

```

int nSize;
int i;
int nStatus;
String sKey;
GThreadBase *pObject;
static const char *sMsg =
    "%s %iKey[ %s ]%iGroup[ %s ]";
static const char *sErrMsg =
    "%s %iKey[ %s ]%iGroup[ %s ]";
static const char *sInval =
    "Invalid Method Can't Processing [ %d ]";
nSize = 0;
i = 0;
nStatus = 0;
pObject = NULL;
nStatus = this->oMutex.Lock();
ERR_RETURN( nStatus );
nSize = this->oThreadMap.GetElementSize();
for( i = 0 ; i < nSize ; i++ )
{
    nStatus = this->oThreadMap.GetToIndex( i, sKey, pObject );
    if( ERR_CHECK( nStatus ) )
    {
        this->oMutex.Unlock();
        return nStatus;
    }
    if( pObject == NULL )
    {
        this->oMutex.Unlock();
        return RT_F_NULL_POINTER;
    }
    switch( _nExec )
    {
        case EXEC_START_THREAD:
            nStatus = pObject->StartThread( _nArg );
            break;
        case EXEC_STOP_THREAD:
            nStatus = pObject->StopThread( _nArg );
            break;
        case EXEC_ACTIV_COUNT:
            nStatus = pObject->ActivCount( _nArg );
            break;
        case EXEC_THREAD_COUNT:
            nStatus = pObject->ThreadCount( _nArg );
            break;
        case EXEC_CHECK_THREAD:
            nStatus = pObject->CheckThread();
            break;
        case EXEC_MAX_PRIORITY:
            nStatus = pObject->GetMaxPriority( _nArg );
            break;
        case EXEC_THREAD_JOIN:
            nStatus = pObject->Join();
            break;
        default:
            this->oMutex.Unlock();
            RT_LOG_FORMAT_START( RT_E_INVAL )
                , sInval
                , _nExec
            RT_LOG_END;
            return RT_E_INVAL;
            break;
    }
    if( ERR_CHECK( nStatus ) )
    {
        RT_LOG_FORMAT_START( nStatus )
            , sErrMsg
            , sProc[_nExec]
            , sKey.C_Str()
            , this->sGroupName.C_Str()
        RT_LOG_END;
        this->oMutex.Unlock();
        return nStatus;
    }
}
nStatus = this->oMutex.Unlock();
ERR_RETURN( nStatus );
return RT_S_SUCCESS;
}

```