

Sedna Administration Guide

Contents

1	Introduction	3
2	Administration via Command Line	4
2.1	Sedna Directory Structure	4
2.2	Managing Sedna	4
2.2.1	Running Sedna	4
2.2.2	Stopping Sedna	6
2.2.3	Configuring Sedna	7
2.3	Managing Databases	8
2.3.1	Creating a Database	8
2.3.2	Configuring a Database	11
2.3.3	Deleting a Database	12
2.3.4	Running a Database	12
2.3.5	Stopping a Database	13
2.4	Sedna Terminal	14
2.5	Backup and Restore	16
2.5.1	Export/Import Utility	17
2.5.2	File system level backup	19
2.5.3	Hot Backup	19
2.5.4	Migration Between Releases	22
2.6	Event Log	23
2.6.1	Files	23
2.6.2	Severity Level Configuration	23
2.7	The Examples of Using Command Line Utilities	24
3	User-Based Authorization System	25
3.1	Database Users	25
3.2	Privileges and Roles	27
3.3	Granting privileges	28
3.4	Revoking privileges	29

3.5 Client Authentication	30
4 Localization	30

1 Introduction

This guide describes administration of the Sedna XML Database Management System (Sedna for short). Sedna has the following components.

Governor serves as a “control center” of the system. All other components register at the Governor. The Governor knows which other components (e.g. databases and sessions) are running in the system and controls them. Other components cannot function properly if the Governor is not running so you must start it to use Sedna. Besides, Governor is responsible for handling remote client’s requests. For each request it creates a session which keeps the direct connection with client. Then the client interacts with the system via session.

Session keeps the client’s settings and allows client to run a sequence of transactions. The transactions within a session are strictly serial, that is one transaction ends before next one starts, there is only one active transaction per session. Transaction provides functionality required for execution of the client’s queries. The query execution consists of several steps: (1) parsing of the query and translation of the query into its logical representation, (2) processing of the logical representation by optimizer that produces the optimal query execution plan, (3) execution of the optimal query plan and passing results to the client. Transaction is rolled back if error is occurred during its execution.

Storage manager manages a database and provides memory management functionality to the executor. There is an instance of storage manager for each database run.

All the components described above are implemented as operating system processes.

One can run sessions via Sedna client applications (See Sedna Programmer’s Guide how to use Sedna APIs) or via Sedna interactive terminal (see [2.4](#)). In both cases, first of all the Governor and storage managers (one for each databases required) components should be run.

There are a set of command line utilities that allow you to administrate all aspects of Sedna. These utilities are described in detail in the following section.

2 Administration via Command Line

2.1 Sedna Directory Structure

The Sedna directory hierarchy is fundamental to obtaining an overall understanding of the system. Below `SEDNA_INSTALL` refers to the directory where Sedna is installed. The directory structures are identical on all supported platforms.

<code>SEDNA_INSTALL/bin</code>	command line utilities for managing Sedna and databases, running queries, etc.
<code>SEDNA_INSTALL/cfg</code>	database configuration files named <code><db_name>_cfg.xml</code> , where <code><db_name></code> is the name of the corresponding database
<code>SEDNA_INSTALL/data</code>	database data stored in subdirectories named <code><db_name>_files</code> , where <code><db_name></code> is the name of the corresponding database
<code>SEDNA_INSTALL/lib</code>	database-independent libraries of external functions
<code>SEDNA_INSTALL/data/<db_name>_files/lib</code>	database-specific libraries of external functions
<code>SEDNA_INSTALL/doc</code>	Sedna documentation
<code>SEDNA_INSTALL/driver</code>	API drivers for various programming languages
<code>SEDNA_INSTALL/include</code>	include files
<code>SEDNA_INSTALL/etc</code>	Sedna configure sample file <code>sednaconf.xml.sample</code>
<code>SEDNA_INSTALL/examples</code>	example databases and applications for the Sedna APIs
<code>SEDNA_INSTALL/share</code>	Sedna database metadata files (currently only database users and privileges metadata)

For the installation instructions see the `INSTALL` file shipped with the Sedna distribution.

2.2 Managing Sedna

2.2.1 Running Sedna

To start the Sedna server one should start *Governor* by executing the `se_gov` command. This starts the main server component. After that Administrator can run databases (as discussed in Section [2.3.4](#)).

Important note: since version 3.5 Sedna server by default listens on 'localhost' and allows only local clients. If you want to work with Sedna remotely make sure that `-listen-address` parameter value is adjusted properly.

The usage of the `se_gov` command is as follows:

Usage: `se_gov [options]`

options:

<code>--help</code>	display this help and exit
<code>-help</code>	display this help and exit
<code>-version</code>	display product version and exit
<code>-background-mode on/off</code>	start the server in the background mode (default on)
<code>-listen-address address</code>	local address Sedna listens for client connections (default localhost)
<code>-port-number <int></code>	socket listening port (default 5050)
<code>-el-level level</code>	event logging level (default 3): 0 - event logging is off 1 - log only fatal errors 2 - log all errors/warnings 3 - system operational messages 4 - log everything (+debug messages)
<code>-alive-timeout timeout</code>	session keep alive timeout (default 0 - infinite timeout)
<code>-stack-depth depth</code>	maximum executor stack depth (default 4000)

`--help` and `-help` options provide the exhaustive information about the `se_gov` command.

`-version` option allows getting the version of the *Governor*.

`-background-mode` option allows Administrator to start *Governor* in the background mode.

`-listen-address` specifies the address on which the server is to listen for connections from client applications. The value takes the form of a numeric IP address or host name (it must be resolvable by your system, check your host file). The value `0.0.0.0` corresponds to all available IP interfaces. The default value is `localhost`, which allows only local "loopback" connections to be made. If value is not specified Sedna server tries to find the `sednaconf.xml` configuration file and uses the address specified as `listen_address` parameter (see section 2.2.3, Configuring Sedna). If there is no configuration file or `listen_address` parameter is not specified Sedna would listen on `localhost`.

Note, if you want to make Sedna work in the old-fashioned (prior version 3.5) way you should set `-listen-address` option to `0.0.0.0` — that would force Sedna

to listen on every interface in your system as it worked in older versions.

-port-number option specifies the port number on which Sedna listens for connections from clients. If port number is not specified Governor tries to find the `sednaconf.xml` file and use port number specified as `listener_port` parameter (see section 2.2.3, Configuring Sedna). If it still cannot find port number it will use default 5050 value.

-el-level option specifies the event log severity level. For details refer to the section 2.6, Event Log.

-alive-timeout option specifies number of seconds to wait for the next request from the some client on the same connection. Session keep alive timeout can be defined also in the `sednaconf.xml` file. Value provided via command line overrides value (if any) defined in the `sednaconf.xml` file.

-stack-depth option specifies the maximum safe depth of the session's physical operations stack. Setting the parameter higher than the actual process stack may fit will mean that a runaway recursive function can crash an individual session (and server in the current architecture). The default setting is 4000, which is unlikely to risk crashes. However, it may be too small to allow execution of complex functions. Value provided via command line overrides value (if any) defined in the `sednaconf.xml` file.

To get the information about which Sedna components are run use `se_rc` command.

The usage of the `se_rc` command is as follows:

Usage: `se_rc [options]`

options:

<code>--help</code>	display this help and exit
<code>-help</code>	display this help and exit
<code>-version</code>	display product version and exit
<code>-sm-list</code>	display running databases list

2.2.2 Stopping Sedna

To stop Sedna, Administrator should use the `se_stop` command. This command stops all Sedna components including all databases, which are run. Command has several optional parameters.

The usage of the `se_stop` command is as follows:

Usage: `se_stop [options]`

options:

<code>--help</code>	display this help and exit
<code>-help</code>	display this help and exit
<code>-version</code>	display product version and exit
<code>-hard</code>	attempt to roll back transactions immediately

The first and second options are straightforward, typing these options Administrator can get the exhaustive information about the `se_stop` command. Option `-version` allows getting the version of the `se_stop`.

Note, `se_stop` may cause a rollback of some active transactions in the system. However, Sedna waits for the last statement in each transaction is completed. Option `-hard` causes Sedna to attempt to roll back transactions as soon as possible (except bulk load and update transactions).

2.2.3 Configuring Sedna

Sedna configuration parameters are stored in the file `sednaconf.xml` located in:

```
[win:] SEDNA_INSTALL\etc
[nix:] SEDNA_INSTALL/etc
```

The file is optional. If there is no such file, the default values are used. The file is an XML document, which satisfies the following DTD:

```
<!ELEMENT sednaconf (sedna_data, os_primitives_id_min_bound,
                      listener_port, ping_port,
                      event_log_level, keep_alive_timeout,
                      session_stack_depth)>
<!ELEMENT sedna_data (#PCDATA)>
<!ELEMENT os_primitives_id_min_bound (#PCDATA)>
<!ELEMENT listen_address (#PCDATA)>
<!ELEMENT listener_port (#PCDATA)>
<!ELEMENT ping_port (#PCDATA)>
<!ELEMENT event_log_level (#PCDATA)>
<!ELEMENT keep_alive_timeout (#PCDATA)>
<!ELEMENT session_stack_depth (#PCDATA)>
```

The `sedna_data` element contains the path to the directory where database configuration files (in the `cfg` subdirectory) and databases (in the `data` subdirectory) are stored. Path must be absolute and exist. The path by default is `SEDNA_INSTALL`.

The `os_primitives_id_min_bound` element defines the minimal value for identifiers of OS resources (e.g. semaphores) used in Sedna. In order to run several Sedna instances on one machine Administrator should configure this parameter in a such way that identifiers belonging to different Sedna instances does not intersect. In a standard configuration one Sedna instance acquires a range of identifiers which length is equal to 150.

The `listen_address` element specifies the address on which the server is to listen for connections from client applications. The value takes the form of a host name or numeric IP address. The value `0.0.0.0` corresponds to all available IP

interfaces. The default value is `localhost`, which allows only local "loopback" connections to be made.

The `listener_port` element defines the port number for listening for connection requests from a client.

The `ping_port` element defines the port number of Sedna ping server that is an internal kernel mechanism. It is used to stop all Sedna processes in case of crash.

The `event_log_level` defines event log severity level (0-4). For full description see section 2.6, Event Log.

The `keep_alive_timeout` defines session keep alive timeout, i.e. number of seconds to wait for the next request from the some client on the same connection.

The `session_stack_depth` defines the maximum safe depth of the session's physical operations stack. Setting the parameter higher than the actual process stack may fit will mean that a runaway recursive function can crash an individual session (and server in the current architecture). The default setting is 4000, which is unlikely to risk crashes. However, it may be too small to allow execution of complex functions.

To tune the Sedna configuration, Administrator should edit the proper elements of the Sedna configuration file. Note, in order for new configuration to take effect you must restart Sedna server (`se_gov`).

2.3 Managing Databases

2.3.1 Creating a Database

To create a database, use the `se_cdb` command.

Each database consists of the following files, which are located as described in Section 2.1.

- The `<db_name>.sedata` file is used for storing persistent XML data, which are loaded to the database by user using `LOAD/UPDATE` expressions.
- The `<db_name>.setmp` file is used for storing temporary XML data. It stores intermediate results and size may grow during query execution.
- The `<db_name>.*llog` files are used for storing database logical log.

Besides, for each database there are run-time configuration parameters, which are stored in the database configuration file (see Section 2.3.2).

The usage of the `se_cdb` command is as follows:

Usage: `se_cdb [options] dbname`

options:

`--help` display this help and exit

<code>-help</code>	display this help and exit
<code>-version</code>	display product version and exit
<code>-data-file-max-size Mbs</code>	the max size of data file (in Mb), infinite size by default (0)
<code>-tmp-file-max-size Mbs</code>	the max size of tmp file (in Mb), infinite size by default (0)
<code>-data-file-ext-portion Mbs</code>	the data file extending portion size (in Mb), default 100Mb
<code>-tmp-file-ext-portion Mbs</code>	the tmp file extending portion size (in Mb), default 100Mb
<code>-data-file-init-size Mbs</code>	the data file initial size (in Mb), default 100Mb
<code>-tmp-file-init-size Mbs</code>	the tmp file initial size (in Mb), default 100Mb
<code>-bufs-num N</code>	the number of buffers in main memory, default 1600 (64Kb per buffer)
<code>-upd-crt N</code>	update criterion parameter (fraction of database), default 0.25
<code>-max-log-files N</code>	maximum log files until log truncate (default: 3)
<code>-log-file-size Mbs</code>	maximum one log file size (in Mb), (default 100Mb)
<code>-db-security level</code>	the level of database security: 1) 'off' - none; 2) 'authentication' (default); 3) 'authorization'
<code>db_name</code>	name of the database to be created

The first and second options are straightforward, typing one of these options administrator can get the exhaustive information about the `se_cdb` command.

The third option allows getting the version of the `se_cdb` command.

The `-data-file-max-size` option allows Administrator to set up the maximum size (in Mb) of the file where the persistent data are stored. The default value for this option is infinite (0). Note, when database hits this limit it raises system error (SE1012) and shutdowns.

The `-tmp-file-max-size` option allows Administrator to set up the maximum size (in Mb) of the file where the temporary data are stored. The default value for this option is infinite (0). Note, when database hits this limit it raises system error (SE1012) and shutdowns.

The `-data-file-ext-portion` option allows Administrator to set up the size (in Mb) of the portion to which the persistent data file will be extended as result of executing resize operation. Resize operation is applied to the persistent data

file if its size is less than needed for storing XML documents. The default value is 100 Mb.

The **-tmp-file-ext-portion** option allows Administrator to set up the size of the portion to which the temporary data file will be extended as result of executing resize operation. The resize operation is applied to the temporary data file if its size is less than needed for storing temporary data intermediate results of the query, for example). The default value is 100 Mb.

The **-data-file-init-size** option allows Administrator to set up the initial size (in Mb) of the persistent data file. It is obvious that the initial size can't be greater than the maximum size of the persistent data file. The default value is 100 Mb.

The **-tmp-file-init-size** option allows Administrator to set up the initial size (in Mb) of the temporary data file. It is obvious that the initial size can't be greater than the maximum size of the temporary data file. The default value is 100 Mb.

The **-bufs-num** option allows Administrator to set up the number of buffers in main memory. This number is connected with the size of the main memory, but the performance of the database depends of this number very much. The default value is 1600 buffers, which is equal to 100Mb and we do not recommend to use less number of buffers. The size of one buffer is 64Kb and can't be customized by Administrator.

The **-upd-crt** option is the parameter of the Sedna Storage Manager to control snapshot advancement rate. It allows Administrator to specify the fraction of the database updating of which will result in snapshot advancement. The default value is 0.25.

The **-max-log-files** option allows Administrator to control how many log files will be created until Storage Manager will try to truncate logical log by making a checkpoint. For example, if the value is 3, then after creating fourth file Storage Manager will try to truncate log. This may significantly speed up recovery process, since there would be much shorter log to analyze. Note, however, that checkpoint may be quite a performance-heavy procedure. So you should not set the value too small. This parameter can be changed later on every Storage Manager run.

The **-log-file-size** option allows to control size of each log file. When current log file size becomes greater than the value specified, new log file is created. With the previous parameter it allows Administrator to find compromise between frequent checkpoints and faster recovery. This parameter can be set only on database creation and cannot be changed later.

The **-db-security** option is used to set the level of security in a database being created. Currently there are 3 possible levels: (1) **off** - there is no authentication, neither authorization; (2) **authentication** - only authentication supported (checks user name and password on session open) (3) **authorization** - authentication and authorization supported (support for database users and privileges). By default database is created with **authentication** level. Note, that this parameter

is set only once on database creation and cannot be changed later for the database.

The last parameter `db_name` is the name of the database to be created. Database name may contain Latin letters, numbers. Also the following special symbols are allowed `#%&()[]{},.-_=@^'~` (double quote (quote on Windows) database name parameter to create a database with the name which contains special symbols). There cannot be two databases with the same name.

2.3.2 Configuring a Database

For each database there is a set of run-time configuration parameters, which store in the database configuration file. The name of the database configuration file is `<db_name>.cfg.xml` where `<db_name>` is the name of the database. The location of configuration file is:

```
[win:] SEDNA_INSTALL\cfg
[nix:] SEDNA_INSTALL/cfg
```

The configuration file is an XML document, which satisfies the following DTD:

```
<!ELEMENT db (name, bufs_num, max_trs_num, upd_crt,
               tmp_file_initial_size)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT bufs_num (#PCDATA)>
<!ELEMENT max_trs_num (#PCDATA)>
<!ELEMENT max_log_files (#PCDATA)>
<!ELEMENT upd_crt (#PCDATA)>
<!ELEMENT tmp_file_initial_size (#PCDATA)>
```

The root of this XML document is the `db` element, which contains `name`, `bufs_num`, `max_trs_num`, `max_log_files`, `upd_crt`, `tmp_file_initial_size` elements. The `name` element defines the name of the database. The `bufs_num` element defines the number of buffers in main memory for Storage Manager component. The semantic of `max_trs_num` element will be defined in future versions of Sedna. The `max_log_files` element defines the maximum number of log files prior to truncation attempt. The `upd_crt` element defines the fraction of the database updating of which will result in snapshot advancement. Finally, `tmp-file-initial-size` defines initial size of the temporary data file in megabytes. Once the database is started up, old temporary file is removed and empty one of this size is initialized. Default value is 100 Mb. To tune the database run-time configuration, Administrator should edit the proper elements of the database configuration file. Note, in order for new configuration to take effect you must restart Storage Manager.

2.3.3 Deleting a Database

To drop a database, use the `se_ddb` command. The usage of the `se_ddb` command is as follows:

Usage: `se_ddb [options] dbname`

options:

<code>--help</code>	display this help and exit
<code>-help</code>	display this help and exit
<code>-version</code>	display product version and exit
<code>db-name</code>	The name of the database

The first and second options are straightforward, typing one of these options Administrator can get the exhaustive information about the `se_ddb` command.

The third option allows getting the version of the `se_ddb` command.

The last parameter `db_name` is the name of the database to be dropped.

Note that administrator must stop database firstly (see [2.3.5](#)) and then run `se_ddb` utility.

2.3.4 Running a Database

To run the database with name `db_name` Administrator should use `se_sm` command. The run-time configuration parameters of the database can be set by means of `se_sm` command line options. If configuration parameter is not set via command line option, than the database Storage Manager retrieves the default value of this parameter from configuration file (see section [2.3.2](#)).

The usage of the `se_sm` command is as follows:

Usage: `se_sm [options] dbname`

options:

<code>-help</code>	display this help and exit
<code>--help</code>	display this help and exit
<code>-version</code>	display product version and exit
<code>-background-mode on/off</code>	start the server in the background mode (default on)
<code>-bufs-num N</code>	the number of buffers in main memory, (config file defines default value)
<code>-upd-crt N</code>	criterion parameter to advance snapshots (config file defines default value)
<code>-max-log-files N</code>	maximum log files until log truncate (default: 3)
<code>-tmp-file-init-size Mbs</code>	the tmp file initial size (in Mb), (config file defines default value)
<code>db-name</code>	The name of the database

The first and second options are straightforward, typing this option Administrator can get the exhaustive information about the `se_sm` command.

The third option allows getting the version of the `se_sm` command. The `-background-mode` option allows Administrator to start database in the background mode.

The `-bufs-num` option allows to set up the number of buffers in main memory. This number is connected with the size of the main memory, but the performance of the database depends of this number very much. The default value is retrieved from configuration file. The size of one buffer is 64Kb and can't be customized.

The `-upd-crt` option allows Administrator to specify the fraction of the database updating of which will result in snapshot advancement.

The `-max-log-files` option allows Administrator to control how many log files will be created until Storage Manager will try to truncate logical log by making a checkpoint. For example, if the value is 3, then after creating fourth file Storage Manager will try to truncate log. This may significantly speed up recovery process, since there would be much shorter log to analyze. Note, however, that checkpoint may be quite a performance-heavy procedure. So you should not set the value too small. This parameter can be changed on every Storage Manager run.

The `-tmp-file-init-size` defines initial size of the temporary data file in megabytes. Once the database is started up, old temporary file is removed and empty one of this size is initialized. Default value of this parameter is retrieved from the database configuration file [2.3.2](#).

The last parameter `dbname` defines the name of the database to be started up.

Note: before running any database be sure that Sedna Server is started up (see section [2.2.1](#)).

2.3.5 Stopping a Database

To stop the database with name `db_name`, Administrator should use `se_smsd` command. This command takes one input parameter, which defines the name of the database to be shut down. The usage of the `se_smsd` command is as follows:

Usage: `se_smsd [options] dbname`

options:

<code>--help</code>	display this help and exit
<code>-help</code>	display this help and exit
<code>-version</code>	display product version and exit
<code>db-name</code>	The name of the database

The first and second options are straightforward, typing these options Administrator can get the exhaustive information about the `se_smsd` command. The third option allows getting the version of the `se_smsd`.

Note that `se_smsd` command causes a roll back of all active transactions executed over `db_name` database.

2.4 Sedna Terminal

`se_term` is an interactive terminal to Sedna. It is a C application that uses Sedna C API (see “Sedna Programmer’s Guide”) to work with Sedna. It enables you to open a session to one of the Sedna database, type in queries interactively, issue them to the database and see the query results. Alternatively, input can be from a file, or a single query can be passed for execution as a command line parameter. In addition, it provides a number of meta-commands.

The usage of the `se_term` is as follows:

Usage: `se_term [options] dbname`

options:

<code>-help</code>	display this help and exit
<code>--help</code>	display this help and exit
<code>-version</code>	display product version and exit
<code>-file filename</code>	file with an XQuery query
<code>-output filename</code>	output file (default stdout)
<code>-query "query"</code>	XQuery query to execute
<code>-echo on/off</code>	display <code>se_term</code> output (default: on for interactive mode, off for batch mode)
<code>-show-time on/off</code>	show time of the latest query execution (default off)
<code>-debug on/off</code>	execute statements in debug mode (default off)
<code>-host host</code>	hostname of the machine with Sedna running (default localhost)
<code>-port-number port</code>	socket listening port (default 5050)
<code>-name name</code>	user name
<code>-pswd password</code>	user password
<code>db-name</code>	database name

`-help` or `--help` provides the exhaustive information about the `se_term`.

`-version` option allows getting the version of the `se_term`.

`-file` option specifies the name of the file that contains any number of queries and meta-commands. `se_term` uses this file as the source of queries and meta-commands instead of reading them interactively. `se_term` executes the queries and meta-commands from file consequently and exit. XQuery and XUpdate statements must be delimited with ampersand symbol at the end.

For example, the following script turns off autocommit mode, then executes a number of XQuery statements and finally commits transaction explicitly:

```

\nac
CREATE DOCUMENT "test"&
UPDATE INSERT <test>{"test"}</test> INTO fn:doc("test")&
fn:doc("test")&
\commit

```

-**output** option specifies a filename of a file to redirect all **se_term** output to.

-**query** option specifies the query to execute.

-**echo** option specifies if **se_term** output needs to be displayed or not. If the **echo** is on, the output is displayed, if the **echo** is off, the output is not displayed. By default, the **echo** option is set to on when using **se_term** in an interactive mode, and the **echo** option is set to off when using **se_term** in a batch mode (running queries/commands from file).

-**show-time** option allows getting the time of the latest query execution. If used with the **-query** option provides the time of the specified query execution; if used with the **-file** option provides the time of the execution of the last query in file.

-**debug** option specifies session debug mode. If **-debug** option is on, statements of this session are executed in a debug mode. If **-debug** option is off, statement of this session are executed in normal (not debug) mode. For details see "Debug Facilities" section of the Sedna Programmer's Guide.

-**host** option specifies the name of the machine with Sedna DBMS running. If not used, default value **localhost** is used.

-**port** option specifies the TCP port on which Sedna server is listening to client applications. If omitted, port number 5050 is used by default.

-**name** and **-password** options specify the user name and the password. If omitted, connects as a predefined user **SYSTEM** with password **MANAGER**.

dbname is a required argument. It specifies the name of the database to connect to.

When **se_term** is used in the interactive mode it takes in query/update statements and meta-commands from stdin. To execute a query/update statement type in a statement and use ampersand and line feed to terminate it (**&'\n'** terminates the query). An end of line does not terminate the query, thus queries can be spread over several lines for clarity. Alternatively, **se_term** can be used in a batch mode, then query/update statements and meta-commands are taken from the input file specified by the **-file** option. By default, **se_term** session is run in the *autocommit mode*, that is, each statement is run in a separate transaction. To switch to a *manual-commit mode* use **unset** meta-command (**se_term** meta-commands are described below in this section). If the statement executed successfully, the results are displayed on the screen.

se_term returns 0 to the shell if it finished normally, 1 if a fatal error of its own (out of memory, file not found) occurs, 2 if the connection to the Sedna server went bad and the session is not interactive, and 3 if a statement or a command failed

and the variable `ON_ERROR_STOP` was set (`se_term` internal variables are described below in this section).

Meta-commands are commands for `se_term` that processed by the `se_term` itself. A meta-command begins with a backslash and that differs it from the query/update statements. Thus, the format of `se_term` meta-command is the backslash, followed immediately by a command, with no ampersand at the end. `se_term` takes in the following meta-commands:

```
\? - for help on internal slash commands
\commit - to commit transaction
\rollback - to rollback transaction
\showtime - to show the time of the latest query execution
\set - to set the terminal internal variable
\unset - to unset the terminal internal variable
\quit, \q - to close session and quit the Sedna Terminal
```

`set` and `unset` meta-commands are used for managing `se_term` internal variables. Notice, that there are aliases for some commands. So you can write `\ac` instead of `\set AUTOCOMMIT`. There are following `se_term` internal variables:

`AUTOCOMMIT` (`\ac` for `set`, `\nac` for `unset`) - when set, autocommit mode is on. When unset manual-commit mode is on. `AUTOCOMMIT` is set by default.

`ON_ERROR_STOP` - when set, `se_term` returns with the code 3 when statement or meta-command fails. When unset `se_term` processing continues, unless it is the connection failure.

`DEBUG` - when set, session debug mode is on. When unset, session debug mode is off. See "Debug Facilities" section of the Sedna Programmer's Guide for details.

`TRANSACTION_READ_ONLY` (`\ro` for `set`, `\upd` for `unset`) - transactions are run as `READ-ONLY` when set. When unset, transactions are run as `UPDATE`-transactions. By default transactions are run as `UPDATE` - transactions.

`LOG_LESS_MODE` (`\ll` for `set`, `\fl` for `unset`) - when set, every following bulkload will be less logged and checkpoint will be made on every commit. When unset, every following bulkload will be fully logged. By default transactions are run in full log mode. This option should be used with care (see details about `SEDNA_LOG_AMOUNT` connection attribute in "Sedna Programmer's guide").

`QUERY_TIMEOUT=<time in seconds>` - when set, every query execution will be dropped on server if it lasts longer than timeout set. By default there is no any timeout set (query is executed as long as needed).

`set?` - provides help on `se_term` internal variables.

2.5 Backup and Restore

As any database system that contains valuable data, Sedna databases should be backed up regularly. In this section we present different approaches to back up Sedna data and the process of migration between different Sedna releases.

2.5.1 Export/Import Utility

The purpose of `se_exp` utility is to provide functionality of exporting/importing data. The idea behind the `se_exp` method is to generate a set of XML files and XQuery scripts to restore the database in the same state as it was at the time of the exporting.

Important note: current version of the `se_exp` utility doesn't support exporting/importing triggers, documents with multiple roots and empty documents (empty documents nodes and document nodes with multiple root elements are allowed by XQuery data model but cannot be serialized as is without modifications).

Note that `se_exp` is a regular Sedna client application. This means that you can export data from any remote host that has access to Sedna. The requirement is that `se_exp` should operate with special permissions. In particular, it must have read access to all documents in the database including system metadata. This means that you should run it as a user which has DBA role.

The usage of `se_exp` utility is:

```
se_exp [options] command dbname path
```

options:

<code>-help</code>	display this help and exit
<code>--help</code>	display this help and exit
<code>-version</code>	display product version and exit
<code>-verbose on/off</code>	verbose output (default off)
<code>-host host</code>	hostname of the machine with Sedna running (default localhost)
<code>-port-number port</code>	socket listening port (default 5050)
<code>-name name</code>	user name
<code>-pswd password</code>	user password
<code>command</code>	export restore import
<code>db-name</code>	database name
<code>path</code>	path to exported/imported data

There are three commands to manipulate data with `se_exp`. They are `export`, `restore` and `import`. Below we describe each of these commands in details.

Export

The purpose of `export` command is to export data from specified database. The basic usage of this command is:

```
se_exp export dbname path
```

The parameter `dbname` specifies the database in Sedna to export data from. The `path` parameter specifies the directory to store files with exported data.

Note 1 *If the directory to which **path** refers contains files that have the same names as the files created by **se_exp** they will be replaced.*

While export process **se_exp** generates a set of XML files and a set of XQuery scripts to recreate the state of database. For each XML document in the database including XML documents in collections **se_exp** generates an XML file. Some XML files with system metadata are also generated. Note that the security metadata is exported in insecure way (the file contains unencrypted user names and passwords).

Exported data created by **se_exp** is transaction consistent, that is, updates to the database while **se_exp** is running will not be in the exported data.

To specify which database server **se_exp** should contact, use the command line options **-host host**. The default host is the local host. As Sedna client application, **se_exp** requires user name and password to connect to the database. You can either specify them with **-name** and **-pswd** options or type user name and password in the dialog while running **se_exp**.

Restore

The **restore** command restores data created by the export command into the empty database. The restore command is intended for migration between different releases of Sedna and for back up of your data in XML format. The basic usage of this command is:

```
se_exp restore dbname path
```

The parameter **dbname** specifies the database in Sedna to restore data into. The **path** parameter specifies the directory with data to restore.

The database **dbname** will not be created by this command. It is required that the target database already exist and run before starting the restore process. You must create it yourself with the help of **se_cdb** command and start it with **se_sm** command. It is also required that the target database is empty, i.e. it doesn't contain any data or any users or roles except the default one.

Import

The **import** command imports data created by the export command into an existing database which may be not empty. The basic usage of this command is:

```
se_exp import dbname path
```

The parameter **dbname** specifies the database in Sedna to import data to. The **path** parameter specifies the directory with data to import.

The database **dbname** will not be created by this command. It is required that the target database already exist and run before starting the import process.

You can import data into database which contains some data and has some security politics. The only restriction is that there should not be any conflicts in the names of XML documents, collections or indices.

The main difference between `restore` and `import` command is that `import` command doesn't import any security information. All data is imported by the user who run the `se_exp` utility, i.e. by the user with the name and password specified with `se_exp` parameters. Of course this user should have enough rights to create collections, load documents and create indices.

2.5.2 File system level backup

An alternative strategy to backup a database is to directly copy the directories that Sedna uses to store the data of the database. Read Section 2.1 to find out where Sedna stores databases. You can use whatever method you prefer for doing usual file system backups. To restore a database, copy the corresponding backup directory to the location where Sedna stores databases.

There is a requirement is that the target database must be stopped in order to get a usable backup. Half-way measures such as disallowing all connections will not work.

Note that a file system backup will not necessarily be smaller than an back up via export. On the contrary, it will most likely be larger.

Note 2 *The database directory copied to different machine or different version of the same operating system might not work properly. If you want to restore a database on another machine or OS installation use `se_exp` utility instead.*

2.5.3 Hot Backup

Another alternative is to backup a database while it is still running. Such procedure is called hot backup. The purpose is to create a consistent backup copy while users are still performing some requests. This copy can then be restored by copying corresponding backup directory to the directory where Sedna stores databases. Such hot backups can be done in incremental mode, which allows more efficient archiving of database changes.

In a nutshell, when hot backup is called Sedna makes copies of all database files necessary to restore consistent database state in case of failure. The main difference between file system level backup (described in Section 2.5.2) and hot backup is that the target database has not to be stopped. As a tradeoff, restoration from hot backup copy may be a slower process, depending on the recency of the copy. Note, that hot backup copy guarantees durability of all transactions that had been committed at the moment of starting hot backup process.

To make a hot backup you must use provided `se_hb` utility. The usage is as follows:

Usage: `se_hb [options] dbname path`

options:

<code>-help</code>	display this help and exit
<code>--help</code>	display this help and exit
<code>-checkpoint</code>	make checkpoint before backup
<code>-time-dir</code>	create timestamp-subdir
<code>-make-dir</code>	create directory if it doesn't exist
<code>-incr-mode <increment_mode></code>	increment mode (start, add, stop)
<code>-port port-number</code>	port number to connect to Governor

<code>dbname</code>	the name of the database
---------------------	--------------------------

<code>path</code>	the name of the backup directory
-------------------	----------------------------------

The parameter `dbname` specifies the database to backup.

The parameter `path` specifies the directory to store files of the hot backup.

You can use `-checkpoint` option to make sure that checkpoint is made before hot backup process is started. This may make restoration process faster, since checkpoint fixates consistent state of the database and this state will be reflected in the hot backup copy. But at the same time backup process may take more time, depending on the user activity at the time of the backup.

If you want the destination directory (specified as `path` in command line) to be created, you must specify `-make-dir` option.

Note that `se_hb` may overwrite some of the previous backup files if you specify nonempty destination directory. So, if you make two consequent hot backups of the same database in the same directory, the older backup will be lost. `-time-dir` option prevents this by creating subdirectory named with the current date-time within `path` directory. In this case the destination of hot backup copy will be: `<path>/backup-<dbname>--<current date>--<current time>/`. It is recommended that you use the `-time-dir` option or provide a directory free of the previous backups.

You can specify port number to connect to governor through `-port` option. If port number is not specified in command line, hot backup process tries to find `sednaconf.xml` file and use port number specified as `listener_port` parameter. If it still cannot find port number, it will try to use default 5050 value.

Note 3 *You must run `se_hb` utility on the same machine as the target database is running. It will try to connect to the target database through the specified port to the localhost.*

Incremental hot backup and corresponding `-incr-mode` option will be explained below.

Incremental Hot Backup

Let us assume that you have made hot backup using command like this `se_hb mydb /backup`. If later you want to make another one to be sure updated data will be reflected in the backup copy, you can issue the same command. However, if the amount of changes is small, it is desirable to copy only this changes without making copy of the entire database again. This is where incremental mode becomes useful.

First of all, you must create a primary copy, which is essentially the copy of the entire database. You can do it by specifying `-incr-mode start` in command line (for example, `se_hb -incr-mode start mydb /backup`). Then, when you need to make subsequent hot backups of the same database, you can specify `-incr-mode add` in command line (for example, `se_hb -incr-mode add mydb /backup`). If the amount of changes is small, such backup process will take much less time.

Note, however, that when you use `-incr-mode start` option it will switch the original (active) database in incremental mode. This means it will start to store more files to allow “`-incr-mode add`” backups. In this case original database can grow in size more rapidly in case it is updated. To switch off incremental mode you must specify `-incr-mode stop` in command line. This command allows the original database to drop unnecessary files, but it also makes “`-incr-mode add`” option impossible. Thus, to start another incremental backup process you must repeat the whole process all over again (`-incr-mode start` call and additional `-incr-mode add` calls when needed). Note that `se_hb` with `-incr-mode stop` option does not make any additional hot backup copies, it just switches off incremental mode.

Note 4 *The database switches off incremental mode when new nonincremental (without `-incr-mode` option) hot backup is created. This is similar to the `-incr-mode stop` option, only in this case hot backup copy of the entire database is created.*

If you make new primary copy (i.e. with the `-incr-mode start` option specified) while database is still in incremental mode, `-incr-mode add` option will archive increments valid only for this last primary copy. It is recommended that you periodically make new primary copy with `-incr-mode start` option. Unless, of course, the database is rarely updated.

With incremental hot backup you have two options: you can archive all backups in the same directory or in different directories. If you archive all backups in the same directory (as in `/backup` in our example above) you can restore only state corresponding to the last incremental backup. On the other hand, storing incremental backups in different directories makes possible some kind of point-in-time recovery, i.e. you can restore state corresponding to any of the incremental backups. For the details see the next section.

Restore from Hot Backup Copy

Note 5 *Since hot backup is made on file system level basis, the same note as in the “File system level backup” section applies here too, i.e. recovery of the hot backup copy on different machine or different version of the same operating system cannot be guaranteed.*

Note 6 *Recovery of hot backup copy on different release of Sedna cannot be guaranteed. See Section 2.5.4 for further details.*

To restore the backed up database you must copy saved files to the directory where Sedna stores database files. For the information about Sedna directory structure read Section 2.1. For example, let us assume that hot backup have been made in `/backup` directory, and Sedna stores its files in `SEDNA_INSTALL` directory. In this case you can find `cfg` and `data` subdirectories in `/backup` directory. To restore database you must copy this directories in `SEDNA_INSTALL` directory. Note, that you should remove `SEDNA_INSTALL/data/<dbname>_files/` (where `<dbname>` is the name of the backed up database) directory before you copy backup files, since old files may interfere with restoration process. After you copy backup files in the corresponding directories, you can use `se_sm` command to start the database. When you do it for the first time SM runs recovery process to restore database state corresponding to the moment hot backup took place. It can take some time depending on the recency of the copy.

For the incremental backed up database the process may be different. If you have made all backups (primary copy and additional “`-incr-mode add`” copies) in the same directory the process is the same. However, if you have some of the “`-incr-mode add`” copies in the different directories you must copy files from all this directories in order corresponding hot backups were taken to fully restore database state. This makes possible restoration of older state of the database. For example, let us assume that primary copy is stored in `/backup/p` directory and additional `-incr-mode add` copies are stored in `/backup/1` and `/backup/2` directories in the order of creation. Then you can restore database to the state corresponding to any of the hot backups by copying only those directories that you need. For example, by copying `/backup/p` and `/backup/1` directories you can restore database state corresponding to the moment when `-incr-mode add /backup/1` was made. Of course, you cannot “skip” directories. For example, copying only `/backup/p` and `/backup/2` would not be possible, since in this case `/backup/1` is also needed.

2.5.4 Migration Between Releases

In this section we discuss how to migrate your data from one release of Sedna to another. As the internal data storage format is subject to change between different releases of Sedna it is a frequently required task to accurately migrate data.

It is recommended that you use **se_exp** utility to pass through this problem. The process consists of four steps.

1. Run **se_exp** utility with **export** command to export your data to some directory on the filesystem.
2. Shut down the old version of Sedna database server and remove it from your system.
3. Install and run a new version of Sedna, create the database with **se_cdb** and start it with **se_sm**. Make sure that no transaction has been run with the new database before restoring the data.
4. Run **se_exp** utility with **restore** command to restore data into the new database.

2.6 Event Log

Sedna keeps track of all important events that happen during its functioning. This section describes where log files are located and how to configure Sedna logging capabilities.

2.6.1 Files

Log files are located in the same folder where Sedna stores databases (see section 2.1, Sedna Directory Structure). By default it means that you can find event log files in **SEDNA_INSTALL/data**. Latest log file is named **event.log**. Sedna also stores old event log files which are named **event-{data}.log**.

2.6.2 Severity Level Configuration

Sedna provides a flexible way to set severity level of the logging output. It can be done either through **se_gov** command line parameter:

```
se_gov -el-level number
```

or defined in **sednaconf.xml** (see section 2.2.3, Configuring Sedna):

```
<event_log_level>number</event_log_level>
```

The following severity levels are available:

Level	Description
0	Event logging off.
1	Logs fatal errors that caused all database sessions to abort.
2	Logs all errors and warnings.
3	Logs detailed information (queries, IO statistics, etc).
4	Logs successively-more-detailed information for use by developers.

2.7 The Examples of Using Command Line Utilities

In this section we present several examples of using the utilities described in the previous sections. These examples demonstrate how to create the database, run the Sedna server and the database, load documents into the database, run the query over the database, and finally stop Sedna.

To run Sedna, type the following command:

```
se_gov
```

For creating a database named `xmark` with default settings use the following command:

```
se_cdb xmark
```

After execution of these commands the created database can be started up. To do it, type the following command:

```
se_sm xmark
```

The *xmark* database is ready for serving user queries. Below we show how to load an XML document to the *xmark* database and how to write a queries over this document.

To load the XML document into the *xmark* database, pass the following steps:

1. Create the following file with name `load.xquery`:

```
LOAD "<path>" "<name>"
```

The `<path>` is the path to the XML document to be loaded to the *xmark* database. The `<name>` is the name of this document in the *xmark* database.

2. Run the command:

```
se_term -file load.xquery xmark
```

3. Create the following file with name `query1.xquery`:

```
doc("<name>")/*
```

Where `<name>` is the name of the loaded document in the *xmark* database.

4. Type the command and enter it:

```
se_term -file query1.xquery xmark
```

To stop Sedna and all its components and databases, type and enter the following command:

```
se_stop
```


3 User-Based Authorization System

This chapter describes how to create and manage users and introduces the privilege system.

When created with `-db-security authorization` option, a database contains a set of database users. Those users are separate from the users managed by the operating system on which Sedna runs. The primary function of the Sedna privilege system is to authenticate a user and to associate that user with privileges on a database object such as `DROP`, `CREATE` or `QUERY`. Users own database objects (for examples, document) and can assign privileges on those objects to other users.

3.1 Database Users

Database user names are global across a database (and not per all Sedna databases). *Database users* interact with *database objects*. Every database object has its *owner* - the user that created it. Every user and *role* (we will discuss roles in the Section 3.2) has its *creator*.

In order to bootstrap the database, a freshly created database always contains one predefined DBA user with name `"SYSTEM"` and password `"MANAGER"`. To start your work with the database, you first have to connect as this initial user, then you can create more users and change default password (if you care for preventing unauthorized access to your database).

There are following kinds of Sedna database objects:

- Standalone document
- Collection of documents
- Value based index
- Full-text index
- Module
- Trigger
- Metadata document

There are two types of Sedna database users:

- Database administrator (DBA user). Formally, DBA user is a user that has the "DBA" role.
- Ordinary user (below we call "user")

DBA user:

- has all possible privileges on any object in the database;
- can remove any object in the database;
- can remove any user of the database;
- can grant/revoke any privilege to/from any user of the database;

- can grant "DBA" role to a user, thus making that user also a DBA user (not recommended, as the database with multiple DBA users is hard to administrate). Any DBA user can also revoke the "DBA" role from any DBA user.

An ordinary user:

- can act according to the privileges that he has;
- can grant and revoke any privileges on the database object that he owns to any user;
- can remove database objects that he owns and drop users that he has created.

Every user has its name and password.

To create a user use `CREATE USER` statement:

```
CREATE USER "user-name" WITH PASSWORD "user-password"
```

For example, the following statement:

```
CREATE USER "Alice" WITH PASSWORD "mypass"
```

creates user *Alice* identified with *mypass* password.

To remove an existing user, use `DROP USER` statement:

```
DROP USER "user-name"
```

For example, the following statement removes user *Alice*:

```
DROP USER "Alice"
```

A user can drop only a user he has created. DBA user can drop any user of the database.

To change user password use `ALTER USER` statement. A user can change a password for himself or for a user he has created. DBA user can change a password for any user of the database.

```
ALTER USER "user-name" WITH PASSWORD "new-password"
```

This statement changes the password of the user *user-name* to the *new-password*.

3.2 Privileges and Roles

When a database object is created, it is assigned an owner. The owner is the user that executed the creation statement. By default, only an owner and DBA user can do anything with the database object. In order to allow other users to use it, *privileges* must be granted.

There are several possible privileges:

- CREATE-USER - create new user
- CREATE-DOCUMENT - create new document
- CREATE-COLLECTION - create new collection
- CREATE-INDEX - create new value-based index
- CREATE-FT-INDEX - create new full-text index
- CREATE-TRIGGER - create new trigger
- LOAD-MODULE - load new module into database
- LOAD - load new document either into database or collection
- DROP - drop index, document, collection, user, role
- QUERY - query document, collection or database
- INSERT - perform update insert statements
- DELETE - perform update delete statements
- RENAME - perform update rename statements
- RETRIEVE-METADATA - retrieve metadata

Privileges are assigned to database objects or to the whole database. The table below for every privilege lists all kinds of database objects or the whole database it can be assigned to.

Privilege	can be assigned to
CREATE-USER	DATABASE
CREATE-DOCUMENT	DATABASE, COLLECTION
CREATE-COLLECTION	DATABASE
CREATE-INDEX	DOCUMENT, COLLECTION
CREATE-FT-INDEX	DOCUMENT, COLLECTION
CREATE-TRIGGER	DOCUMENT, COLLECTION
LOAD-MODULE	DATABASE
RETRIEVE-METADATA	DATABASE
LOAD	DATABASE, COLLECTION
DROP	DOCUMENT, COLLECTION, INDEX, MODULE, TRIGGER, FT-INDEX
QUERY	DOCUMENT, COLLECTION
INSERT	DOCUMENT, COLLECTION
DELETE	DOCUMENT, COLLECTION
RENAME	DOCUMENT, COLLECTION

Plus there is the key word **ALL** that denotes all possible privilege that can be granted to the specified object.

Note, DROP privilege on collection is granted to a user means that the user can drop any document in this collection (there is no such kind of database object as 'document in collection'). LOAD privilege on the database allows user to load standalone documents; LOAD privilege on a collection allows user to load documents into the collection.

Role is a named group of related privileges. Roles provide easy and controlled way to manage privileges. To create role CREATE ROLE statement is used:

```
CREATE ROLE "role-name"
```

This statement creates role with name *role-name*. When created role does not contain any privileges. It is not recommended to create a role with a name of an existing user (in this case privileges will be granted both to this user and to this role).

It is not allowed to use "DBA" and "PUBLIC" for naming roles as they are reserved by the system.

To destroy a role, use DROP ROLE:

```
DROP ROLE "role-name"
```

Role is automatically revoked from any user it was granted to.

3.3 Granting privileges

Privileges are granted to users so that users can access and operate with database objects or to process some tasks with a database.

A user can receive a privilege in two different ways: privileges can be granted to user explicitly; or privileges can be granted to role, and the role can be granted to one or more users. Roles allow easier and better management of privileges, thus privileges are normally granted to roles and not to specific users. Roles can be granted both to users and to roles.

A user who has granted a privilege or a role is *grantor* of this privilege (role).

To grant a privilege on a database object you must be an owner of this object or DBA user. To grant one or more privileges on a database object to one or more users or roles use:

```
GRANT "privilege" | ALL  
ON [DOCUMENT|COLLECTION] "database-object-name"  
TO "user-name|role-name" | PUBLIC
```

For example, the following statement:

```
GRANT "QUERY" ON DOCUMENT "auction" TO "Alice"
```

grants `QUERY` privilege on document *auction* to the user with name *Alice*. She will be able to perform any XQuery queries on this document.

The key word `PUBLIC` is used when the privileges are to be granted to all users, including those that may be created later. `PUBLIC` may be thought of as an implicitly defined role that every user has.

If the kind of the database object (`DOCUMENT` or `COLLECTION`) is not specified, database object is considered to be a document.

To grant a privilege on a database you must be DBA user. To grant one or more privileges on a database to one or more users or roles use:

```
GRANT "privilege" | ALL
ON DATABASE
TO "user-name|role-name" | PUBLIC
```

For example, the following statement:

```
GRANT "LOAD" ON DATABASE TO "Alice"
```

allows *Alice* to perform bulk load into the database.

To grant a role you must be allowed to grant every privilege of the role: to be owner of every database object of privileges of the role, or to be DBA user. To grant a role to another role means grantee will add grantor's privileges to its own set of privileges. To grant a role to one or more users or roles use:

```
GRANT "role-name"
TO "user-name|role_name" | PUBLIC
```

3.4 Revoking privileges

Privileges or roles can be revoked from the user. Roles cannot be revoked from roles, however. Only grantor of the privilege (role) or DBA user can revoke privilege (role).

`REVOKE` statements are similar to `GRANT` statements.

To revoke one or more privileges on a database object from one or more users or roles use:

```
REVOKE "privilege" | ALL
ON [DOCUMENT|COLLECTION] "database-object-name"
FROM "user-name|role-name" | PUBLIC
```

If the kind of the database object (`DOCUMENT` or `COLLECTION`) is not specified, database object is considered to be a document.

To revoke one or more privileges on a database from one or more users or roles use:

```
REVOKE "privilege" | ALL
ON DATABASE
FROM "user-name|role-name" | PUBLIC
```

To revoke a role from one or more users use:

```
REVOKE "role-name"
FROM "user-name" | PUBLIC
```

As mentioned above DBA user is a user that has a "DBA" role. Thus, "DBA" is a reserved name for a role: a role with name "DBA" can not be created, privileges or roles cannot be granted to "DBA" role.

A DBA user can grant "DBA" role to another user, thus making that user also a DBA user. This is not recommended, as multiple powerful users of a database can lead to hard database administration, and can cause insecure usage of the database and database objects.

3.5 Client Authentication

When a client application connects to the database server, it specifies which Sedna database user name it wants to connect as. User name determines access privileges to database objects, therefore, client authentication is used to restrict which database users can connect.

Authentication is the process by which the database server establishes the identity of the client and determines whether the client application (or the user who runs the client application) is permitted to connect with the user name that was requested.

When created with `-db-security authorization` or `-db-security authentication` option, database will check the user password on session open.

Currently, Sedna uses password authentication: client application that connects to the database must specify user name and user password (in `se_term` utility use `-pswd` and `-name` options; for client application that works through Sedna API see "Sedna Programmer's Guide"). Authentication process consists in checking the password correctness.

A newly created database always contains one predefined DBA user with name "SYSTEM" and password "MANAGER". To start your work with the database, you first have to connect as this initial user.

4 Localization

In the current version of Sedna all character data are stored internally in UTF-8. All input queries and XML documents must be encoded in UTF-8. All output is also encoded in UTF-8.