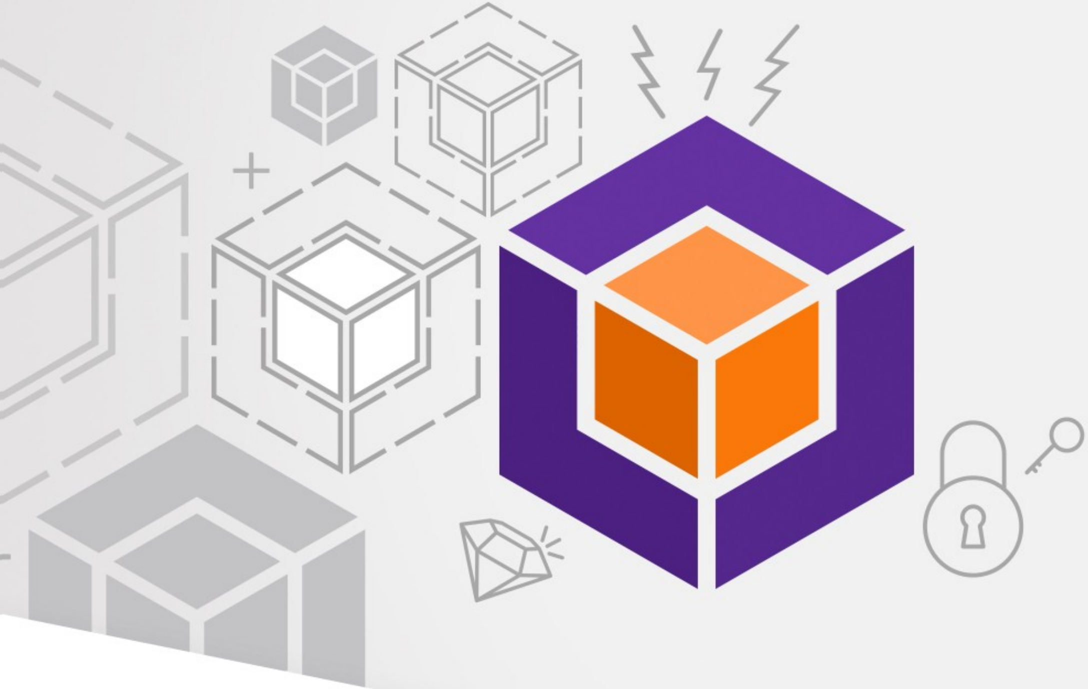
July 18, 2018



MUMPS as a C API

Subhead

Noteworthy Features

- Tight binding of database to language
- Dynamic linking
- Multitasking
- Interactive / incremental usage
- Hierarchical locks (traffic light semantics)
- ACID transactions

The Diamond is the Database

- Mature, proven code
  - “Rock Solid. Lighning Fast. Secure. Pick any three.”

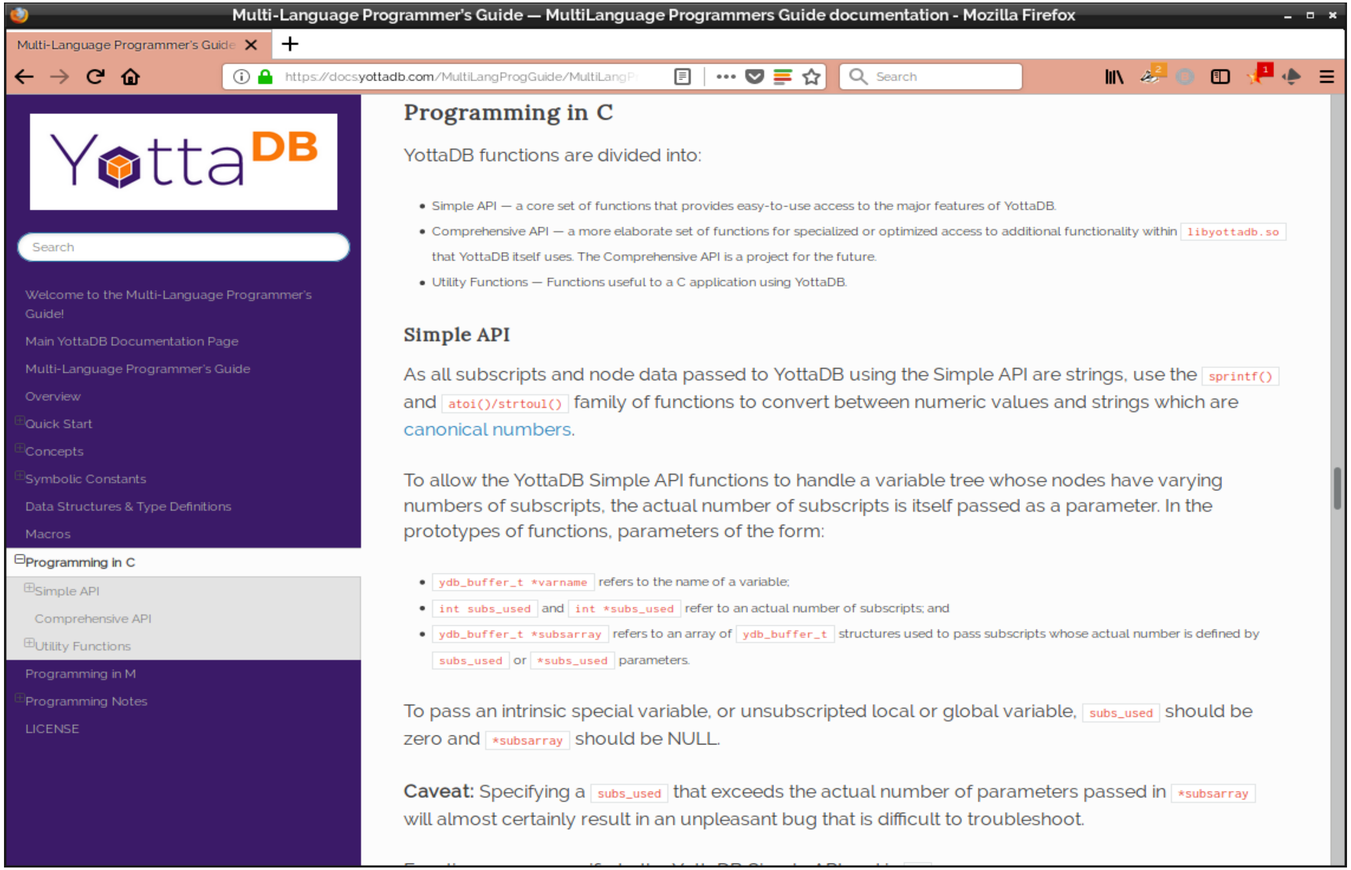
The Language is What it is

- You either love it or you hate it
  - Like anchovies on your pizza
  - or like emacs vs. vi[m] vs. …
  - or like your religion vs. the other guy's religion
  - or…

Noteworthy Features – Applicable to C

- Tight binding of database to language
- ~~Dynamic linking~~
- *Multitasking*
- ~~Interactive / incremental usage~~
- Hierarchical locks (traffic light semantics)
- ACID transactions

Multi-Language Programmers Guide


Multi-Language Programmer's Guide — MultiLanguage Programmers Guide documentation - Mozilla Firefox
Multi-Language Programmer's Guide
https://docs.yottadb.com/MultiLangProgGuide/MultiLangP
Search
YottaDB
Search
Welcome to the Multi-Language Programmer's Guide!
Main YottaDB Documentation Page
Multi-Language Programmer's Guide
Overview
Quick Start
Concepts
Symbolic Constants
Data Structures & Type Definitions
Macros
Programming in C
Simple API
Comprehensive API
Utility Functions
Programming in M
Programming Notes
LICENSE

Programming in C

YottaDB functions are divided into:

- Simple API — a core set of functions that provides easy-to-use access to the major features of YottaDB.
- Comprehensive API — a more elaborate set of functions for specialized or optimized access to additional functionality within `libyottadb.so` that YottaDB itself uses. The Comprehensive API is a project for the future.
- Utility Functions — Functions useful to a C application using YottaDB.

Simple API

As all subscripts and node data passed to YottaDB using the Simple API are strings, use the `sprintf()` and `atoi()/strtoul()` family of functions to convert between numeric values and strings which are canonical numbers.

To allow the YottaDB Simple API functions to handle a variable tree whose nodes have varying numbers of subscripts, the actual number of subscripts is itself passed as a parameter. In the prototypes of functions, parameters of the form:

- `ydb_buffer_t *varname` refers to the name of a variable;
- `int subs_used` and `int *subs_used` refer to an actual number of subscripts; and
- `ydb_buffer_t *subsarray` refers to an array of `ydb_buffer_t` structures used to pass subscripts whose actual number is defined by `subs_used` or `*subs_used` parameters.

To pass an intrinsic special variable, or unsubscripted local or global variable, `subs_used` should be zero and `*subsarray` should be NULL.

Caveat: Specifying a `subs_used` that exceeds the actual number of parameters passed in `*subsarray` will almost certainly result in an unpleasant bug that is difficult to troubleshoot.

available in a production release today

https://docs.yottadb.com/MultiLangProgGuide/

Elements of C API … 1

- Symbolic constants
  - Function return codes
    - Normal return codes, e.g., YDB_OK
    - Error return codes. eg., YDB_ERR_GVUNDEF
  - Limits, e.g., YDB_MAX_SUBS
  - Severity (in $zstatus), e.g., YDB_SEVERITY_WARNING
  - Other, e.g., YDB_DEL_NODE

Elements of C API … 2

- Symbolic constants
- Data structures and type definitions
  - `ydb_buffer_t` – `buf_addr`, `len_alloc`, `len_used`
    - Main structure for data value interchange
  - `ydb_string_t`
    - For continuity of existing code
  - `ydb_tpfnptr_t`
    - Function pointer for transaction processing

Elements of C API … 3

- Symbolic constants
- Data structures and type definitions
- Macros
  - Mostly for allocating and getting data into `ydb_buffer_t` structures
  - Some utility macros

Elements of C API … 4

- Symbolic constants
- Data structures and type definitions
- Macros
- Simple API
  - All essential functionality
  - Discuss in a few slides
  - Functions end in _s()

Elements of C API … 5

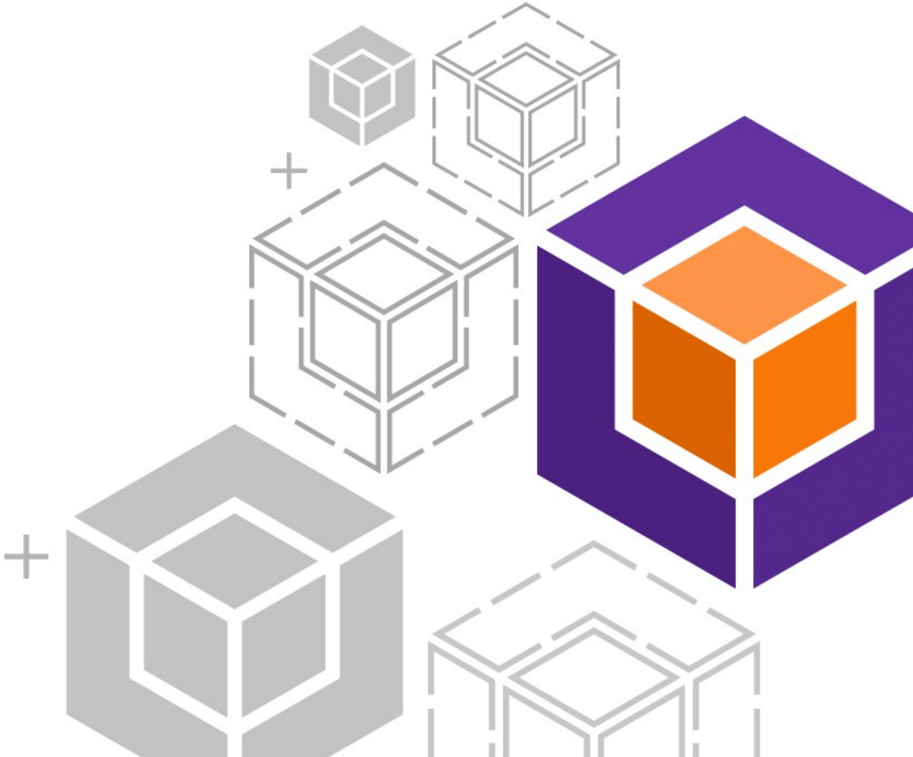
- Symbolic constants
- Data structures and type definitions
- Macros
- Simple API
- Comprehensive API
  - An exercise for the future, based on user experience with and feedback from Simple API

Elements of C API … 6

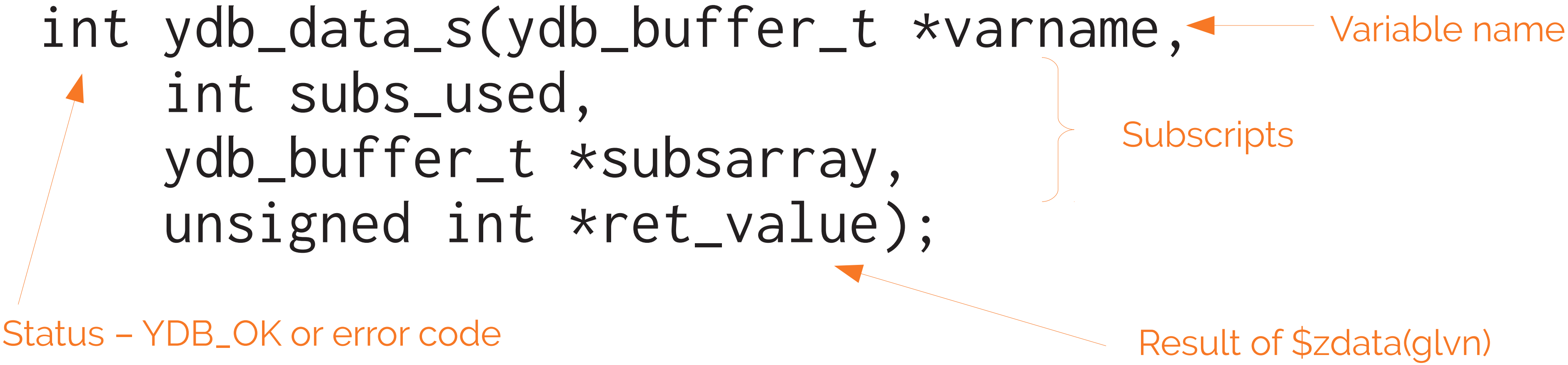
- Symbolic constants
- Data structures and type definitions
- Macros
- Simple API
- Comprehensive API
- Utility Functions

Simple API

Essential Functions



ydb_data_s() – $zdata()



```
int ydb_data_s(ydb_buffer_t *varname,
        int subs_used,
        ydb_buffer_t *subsarray,
        unsigned int *ret_value);
```

Variable name

Subscripts

Status – YDB_OK or error code

Result of $zdata(glvn)

ydb_delete_s() – kill

```
int ydb_delete_s(ydb_buffer_t *varname,
    int subs_used,
    ydb_buffer_t *subsarray,
    int deltype)
```

YDB_DEL_NODE or YDB_DEL_TREE

ydb_delete_excl_s() – zkill

```
int ydb_delete_excl_s(int namecount,
    ydb_buffer_t *varnames);
```

Names of local variables to save

ydb_get_s() – get node value

```
int ydb_get_s(ydb_buffer_t *varname,
    int subs_used,
    ydb_buffer_t *subsarray,
    ydb_buffer_t *ret_value);
```

Same signature as ydb_data_s()

ydb_incr_s() – $increment()

```
int ydb_incr_s(ydb_buffer_t *varname,
    int subs_used,
    ydb_buffer_t *subsarray,
    ydb_buffer_t *increment,
    ydb_buffer_t *ret_value);
```

ydb_lock_s() – lock

No untimed locks!

```
int ydb_lock_s(unsigned long long timeout_nsec,
   int namecount[,
   [ydb_buffer_t *varname,
   int subs_used,
   ydb_buffer_t *subsarray], ...]);
```

Variable number of parameters

Standard way to pass a name

ydb_lock_decr_s() – lock -

```
int ydb_lock_decr_s(ydb_buffer_t *varname,
    int subs_used,
    ydb_buffer_t *subsarray);
```

ydb_lock_incr_s() – lock +

```
int ydb_lock_incr_s(
    unsigned long long timeout_nsec,
    ydb_buffer_t *varname,
    int subs_used,
    ydb_buffer_t *subsarray);
```

No untimed locks!

ydb_node_next_s() – $query()

```
int ydb_node_next_s(ydb_buffer_t *varname,
    int subs_used,
    ydb_buffer_t *subsarray,
    int *ret_subs_used,
    ydb_buffer_t *ret_subsarray);
```

Same pattern for inpout and output subscripts

ydb_node_previous_s() – $query(,-1)

```
int ydb_node_previous_s(ydb_buffer_t *varname,
    int subs_used,
    ydb_buffer_t *subsarray,
    int *ret_subs_used,
    ydb_buffer_t *ret_subsarray);
```

ydb_set_s() – set

```
int ydb_set_s(ydb_buffer_t *varname,
    int subs_used,
    ydb_buffer_t *subsarray,
    ydb_buffer_t *value);
```

Same signature as ydb_get_s()

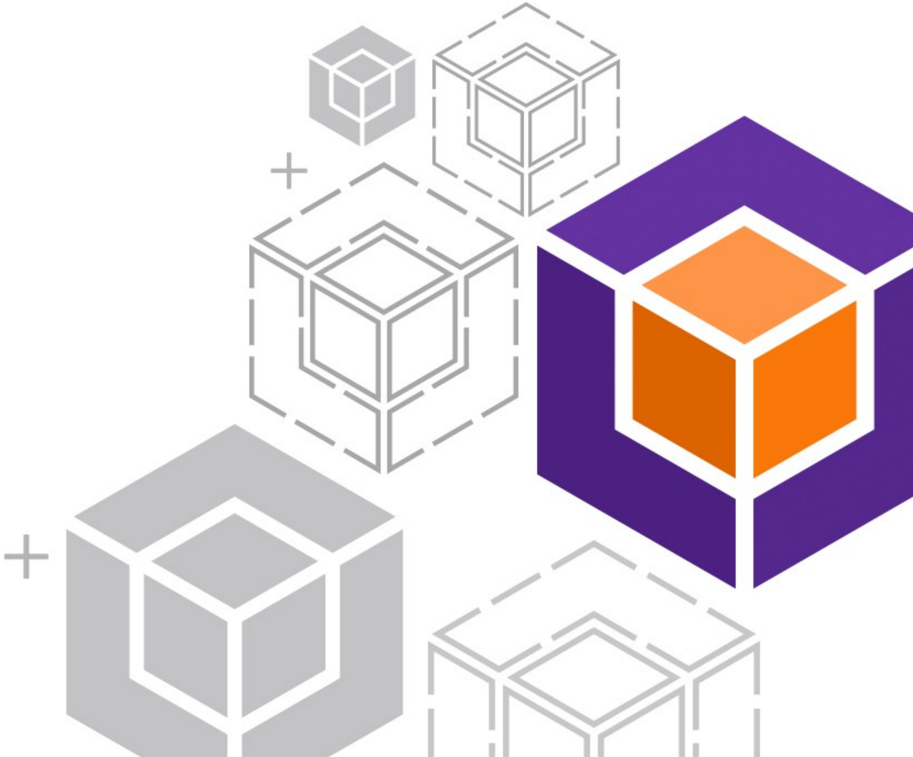
ydb_subscript_next_s() – $order()

```
int ydb_subscript_next_s(ydb_buffer_t *varname,
    int subs_used,
    ydb_buffer_t *subsarray,
    ydb_buffer_t *ret_value);
```

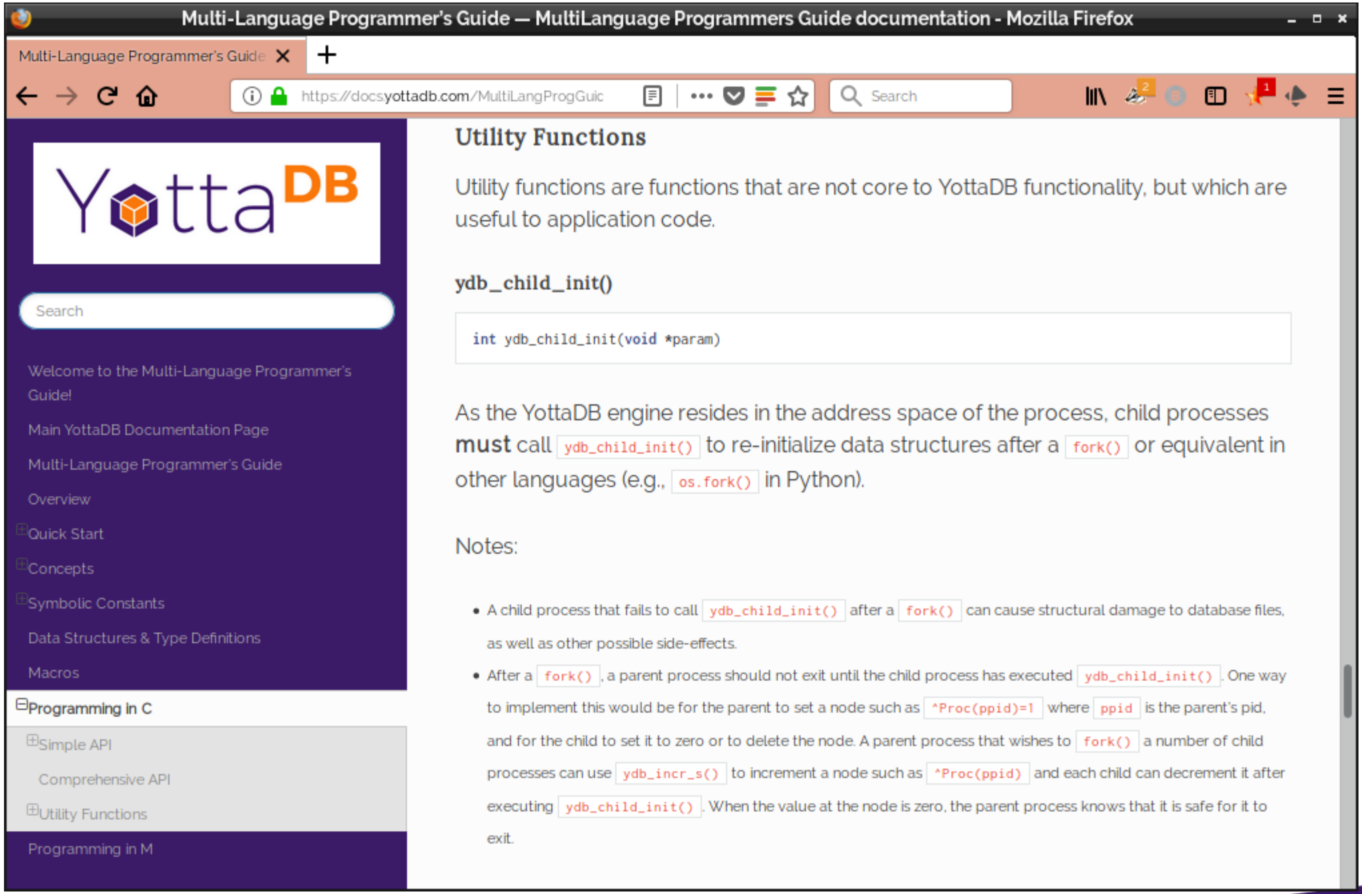
ydb_subscript_previous_s() – $order(,-1)

```
int ydb_subscript_previous_s(
    ydb_buffer_t *varname,
    int subs_used,
    ydb_buffer_t *subsarray,
    ydb_buffer_t *ret_value);
```

Utility Functions



Utility Functions


Multi-Language Programmer's Guide — MultiLanguage Programmers Guide documentation - Mozilla Firefox
Multi-Language Programmer's Guide
https://docs.yottadb.com/MultiLangProgGuic
Search
Search
Welcome to the Multi-Language Programmer's Guide!
Main YottaDB Documentation Page
Multi-Language Programmer's Guide
Overview
Quick Start
Concepts
Symbolic Constants
Data Structures & Type Definitions
Macros
Programming in C
Simple API
Comprehensive API
Utility Functions
Programming in M

Utility Functions

Utility functions are functions that are not core to YottaDB functionality, but which are useful to application code.

ydb_child_init()

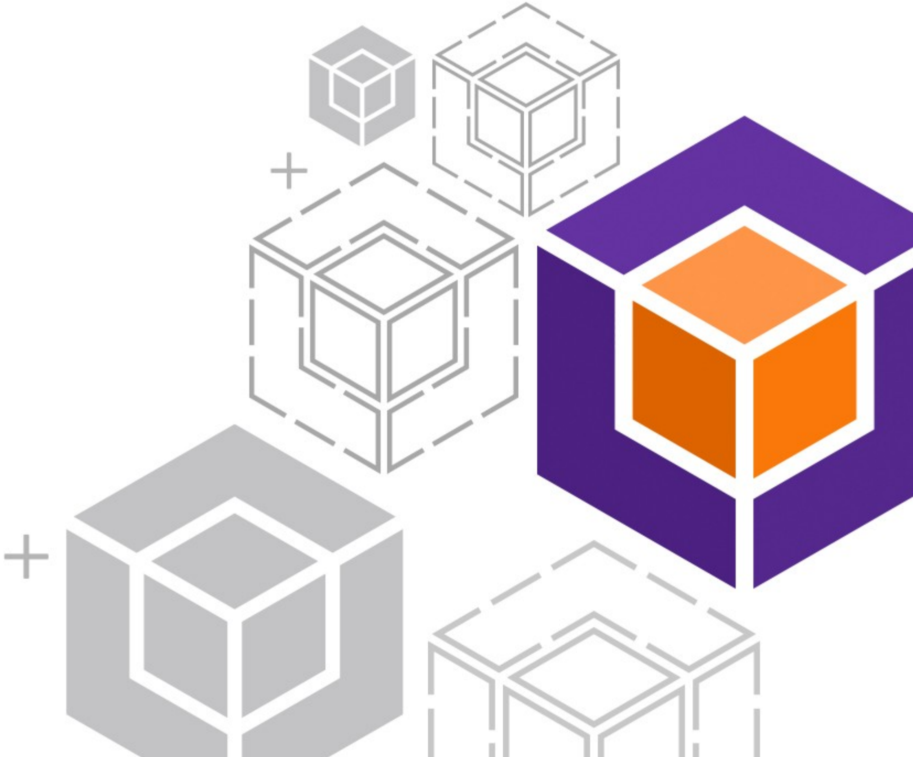
```
int ydb_child_init(void *param)
```

As the YottaDB engine resides in the address space of the process, child processes **must** call `ydb_child_init()` to re-initialize data structures after a `fork()` or equivalent in other languages (e.g., `os.fork()` in Python).

Notes:

- A child process that fails to call `ydb_child_init()` after a `fork()` can cause structural damage to database files, as well as other possible side-effects.
- After a `fork()`, a parent process should not exit until the child process has executed `ydb_child_init()`. One way to implement this would be for the parent to set a node such as `^Proc(ppid)=1` where `ppid` is the parent's pid, and for the child to set it to zero or to delete the node. A parent process that wishes to `fork()` a number of child processes can use `ydb_incr_s()` to increment a node such as `^Proc(ppid)` and each child can decrement it after executing `ydb_child_init()`. When the value at the node is zero, the parent process knows that it is safe for it to exit.

Demo



Links

- Web site – https://yottadb.com
- Quick start – https://docs.yottadb.com/MultiLangProgGuide/MultiLangProgGuide.html#quick-start
- User documentation – https://yottadb.com/resources/documentation/
- Blog - https://yottadb.com/blog/
- Contact – K.S. Bhaskar / bhaskar@yottadb.com

YottaDB

Thank You!

yottadb.com