# TPCACHE.INI(5)

**REVISION HISTORY**

| NUMBER | DATE | DESCRIPTION | NAME |
|---|---|---|---|
| | | | |

# Contents

# Chapter 1

# SYNOPSIS

```
[@cachedb/DBNAME_SECTION]
cachedb=DBNAME_DB
resource=DATABASE_DIRECTORY
flags=FLAGS
limit=LIMIT
expiry=EXPIRY
max_readers=MAX_READERS
max_dbs=MAX_NAMED_DBS
map_size=MAP_SIZE
perms=DB_PERMISSIONS


[@cachedb/DBNAME_2]
...

[@cachedb/DBNAME_N]
...


[@cache]
svc SERVICE_NAME=
    {
        "caches":[
                {
                    "cachedb":"CACHE_DBNAME",
                    "flags":"CACHE_FLAGS",
                    "type":"TYPE",
                    "subtype":"SUBTYPE",
                    "rule":"RULE",
                    "keyfmt":"KEYFMT",
                    "save":"SAVE",
                    "delete":"DELETE",
                    "rsprule":"RSPRULE",
                    "refreshrule":"REFRESHRULE",
                    "inval_svc":"INVAL_SVC",
                    "inval_idx":"INVAL_IDX",
                    "keygrpdb":"KEYGRPDB",
                    "keygrpfmt":"KEYGRPFMT",
                    "keygrpmaxtperrno":"KEYGRPMAXTPERRNO",
                    "keygrpmaxtpurcode":"KEYGRPMAXTPURCODE",
                    "keygroupmrej":"KEYGROUPMREJ",
                    "keygroupmax":"KEYGROUPMAX"
                },
                {
```

```
                    "cachedb":"DBNAME_2",
                    ...
                },
                ...
            ]
    }
svc SERVICE_NAME_N=
    {
        "caches":[
                {
                    "cachedb":"DBNAME_N",
                    ...
                },
                ...
            ]
    }
```

# Chapter 2

# DESCRIPTION

Enduro/X supports tpcall(3) caching, by writing results into LMDB. LMDB is integral par to of the Enduro/X. To avoid conflict with users choice of LMDB version, the embedded version is renamed to "EXDB", so that global C symbols does not conflict. To configure Enduro/x tpcall caching, firstly database must be configured. Enduro/X uses approach, that each database is stored into separate file system directory. On Linux systems (or others which support RAM drivers), the directory can be defined in RAM drive, and thus avoid the cache writing to disk (as the LMDB uses memory mapped files). On Linux for this purpose /dev/shm can be used.

Caching is performed at service level. Thus when software is doing **tpcall(3)**, the client's call is intercepted, and tested against cache. If saved data is found, then data is directly returned from LMDB and no service invocation is done.

Each cached item must have a key. The key is build from the call data. Key is zero (0x00 byte) terminated string. Key format is given in cache configuration. Before lookup to DB is made, rule expression is tested. This indicates either the call must be cached or not. If rule is true, key is built and with key DB is tested, if result is found, it is returned to caller. If result is not found, then service invocation is performed. When service responds, data is written to cache.

Service can have multiple caches, with different rules. Once request is processed the first cache which matches the rule is used for lookup or data saving.

Each cache references database definition, single database can be use by multiple caches or each cache can have it's own database. Keeping multiple databases could make better performance for caches were often writes are done. As concurrent writes are synchronized so that only one process writes to the same database.

If writing to the same database, user shall ensure that keys are unique between different service cache definitions. This can be reached by adding different prefixes in key format.

# Chapter 3

# DEFINITIONS

**Simple cache** - this is cache were data is saved in one cache database. No linked databases are used.

**Keygroup** is separate database where records can be grouped. For example by user id. The group record holds the **UBF** buffer with string key occurrences which are linked in the group. Keygroup database name is encoded as

**Keyitem** is linked record to the group.

# Chapter 4

# DATABASE DEFINITION

**DBNAME_SECTION**
In case of simple database. This is a name of the database. For example "cachedb1". Allowed symbols a-z, A-Z, 0-9. In case of keygroup or keyitems databases, format for section is "<dbname>/<logical_db_name>. For the caches the db names are encoded as "<logical_db_name>@<dbname>". Parameter is **mandatory**.

**DBNAME_DB**
This is database name. For simple caches, name must be the same as **DBNAME_SECTION**. For keygroup or keyitems databases name format is "<logical_db_name>@<dbname>". Parameter is **mandatory**.

**DATABASE_DIRECTORY**
This is file system directory where cache database is located. The directory must exist when application domain is started. In case of keygroup and keyitems databases, the directory must be the same. Parameter is **mandatory**.

**FLAGS**
This is comma separated list of keywords - flags. See next section for list of flags supported for cache db.

**LIMIT**
If number of cached items in database must be limited (for example 1000 recs max), then the number is defined by this parameter. If this parameter is set then one of the following flags must be present **fifo**, **lru** or **hits** - delete strategy. So that **tpcached(8)** process would know how to sort and which records to remove. The limit is not guaranteed maximum. As records are removed by **tpcached** daemon, there could be times that limit is overreached, because **tpcached** works with periods. And during the sleep time, more records could be added to db and only after sleep period **tpcached** will zap them.

**EXPIRY**
Time specification for record to live in cache. After time is passed, the **tpcached** process will zap the record. The configuration is specified as: N+s for seconds, (e.g. 20s), N+.D+m for minutes (e.g. 30.5m - 30 min, 30 sec) or N+.D+h for hours (e.g 3.5h, will store message for 3 hours and 30 minutes).

**MAX_READERS**
See LMDB documentation for this. Basically this is number of threads used by process. See LMDB's mdb_env_set_maxreaders() function description. The default value set by Enduro/X is **1000**.

**MAP_SIZE**
Maximum size of the database in bytes. The size must be multiple of OS page size. See LMDB's mdb_env_set_mapsize() function description. The default value used by Enduro/X is **10485760**. Postifx multiplier can be used for value in configuration: kK(x1000) mM (x1000'000) gG (x1000'000'000) e.g. 10M.

**MAX_NAMED_DBS**
Maximum number of "named" logical databases in given resource. Named DB is only uses with "@" syntax, and usually only for keygroups (to keep transactions atomic between two DBs). The default value is **2**.

**DB_PERMISSIONS**
Octal permissions for map files on file system. The default value is **0664**.

# Chapter 5

# DATABASE FLAGS

**bootreset**
Clean up the cache database when Enduro/X application domain is booting. The reset is performed by **tpcachebtsv** binary. Thus this binary must be configured in **ndrxconfig.xml(5)**. And it should be one of the first XATMI servers booted. The **tpcachetbsv** simple removes database files at boot.

**bcastput**
Should process perform broadcast (send event) of putting data in cache. If this is set and data is put into cache. This invokes calling **tpevsrv** with current tpcall response data buffer. Event is @CPNNN//<SVCNM>. Where NNN is cluster node id.

**bcastdel**
When cache data is deleted either by invalidate cache or by **tpcached** operations, if flag is set, the event is broadcasted to **tpevsrv**. The event in case of invalidated is @CDNNN/F/<SVCNM> (which contains the data buffer by which data is deleted. It is buffer which performs invalidate, not the actually stored data. Data is processed by cache's **delete** parameter which might limit the data sent for deletion over the event server - in order to save some traffic). The NNN is cluster node id which initiates the invalidate. The F is flags, in case if is *G*, then whole group is deleted if record is part of the keygroup, otherwise *F* flag (full) is sent.

**timesync**
In case if of network update received (add to cache) message, the message will be added to cache with out replacing the existing record. The caller will receive the latest record by time stamp.

**scandup**
In case of **timesync** mode, the database may accumulate duplicate records, thus if this flag is set, the **tpcached** daemon will scan the cache records and erase the older records. Note that duplicates also are removed during real time data lookup.

**clrnosvc**
Clear cache when service for which data is cached, is not advertised anymore by any XATMI server. The cleanup is performed by **tpcached**.

**keyitems**
This is key-items database. Required flag for keygroup items database pair.

**keygroup**
This database is for storing key-groups. Note that when using key-group, both keyitems and keygroup must be in the same physical resource. And syntax for boths databases must be "<logical_db_name>@<dbname>".

**nosync**
Do not perform fsync (flush to disk) when committing transaction. This is suitable for caches which are reset at boot. Used for performance increase. In case of persisted databases (used after reboot), at case of crash, this data might be corrupt. See **MDB_NOSYNC** mode for mdb_env_open().

**nometasync**

Flush metadata only when doing commit. The risks are the same as with **nosync**. Recommended for non persisted caches. See **MDB_NOMETASYNC** for mdb_env_open() function.

# Chapter 6

# CACHE DEFINITION

**CACHE_DBNAME**
Reference to database name. This must match with chosen storage db **DBNAME_DB**.

**CACHE_FLAGS**
Flags of the cache (comma separate of flags for cache). See section bellow for flags available.

**TYPE**
This is buffer type name for which cache is designed. Currently supported type is **UBF**.

**SUBTYPE**
This is buffer sub type. For example for **VIEW** buffers this is view name. Currently this parameter is reserved for future use.

**RULE**
This is buffer type specific expression for defining the rule in which case the call shall be cached/cache-looked-up or not. If rule is not specified, then by default call is accepted for cache. In case if **TYPE** is UBF, then standard boolean expression applies here.

**KEYFMT**
This is buffer type specific format. For **UBF** fields this is format string which might contain field free text with format of *$(UBF_FIELD)*, where given construction will be substituted with UBF field value of field *UBF_FIELD*. For example having *T_STRING_3_FLD* equal to **ABC** and *T_STRING_2_FLD* occurrence 2 equal to XX' and format string *SV1$(T_STRING_3_FLD)-$(T_STRING_2_FLD[1])*, then key will render as *SV1ABC-XX*. Key is used to distinguish under which record to save the data and how to lookup data into database.

**SAVE**
This field specifies how data is saved into cache. It is type specific and and flags specific. For **UBF** buffer with flag **putfull** full UBF buffer is saved. If flag is not specified and *SAVE* is set to * then full buffer is saved too. If flags is not specified, but it is comma separated list of **UBF** fields, then only those fields are saved to cache. If flags is set to **putrex**, then *SAVE* field shall contain regular expression which is executed on buffer fields (names). The names which matches expression are added to cache.

**DELETE**
This field specifies field which shall be broadcast to other cluster node in case if invalidate their cache is performed. This makes a projection of incoming buffer in order to save the traffic. The projected buffer is then sent to event server for other cluster node processing. Rules for the field syntax is similar to the *SAVE* parameter. For **UBF** buffer with flag **delfull** full UBF buffer is sent in event. If flag is not specified and *SAVE* is set to * then full buffer is sent too. If flags is not specified, but it is comma separated list of **UBF** fields, then only those fields are sent in event. If flags is set to **delrex**, then *DELETE* field shall contain regular expression which is executed on buffer fields (names). The names which matches expression are added to to buffer which is sent to other cluster node.

**RSPRULE**

This is boolean expression with following two fields in buffer **EX_TPERRNO** which corresponds to **tperrno** of tpcall() and **EX_TPURCODE** corresponds to tpcall() user return code. If **RSPRULE** setting is not present, then cached are only successful calls (i.e. EX_TPERRNO==0).

**REFRESHRULE**

Optional type specific expression. When performing tpcall(), the saved data is returned from cache. In case if this expression is defined and executes true on tpcall buffer,the real service all will be performed, even if data in cache are present. For **UBF** buffers this is boolean expression.

**INVAL_SVC**

This is service name if "their" service to be invalidated. When service call is performed, this "allows" to intercept the call and when response is received the other service cache, specified in *INVAL_SVC* will be reset (removed cached data). The key used for data access is build by this invalidate cache and not by target *INVAL_SVC* cache definition. If target cache uses **invalkeygrp** flag, then then whole group is invalidated. The group access is made by *KEYGRPDB* and *KEYGRPFMT*.

**INVAL_IDX**

This is cache index (zero based array index) of target invalidate service cache. This must be defined in case if *INVAL_SVC* parameter is used, this parameter must be set too.

**KEYGRPDB**

Optional key group database name. The database must be consists be in format of <logical database>@<physical database name>. If using *KEYGRPDB* DB, then the *CACHE_DBNAME* must be set in the same format. So that logical names for cache database is different but physical is the same. Physical resources must be the same for LMDB, for need to ensure transactional consistency between group and items.

**KEYGRPFMT**

This format string to build a key for a group. It shall be lower level of object "resolution", to have something common between linked *KEYFMT*. For example if paging (user iterates over her statement) needs to be cached and invalidated at any single transaction, then *KEYGRPFMT* is the userid and *KEYFMT* shall contain userid and page number. The syntax for *KEYGRPFMT* is the same as for *KEYFMT* is buffer type specific. Field is **conditional**, must be present when configuring key group.

**KEYGROUPMAX**

Key group can be used to protect against DoS attacks - by limiting number of "new" records that can appear in key group. The time component is processed by **tpcached** binary, which could remove key group after it is expired. They *KEYGROUPMAX* is number records allowed in key group. Parameter is **optional**.

**KEYGROUPMREJ**

This is buffer definition which shall be filled in response (merged or replaced according to flags) in case if cache lookup is made, request is part of key group, record does not exists and number of records in key group already reached *KEYGROUPMAX*. In this case service call is denied and response is filled with buffer definition found in this parameter. For **UBF** buffers, it is UBF buffer defined in JSON format (see tpjsontoubf(3) man page). Parameter is **conditional** and must be present in case if *KEYGROUPMAX* is set.

**KEYGRPMAXTPERRNO**

In case of doing reject for *KEYGROUPMAX* max reached, this is TP error code that tpcall(3) will return. Normally it would be **11** (service routine error). Parameter is **optional**, if not present the error value will be **0** - no error, thus data should indicate that reject was performed.

**KEYGRPMAXTPURCODE**

Value of user return code in case if doing key group max reject. See **tpurcode(3)**. Parameter is **optional** and if not set and reject will be performed, user return code value will be **0**.

# Chapter 7

# CACHE FLAGS

**putfull**
Flag indicates that when saving data to cache full XATMI buffer shall be saved. In case if **putfull** or **putrex** is not present then *SAVE* parameter is analyzed.

**putrex**
Indicates that *SAVE* field is regular expression. **putfull** and **putrex** flags cannot be mixed.

**getreplace**
Flag indicates that if data is returned from, then buffer passed to **tpcall(3)** must be replaced with data from cache.

**getmerge**
Flag indicates that if data is returned from, then buffer passed to **tpcall(3)** must be merged with incoming data. If using **UBF** buffer type then for merge operation **Bupdate(3)** is used. Where destination buffer is buffer passed to **tpcall(3)** and source buffer is one save in cache.

**nosvcok**
If flag is set and service call is made (**tpcall(3)**) to service which is not currently advertised, but cached data exists in cache, then return result from cache as if the the service was advertised and working ok. For such case if flag is not present, the system will respond with error TPENOENT.

**next**
Flag indicates that next cache (if having multiple for single service) should be processed, after this one. Flag is suitable for cases when current cache is invalidate their. Thus invalidate is performed and for example at next cache would be actually save results. Or multiple invalidates can be configured.

**delfull**
When using invalidate cache, then this flag indicates that full XATMI buffer sent to tpcall() shall be broadcasted (published in event) to other nodes. In case of UBF buffer if flag is not present, then * in *DELETE* cache parameter indicates all fields must be Broadcast. If not * then it is comma separated list of fields.

**delrex**
This indicates that *DELETE* parameter is regular expression. In case of UBF buffer, the expression is executed on each field in incoming buffer, if field name matches expression, then it is added to broadcast list.

**inval**
Flag indicates that this cache is invalidate. In this case cache parameters *INVAL_SVC* and *INVAL_IDX* must be present.

**invalkeygrp**
If performing invalidated of record and record is part of key group, then delete whole key group (all linked records and group by it self). If flag is not present, then single record is delete (keyitem) and group is updated (linkage removed from group). This flag applies to all mechanisms how record can be removed, either by invalidate their, or zapped by **tpcached** or deleted by xadmin tooling.

# Chapter 8

# PERFORMANCE

For performance reasons, if non persistent cache is required, it is recommended to store data file in RAM driver, for example on GNU/Linux systems it is **/dev/shm**. It is considered that in such scenario LMDB will use twice the memory. As one is a copy in RAM drive and another is mapped pages in process to the file. It is up to kernel realization to make some optimizations here.

# Chapter 9

# EXAMPLE

Simple caches

```
[@cachedb/db02_1]
cachedb=db02_1
resource=${TESTDIR_DB}/db02_1
flags=bootreset

[@cachedb/db02_2]
cachedb=db02_2
resource=${TESTDIR_DB}/db02_2
flags=bootreset


[@cache]
svc TESTSV02=
    {
        "caches":[
                {
                    "cachedb":"db02_1",
                    "type":"UBF",
                    "keyfmt":"SV2-$(T_STRING_FLD)",
                    "save":"*",
                    "rule":"T_STRING_2_FLD==’HELLO CACHE 1’"
                },
                {
                    "cachedb":"db02_2",
                    "type":"UBF",
                    "keyfmt":"SV2-$(T_STRING_FLD)",
                    "save":"T_STRING.*|T_FLOAT.*|T_LONG_2_FLD|T_SHORT.*",
                    "flags":"putrex"
                }
            ]
    }

svc SOMEOTHERSVC=
    {
        "caches":[
                {
                    "cachedb":"db02_1",
                    "type":"UBF",
                    "keyfmt":"SVOTHER-$(T_STRING_FLD)",
                    "save":"*",
                }
            ]
```

```
    }
```

Cache with keygroup and buffer reject:

```
[@cachedb/db15]
max_dbs=2
resource=${TESTDIR_DB}/db15
subscr=@C.001/.*/.*|@C.002/.*/.*

#
# These two inherits settings from above.
#
[@cachedb/db15/g]
cachedb=g@db15
flags=bootreset,bcastput,bcastdel,keygroup
expiry=30s

[@cachedb/db15/k]
cachedb=k@db15
flags=bootreset,bcastput,bcastdel,keyitems
expiry=10s


[@cache]

svc TESTSV15=
    {
        "caches":[
                {
                    "cachedb":"k@db15",
                    "keygrpdb":"g@db15",
                    "type":"UBF",
                    "keyfmt":"SV15$(T_STRING_FLD)-SV15$(T_SHORT_FLD)",
                    "keygrpfmt":"SV15$(T_STRING_FLD)",
                    "save":"T_STRING_FLD,T_STRING_2_FLD,T_LONG_2_FLD,T_SHORT_FLD",
                    "flags":"getmerge",
                    "keygroupmax":"7",
                    "keygroupmrej":"{\"T_STRING_3_FLD\":\"REJECT\",\"T_LONG_2_FLD\":[\"1\", ↩
                        \"2\"]}",
                    "keygrpmaxtperrno":"11",
                    "keygrpmaxtpurcode":"4"
                }
            ]
    }
```

For more unit tests please see *atmitest/test048_cache* unit test folder ini files.

# Chapter 10

# BUGS

Report bugs to support@mavimax.com

# Chapter 11

# SEE ALSO

**xadmin(8) ndrxd(8) ndrxconfig.xml(5) tpcached(8) tpcachesv(8) tpcachebtsv(8)**

# Chapter 12

# COPYING