every item in parts. Every part of the item is enveloped in the `se_ItemPart` message. When the whole item is sent, the server sends the `se_ItemEnd` message or `se_ResultEnd` message if it was the last item.

Protocol Version 4.0: Server sends items in the following way. `se_ItemStart` message is sent in the first place. It contains type and content of the item being sent. If content is too long to be sent within one message, server may send it in parts. Every part of the item is enveloped in the `se_ItemPart` message. When the whole item is sent, the server sends the `se_ItemEnd` message or `se_ResultEnd` message if it was the last item.

When the result sequence has ended, server on receiving `se_GetNextItem` from client sends the ResultEnd message without prior sending the `se_ItemPart` message.

While sending result data server may also send any number of messages `se_DebugInfo` containing debug information if there is any.

Note 2 *Client debug information is supported in Sedna Client/Server Protocol since version 2.0.*

Client is not required to get all the items of the result sequence. It can send next query for execution before all the items of the result sequence are received from the server.

If the query failed, the server sends message `se_QueryFailed` to the client. The possible instructions from the client in this phase are:

- `se_Execute`. The body contains an XQuery query text.
- `se_ExecuteLong`. The body contains a part of a long XQuery query.
- `se_LongQueryEnd`. Does not contain a body.
- `se_GetNextItem`. Does not contain a body.

The possible instructions from the server in this phase are:

- `se_QuerySucceeded`. Does not contain a body.
- `se_QueryFailed`. The body contains the error code and error info.
- `se_DebugInfo`. The body contains the debug type and debug info. (Since version 2.0)
- `se_ItemStart`. The body contains type of the item being sent and the part of the item. (Since version 4.0)
- `se_ItemPart`. The body contains the part of the item.
- `se_ItemEnd`. Does not contain a body.
- `se_ErrorResponse`. Body contains info.

3.4.2 Updating Data

The client sends the `se_Execute` message (or the `se_ExecuteLong` messages plus the `se_LongQueryEnd` message) that contains an update statement. The server processes the update. If the update succeeded the server sends the `se_UpdateSucceeded` message to the client. If the update failed, the server sends the `se_UpdateFailed` message.

Before sending `se_UpdateSucceeded` or `se_UpdateFailed` message sever may send any number of `se_DebugInfo` messages if there is any debug information.

The possible instructions from the client in this phase are:

- `se_Execute`. The body contains an update statement.
- `se_ExecuteLong`. The body contains a part of a long XQuery query.
- `se_LongQueryEnd`. Does not contain a body.

The possible instructions from the server in this phase are:

- `se_UpdateSucceeded`. Does not contain a body.
- `se_UpdateFailed`. The body contains the error code and error info.
- `se_DebugInfo`. The body contains the debug type and debug info.
- `se_ErrorResponse`. Body contains info.

3.4.3 Bulk Load

The client sends the `se_Execute` message that contains a bulk load statement. The server picks out the name of the file and sends the `se_BulkLoadFileName` message that contains the name.

Since version 2.0 of the Sedna Client-Server protocol server can send multiple `se_BulkLoadFileName` messages if there were multiple file names in a query. This currently can happen in `LOAD MODULE` statement.

The client reads the file specified. If there is no such file or some access errors occur, the client sends `se_BulkLoadError` message to the server. Else the client transfers the data from the file to the server by portions. Each portion is sent in the `se_BulkLoadPortion` message.

When the whole file is sent, the client sends the `se_BulkLoadEnd` message. The server answers with the `se_BulkLoadSucceeded` or `se_BulkLoadFailed` message.

The possible instructions from the client in this phase are:

- `se_Execute`. The body contains a query for bulk load.
- `se_BulkLoadError`. Does not contain the body.
- `se_BulkLoadPortion`. The body contains portion of data.
- `se_BulkLoadEnd`. Does not contain the body.

The possible instructions from the server in this phase are:

- `se_BulkLoadFileName`. The body contains file name.
- `se_BulkLoadSucceeded`. Does not contain the body.
- `se_BulkLoadFailed`. The body contains the error code and error info.
- `se_ErrorResponse`. Body contains info.

3.5 Termination

Termination can be initiated by the client (for example when it closed the session) or by the server (for example in case of an administrator-commanded database shutdown or some failure).

The normal termination procedure is that the client closes the session after transaction commit. In this case the client sends the `se_CloseConnection` message. The server processes its closing procedure. If no errors occur the server sends the `se_CloseConnectionOk` message and closes the connection.

If the client sends the `se_CloseConnection` message before committing the on-going transaction, the server does rollback for the transaction, sends the `se_TransactionRollbackBeforeClose` message and closes the connection.

If on receipt of the `se_CloseConnection` message, some errors on server occur the server sends the `se_ErrorResponse` message and closes.

While an administrator-commanded database shutdown or some failure occurs the server may disconnect without any client request to do so. Before closing the connection the server sends the `se_ErrorResponse` message that contains error code and error info.

The possible instructions from the client in this phase are:

- `se_CloseConnection`. Does not contain the body.

The possible instructions from the server in this phase are:

- `se_CloseConnectionOk`. Does not contain the body.
- `se_TransactionRollbackBeforeClose`. Does not contain the body.
- `se_ErrorResponse`. Body contains info.

3.6   Server Error Handling

In all phases of client/server interaction an error can occur on the server. In this case the server answers to a client request message by sending the `se_ErrorResponse` message that contains the error code and the error info.

- `se_ErrorResponse`. The body contains the error code and the error info.

4   Message Formats

This section describes the detailed format of each message. Each is marked
to indicate that it may be sent either by a client (C), or a server (S).

```
se_Start-Up (C).
  head:
   110 (int)
   body length = 0(int)
  body:
   empty

se_SessionParameters (C).
   head:
120 (int)
body length (int)
  body:
   major protocol version number (byte);
   minor protocol version number (byte);
   user name (string);
   database name (string);

se_AuthenticationParameters (C).
  head:
   130 (int)
   body length (int)
  body:
   password (string)

se_SendSessionParameters (S).
  head:
   140 (int)
   body length = 0 (int)
  body:
   empty

se_SendAuthParameters (S).
  head:
   150 (int)
   body length = 0 (int)
  body:
   empty
```

```
se_AuthenticationOK (S).
  head:
   160 (int)
   body length = 0 (int)
  body:
   empty

se_AuthenticationFailed (S).
  head:
   170 (int)
   body length (int)
  body:
   error code (int)
   error info (string)

se_ErrorResponse (S).
  head:
   100 (int)
   body length (int)
  body:
   error code (int)
   error info (string)

se_BeginTransaction (C).
  head:
   210 (int)
   body length = 0(int)
  body:
   empty

se_CommitTransaction (C).
  head:
   220 (int)
   body length = 0(int)
  body:
   empty

se_RollbackTransaction (C).
  head:
   225 (int)
   body length = 0(int)
  body:
   empty
```

```
se_BeginTransactionOk (S).
  head:
   230 (int)
   body length = 0(int)
  body:
   empty

se_BeginTransactionFailed (S).
  head:
   240 (int)
   body length (int)
  body:
   error code (int)
   error info (string)

se_CommitTransactionOk (S).
  head:
   250 (int)
   body length = 0(int)
  body:
   empty

se_CommitTransactionFailed (S).
  head:
   260 (int)
   body length (int)
  body:
   error code (int)
   error info (string)

se_RollbackTransactionOk (S).
  head:
   255 (int)
   body length = 0(int)
  body:
   empty

se_RollbackTransactionFailed (S).
  head:
   265 (int)
   body length (int)
  body:
```

```
   error code (int)
   error info (string)

se_Execute (C).
  head:
   300 (int)
   body length (int)
  body:
   result format (byte) + query text (string)

se_ExecuteLong (C).
  head:
   301 (int)
   body length (int)
  body:
   result format (byte) + query text (string)

se_LongQueryEnd (C).
  head:
   302 (int)
   body length = 0 (int)
  body:
   empty

se_GetNextItem (C).
  head:
   310 (int)
   body length = 0 (int)
  body:
   empty

se_QuerySucceeded (S).
  head:
   320 (int)
   body length = 0(int)
  body:
   empty

se_DebugInfo (S).
  head:
   325 (int)
   body length (int)
```

```
  body:
   debug type (int)
   debug info (string)

se_QueryFailed (S).
  head:
   330 (int)
   body length (int)
  body:
   error code (int)
   error info (string)

se_UpdateSucceeded (S).
  head:
   340 (int)
   body length = 0(int)
  body:
   empty

se_UpdateFailed (S).
  head:
   350 (int)
   body length (int)
  body:
   error code (int)
   error info (string)

se_ItemStart (S).
  head:
   355 (int)
   body length (int)
  body:
   item class (byte) - see below possible values of this field
   item type (byte)  - see below possible values of this field
   URL flag (byte)   - either 0 or 1, determines if URL field is empty
   [URL (string)]    - optional, URL of the item
   result part (string)

Item class enumeration (see se_ItemStart message )is defined as follows:
enum se_item_class  {
    se_atomic    = 1,   //item type defines atomic type
    se_document  = 2,
```

```
    se_element  = 3,   //item type defines atomic type
    se_attribute = 4,  //item type defines atomic type
    se_namespace = 5,
    se_pi       = 6,
    se_comment  = 7,
    se_text     = 8
};

Item type enumeration (see se_ItemStart message )is defined as follows:
enum se_item_type {
    /* Abstract base types */
    se_anyType            = 0,
    se_anySimpleType      = 1,
    se_anyAtomicType      = 2,
    /* Built-in simple, non-atomic types */
    se_IDREFS             = 3,
    se_NMTOKENS           = 4,
    se_ENTITIES           = 5,
    /* Built-in complex types */
    se_untyped            = 6,
    /* Built-in atomic types (Primitive types) */
    se_dateTime           = 10,
    se_date               = 11,
    se_time               = 12,
    se_duration           = 13,
    se_yearMonthDuration  = 14,
    se_dayTimeDuration    = 15,
    se_gYearMonth         = 16,
    se_gYear              = 17,
    se_gMonthDay          = 18,
    se_gDay               = 19,
    se_gMonth             = 20,
    se_float              = 21,
    se_double             = 22,
    se_decimal            = 23,
    se_integer            = 24,
    se_boolean            = 25,
    se_untypedAtomic      = 26,
    se_string             = 27,
    se_base64Binary       = 28,
    se_hexBinary          = 29,
```

```
    se_anyURI              = 30,
    se_QName               = 31,
    se_NOTATION            = 32,
    /* Types derived from xs:string */
    se_normalizedString    = 41,
    se_token               = 42,
    se_language            = 43,
    se_NMTOKEN             = 44,
    se_Name                = 45,
    se_NCName              = 46,
    se_ID                  = 47,
    se_IDREF               = 48,
    se_ENTITY              = 49,
    /* Types derived from xs:integer */
    se_nonPositiveInteger = 50,
    se_negativeInteger     = 51,
    se_long                = 52,
    se_int                 = 53,
    se_short               = 54,
    se_byte                = 55,
    se_nonNegativeInteger = 56,
    se_unsignedLong        = 57,
    se_unsignedInt         = 58,
    se_unsignedShort       = 59,
    se_unsignedByte        = 60,
    se_positiveInteger     = 61
};

se_ItemPart (S).
  head:
   360 (int)
   body length (int)
  body:
   result part (string)

se_ItemEnd (S).
  head:
   370 (int)
   body length = 0(int)
  body:
   empty
```

```
se_ResultEnd (S).
  head:
   375 (int)
   body length = 0(int)
  body:
   empty

se_BulkLoadError (C).
  head:
   400 (int)
   body length (int)
  body:
   error code (int)
   error info (string)

se_BulkLoadPortion (C).
  head:
   410 (int)
   body length (int)
  body:
   data portion (string)

se_BulkLoadEnd (C).
  head:
   420 (int)
   body length = 0 (int)
  body:
   empty

se_BulkLoadFileName (S).
  head:
   430 (int)
   body length (int)
  body:
   file name (string)

se_BulkLoadFromStream (S).
  head:
   431 (int)
   body length = 0(int)
  body:
   empty
```

```
se_BulkLoadSucceeded (S).
  head:
   440 (int)
   body length = 0(int)
  body:
   empty

se_BulkLoadFailed (S).
  head:
   450 (int)
   body length(int)
  body:
   error code (int)
   error info (string)

se_ShowTime (C).
  head:
   451 (int)
   body length = 0(int)
  body:
   empty

se_LastQueryTime (S).
  head:
   452 (int)
   body length(int)
  body:
   time (string)

se_CloseConnection (C).
  head:
   500 (int)
   body length = 0(int)
  body:
   empty

se_CloseConnectionOk (S).
  head:
   510 (int)
   body length = 0(int)
  body:
   empty
```

```
se_TransactionRollbackBeforeClose (S).
  head:
   520 (int)
   body length = 0(int)
  body:
   empty

se_SetSessionOptions (C).
  head:
   530 (int)
   body length (int)
  body:
   any number of pairs: option id (int), option value (string)

se_SetSessionOptionsOk (S).
  head:
   540 (int)
   body length = 0(int)
  body:
   empty

se_ResetSessionOptions (C).
  head:
   550 (int)
   body length = 0(int)
  body:
   empty

se_ResetSessionOptionsOk (S).
  head:
   560 (int)
   body length = 0(int)
  body:
   empty
```

References

[1] Oleg Kiselyov. "SXML Specification, Revision 3.0", http://www.okmij.org/ftp/Scheme/SXML.html