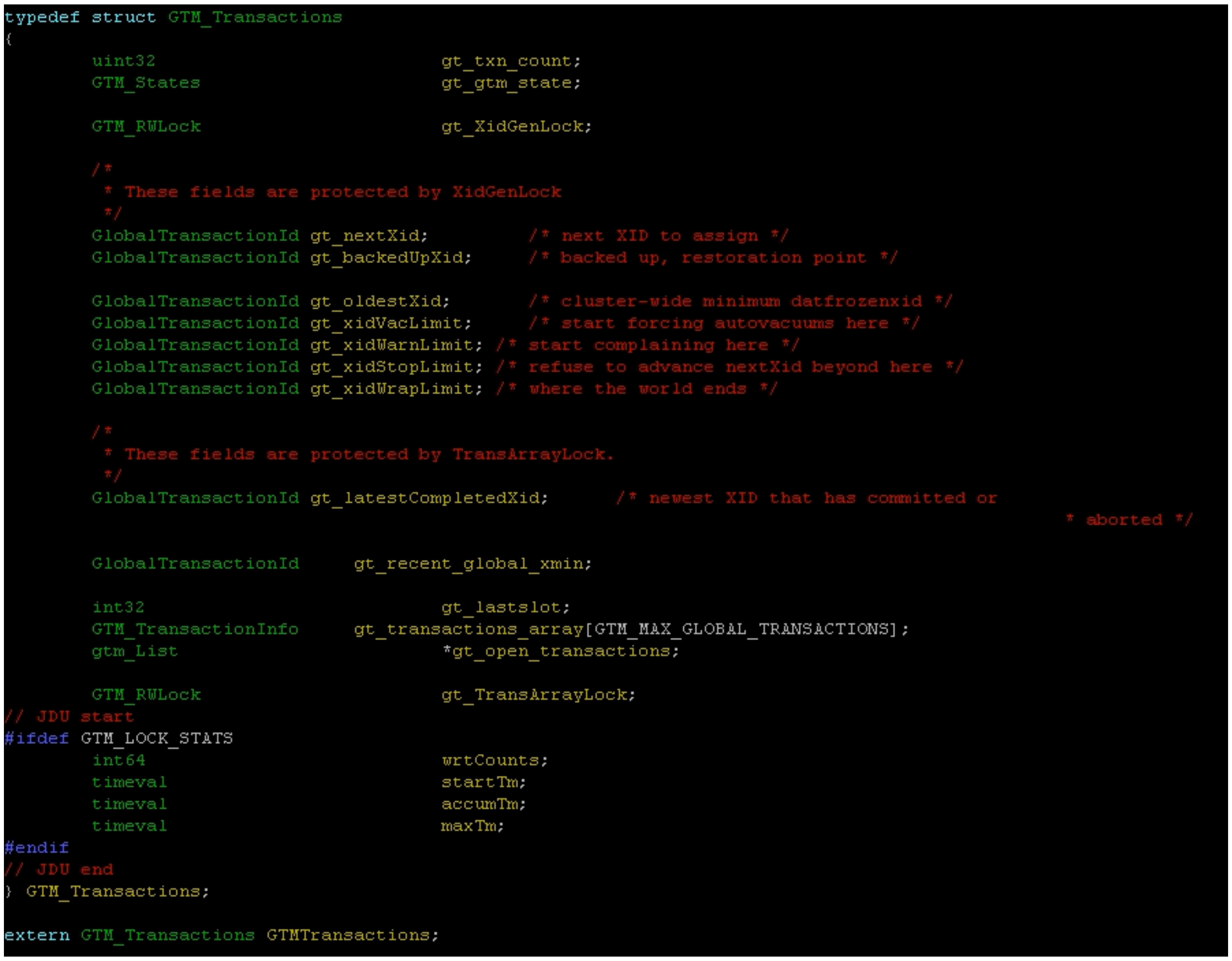
snapshot	“xmin” and “xmax” are used to indicate what transactions are running at a specific time. When a transaction tries to read a table tuple, each tuple has a set of XIDs to indicate transactions which created and deleted the tuple.
xmin	when we insert rows, XID of inserting transation is recorded at xmin field.
xmax	when we update rows, the old row is marked as "deleted" by writing updateing transaction's XID to xmax field. At the same time the new updated row is created whose xmin field is marked with XID of the updating transaction.
gtm_List *gt_open_transactions	./src/gtm/common/gtm_list.cpp ./src/include/gtm/gtm_list.h a dynamic list containing all open transactions at a given time.
GTM_TransactionInfo *gtm_txninfo	src/include/gtm/gtm_txn.h a fixed array with 8192 entries **gt_lastslot**: help locating new element from the gtm_txninfo. it keeps last used element in the array and search for an empty element for the new transaction from **startslot** = **gt_lastslot**+1; **gt_lastfreed**: keep the index of last freed gtm_txninfo array element. For a begin transaction call, if its value is valid, used it to get new element before checking gt_lastslot. Once used, gt_lastfreed is set to an invalid value and gets a valid value set only when the remove transaction call is executed.
LL_*	./src/backend/libcomm/sctp_utils/sctp_uthash/utlish.h
a write lock GTM_RWLockAcquire()	the duration of holding the lock would depend on (how soon an empty element can be located in the gtm_txninfo array) [ProcessingBeginTransactionCommand()] and (the number of entries in the open transactions list) [closedown].

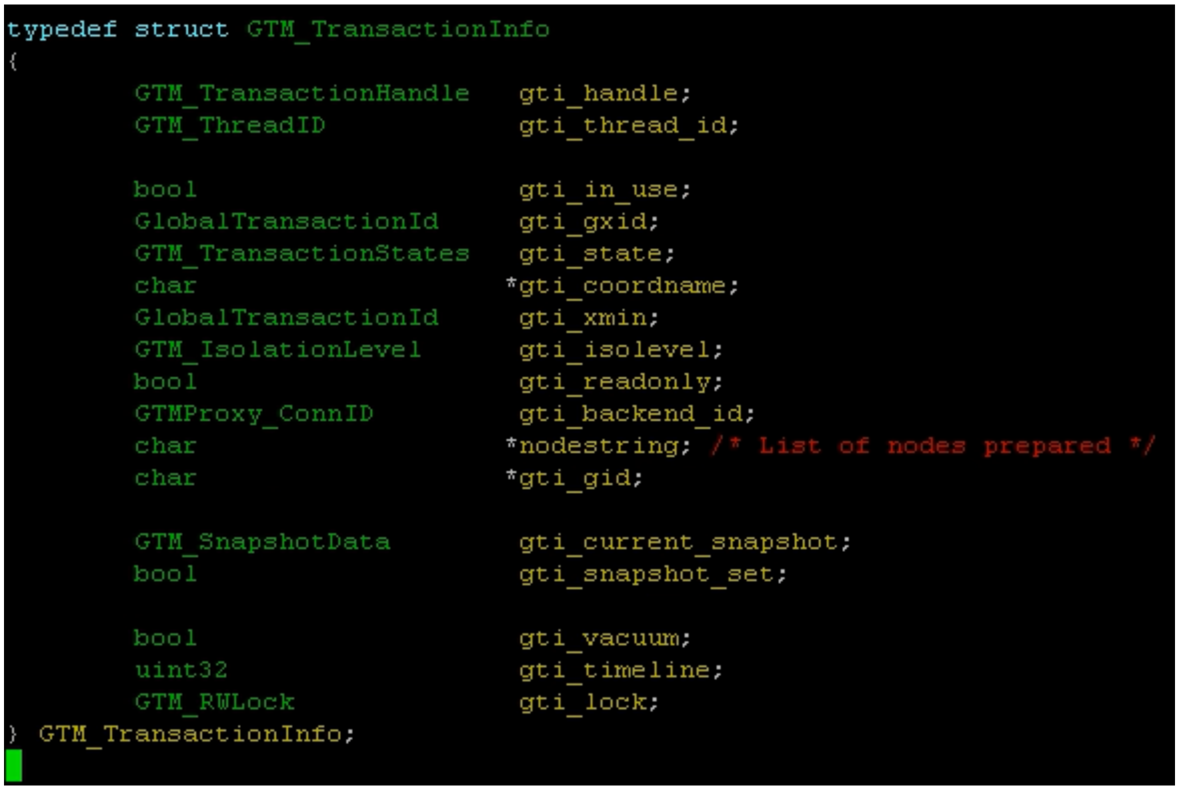
Two costly functions	GTM_GXIDToHandle()
	GTM_GetSnapshot()

Src/include/



```
typedef struct GTM_Transactions
{
        uint32                          gt_txn_count;
        GTM_States                      gt_gtm_state;

        GTM_RWLock                      gt_XidGenLock;

        /*
         * These fields are protected by XidGenLock
         */
        GlobalTransactionId gt_nextXid;         /* next XID to assign */
        GlobalTransactionId gt_backedUpXid;     /* backed up, restoration point */

        GlobalTransactionId gt_oldestXid;       /* cluster-wide minimum datfrozenxid */
        GlobalTransactionId gt_xidVacLimit;     /* start forcing autovacuums here */
        GlobalTransactionId gt_xidWarnLimit; /* start complaining here */
        GlobalTransactionId gt_xidStopLimit; /* refuse to advance nextXid beyond here */
        GlobalTransactionId gt_xidWrapLimit; /* where the world ends */

        /*
         * These fields are protected by TransArrayLock.
         */
        GlobalTransactionId gt_latestCompletedXid;      /* newest XID that has committed or
                                                                                 * aborted */

        GlobalTransactionId     gt_recent_global_xmin;

        int32                           gt_lastslot;
        GTM_TransactionInfo     gt_transactions_array[GTM_MAX_GLOBAL_TRANSACTIONS];
        gtm_List                        *gt_open_transactions;

        GTM_RWLock                      gt_TransArrayLock;
// JDU start
#ifdef GTM_LOCK_STATS
        int64                           wrtCounts;
        timeval                         startTm;
        timeval                         accumTm;
        timeval                         maxTm;
#endif
// JDU end
} GTM_Transactions;

extern GTM_Transactions GTMTransactions;
```

Src/include/



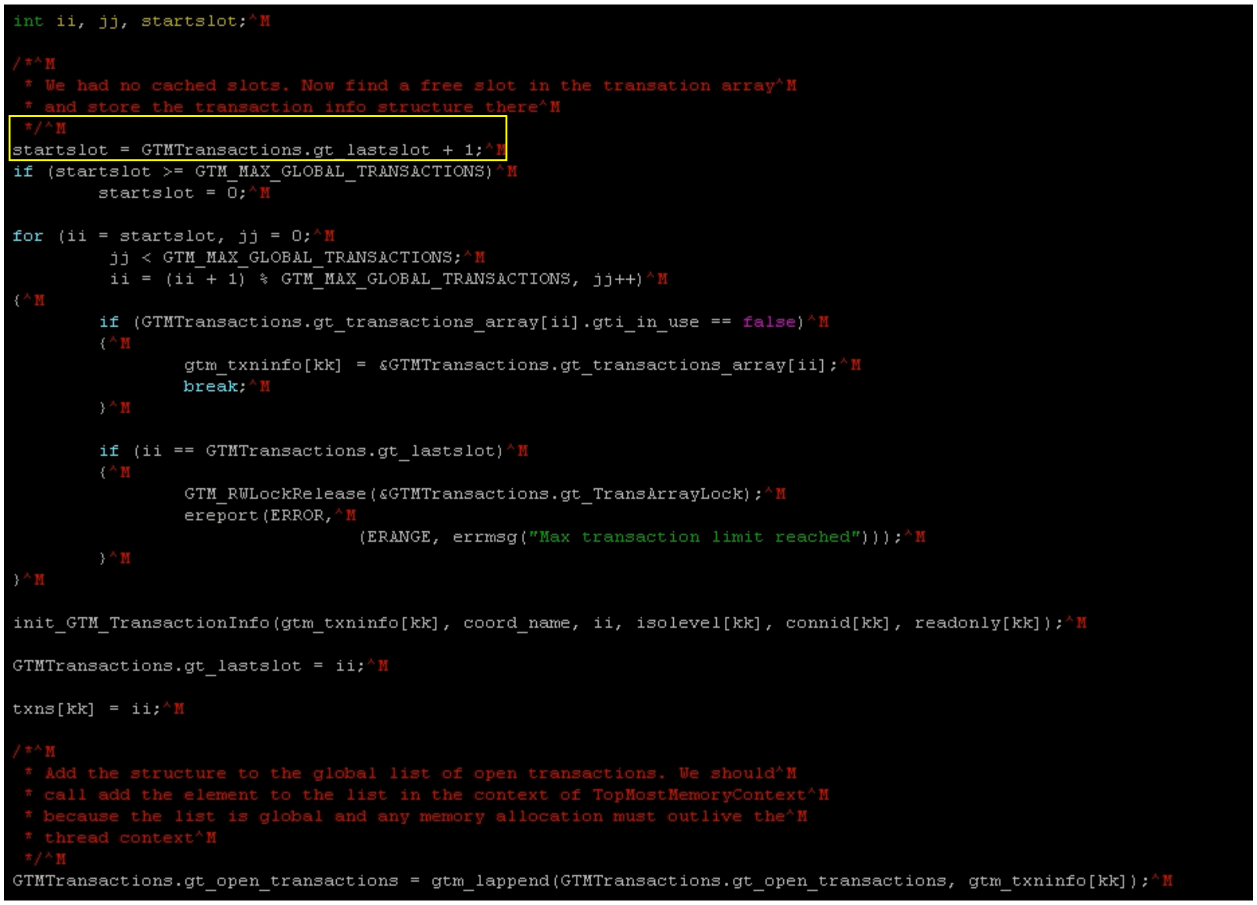
```
typedef struct GTM_TransactionInfo
{
        GTM_TransactionHandle   gti_handle;
        GTM_ThreadID            gti_thread_id;

        bool                    gti_in_use;
        GlobalTransactionId     gti_gxid;
        GTM_TransactionStates   gti_state;
        char                   *gti_coordname;
        GlobalTransactionId     gti_xmin;
        GTM_IsolationLevel      gti_isolevel;
        bool                    gti_readonly;
        GTMProxy_ConnID         gti_backend_id;
        char                   *nodestring; /* List of nodes prepared */
        char                   *gti_gid;

        GTM_SnapshotData        gti_current_snapshot;
        bool                    gti_snapshot_set;

        bool                    gti_vacuum;
        uint32                  gti_timeline;
        GTM_RWLock              gti_lock;
} GTM_TransactionInfo;
```

./src/gtm/main/gtm_txn.cpp

ProcessBeginTransactionGetGXIDCommand()	

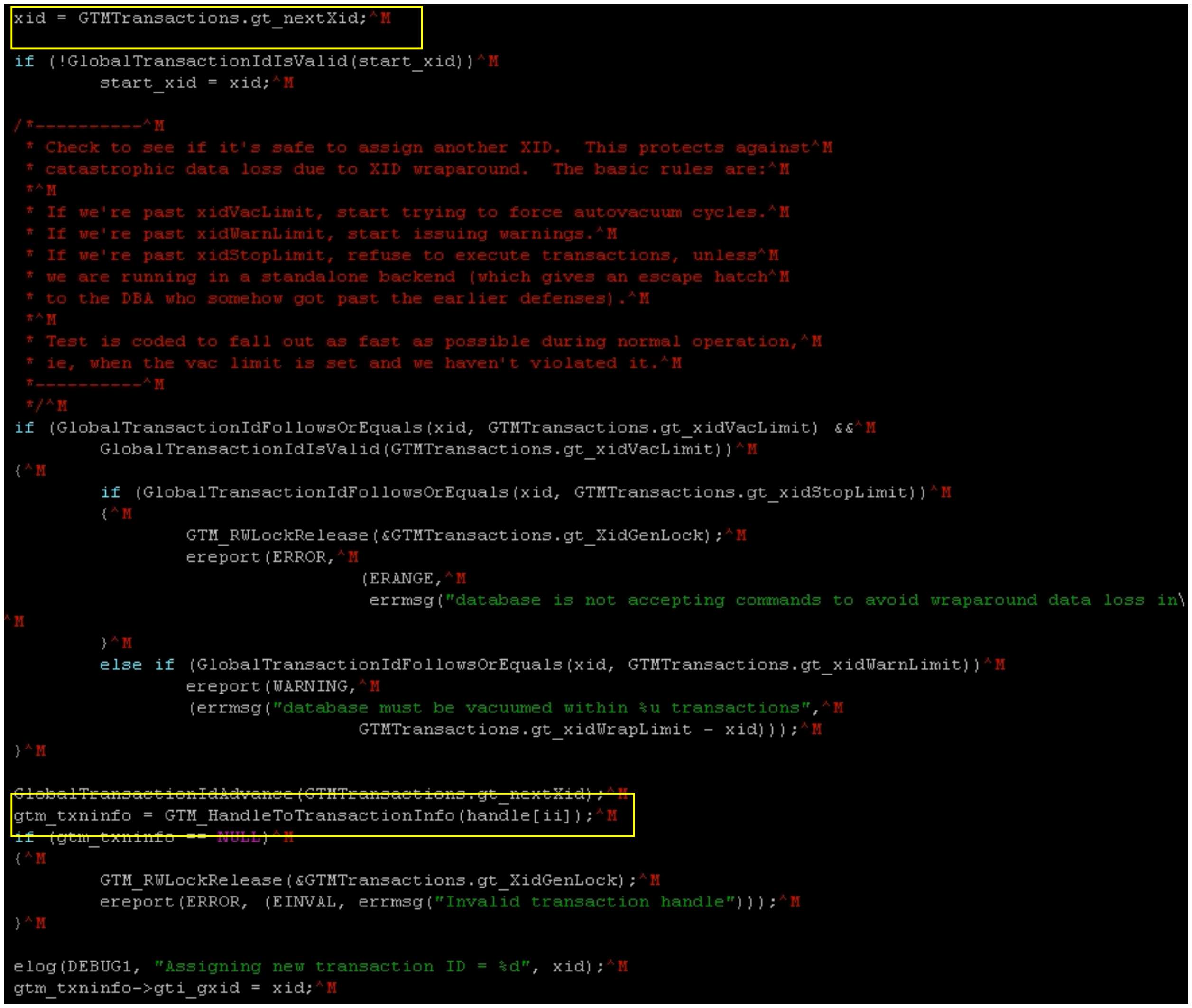
txn = GTM_BeginTransaction() -> GTM_BeginTransactionMulti()
gxid = GTM_GetGlobalTransactionId(txn)
-> GTM_GetGlobalTransactionIdMulti(txn, 1)
-> GTM_HandleToTransactionInfo()	

GTM_BeginTransactionMulti():



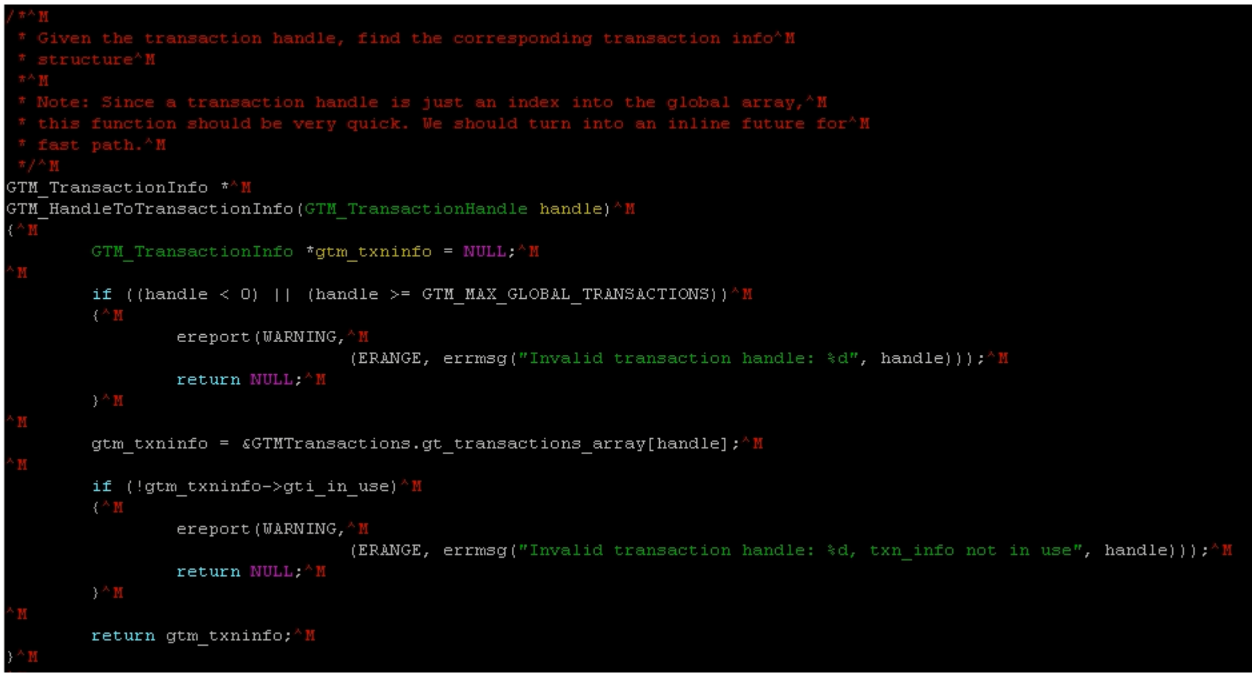
```
int ii, jj, startslot;^M

/*^M
 * We had no cached slots. Now find a free slot in the transation array^M
 * and store the transaction info structure there^M
 */^M
startslot = GTMTransactions.gt_lastslot + 1;^M
if (startslot >= GTM_MAX_GLOBAL_TRANSACTIONS)^M
        startslot = 0;^M

for (ii = startslot, jj = 0;^M
         jj < GTM_MAX_GLOBAL_TRANSACTIONS;^M
         ii = (ii + 1) % GTM_MAX_GLOBAL_TRANSACTIONS, jj++)^M
{^M
        if (GTMTransactions.gt_transactions_array[ii].gti_in_use == false)^M
        {^M
                gtm_txninfo[kk] = &GTMTransactions.gt_transactions_array[ii];^M
                break;^M
        }^M

        if (ii == GTMTransactions.gt_lastslot)^M
        {^M
                GTM_RWLockRelease(&GTMTransactions.gt_TransArrayLock);^M
                ereport(ERROR,^M
                                (ERANGE, errmsg("Max transaction limit reached")));^M
        }^M
}^M

init_GTM_TransactionInfo(gtm_txninfo[kk], coord_name, ii, isolevel[kk], connid[kk], readonly[kk]);^M

GTMTransactions.gt_lastslot = ii;^M

txns[kk] = ii;^M

/*^M
 * Add the structure to the global list of open transactions. We should^M
 * call add the element to the list in the context of TopMostMemoryContext^M
 * because the list is global and any memory allocation must outlive the^M
 * thread context^M
 */^M
GTMTransactions.gt_open_transactions = gtm_lappend(GTMTransactions.gt_open_transactions, gtm_txninfo[kk]);^M
```

GTM_GetGlobalTransactionIdMulti(txn, 1):



```
xid = GTMTransactions.gt_nextXid;^M

if (!GlobalTransactionIdIsValid(start_xid))^M
        start_xid = xid;^M

/*----------^M
 * Check to see if it's safe to assign another XID.  This protects against^M
 * catastrophic data loss due to XID wraparound.  The basic rules are:^M
 *^M
 * If we're past xidVacLimit, start trying to force autovacuum cycles.^M
 * If we're past xidWarnLimit, start issuing warnings.^M
 * If we're past xidStopLimit, refuse to execute transactions, unless^M
 * we are running in a standalone backend (which gives an escape hatch^M
 * to the DBA who somehow got past the earlier defenses).^M
 *^M
 * Test is coded to fall out as fast as possible during normal operation,^M
 * ie, when the vac limit is set and we haven't violated it.^M
 *----------^M
 */^M
if (GlobalTransactionIdFollowsOrEquals(xid, GTMTransactions.gt_xidVacLimit) &&^M
        GlobalTransactionIdIsValid(GTMTransactions.gt_xidVacLimit))^M
{^M
        if (GlobalTransactionIdFollowsOrEquals(xid, GTMTransactions.gt_xidStopLimit))^M
        {^M
                GTM_RWLockRelease(&GTMTransactions.gt_XidGenLock);^M
                ereport(ERROR,^M
                                (ERANGE,^M
                                 errmsg("database is not accepting commands to avoid wraparound data loss in\
^M
        }^M
        else if (GlobalTransactionIdFollowsOrEquals(xid, GTMTransactions.gt_xidWarnLimit))^M
                ereport(WARNING,^M
                (errmsg("database must be vacuumed within %u transactions",^M
                                GTMTransactions.gt_xidWrapLimit - xid)));^M
}^M

GlobalTransactionIdAdvance(GTMTransactions.gt_nextXid);^M
gtm_txninfo = GTM_HandleToTransactionInfo(handle[ii]);^M
if (gtm_txninfo == NULL)^M
{^M
        GTM_RWLockRelease(&GTMTransactions.gt_XidGenLock);^M
        ereport(ERROR, (EINVAL, errmsg("Invalid transaction handle")));^M
}^M

elog(DEBUG1, "Assigning new transaction ID = %d", xid);^M
gtm_txninfo->gti_gxid = xid;^M
```

GTM_HandleToTransactionInfo()


```
/*^M
 * Given the transaction handle, find the corresponding transaction info^M
 * structure^M
 *^M
 * Note: Since a transaction handle is just an index into the global array,^M
 * this function should be very quick. We should turn into an inline future for^M
 * fast path.^M
 */^M
GTM_TransactionInfo *^M
GTM_HandleToTransactionInfo(GTM_TransactionHandle handle)^M
{^M
        GTM_TransactionInfo *gtm_txninfo = NULL;^M
^M
        if ((handle < 0) || (handle >= GTM_MAX_GLOBAL_TRANSACTIONS))^M
        {^M
                ereport(WARNING,^M
                                (ERANGE, errmsg("Invalid transaction handle: %d", handle)));^M
                return NULL;^M
        }^M
^M
        gtm_txninfo = &GTMTransactions.gt_transactions_array[handle];^M
^M
        if (!gtm_txninfo->gti_in_use)^M
        {^M
                ereport(WARNING,^M
                                (ERANGE, errmsg("Invalid transaction handle: %d, txn_info not in use", handle)));^M
                return NULL;^M
        }^M
^M
        return gtm_txninfo;^M
}^M
```