

doc_thread.cpp

```
    memset( &this->oParam, 0x00, sizeof( this->oParam ) );
    this->oParam.sched_priority = _nSetPriority;
#ifdef FOR_DBPS
    this->nThreadID = -1 ;
#endif
    return;
}

int GThread::StartThread(
    int _nInterval
)
{
    int nStatus;
    pthread_attr_t oAttr;
    struct timespec oTime;
    nStatus = 0;
    memset( &oAttr, 0x00, sizeof( oAttr ) );
    memset( &oTime, 0x00, sizeof( oTime ) );
    if( this->pObj->isRunning() == true )
    {
        return RT_S_SUCCESS;
    }
#ifdef FOR_DBPS
    if( this->nThreadID != -1 ) {
        pthread_join( this->nThreadID, NULL );
        pthread_attr_destroy( &oAttr );
    }
#endif
    pthread_attr_init( &oAttr );
    nStatus = pthread_attr_setschedparam( &oAttr, &this->oParam );
    if( nStatus != 0 )
    {
        return RT_E_ERROR;
    }
    nStatus = pthread_attr_setschedpolicy( &oAttr, this->nPolicy );
    if( nStatus != 0 )
    {
        return RT_E_ERROR;
    }
    nStatus = pthread_attr_setscope( &oAttr, this->nScope );
    if( nStatus != 0 )
    {
        return RT_E_ERROR;
    }
    nStatus = this->pObj->StartLoop();
    ERR_RETURN( nStatus );
#ifdef FOR_DBPS
    this->pObj->bRunning = true ;
#endif
    nStatus = pthread_create(
        &( this->nThreadID )
        , &oAttr
        , _runnable
        , this->pObj
    );
    if( nStatus != 0 )
    {
        return RT_E_ERROR;
    }
    if( _nInterval <= 0 )
    {
        return RT_S_SUCCESS;
    }
    while( this->pObj->isRunning() == false )
    {
        oTime.tv_sec = _nInterval / 1000;
        oTime.tv_nsec = ( _nInterval % 1000 ) * 1000 * 1000;
        nanosleep( &oTime, NULL );
    }
    return RT_S_SUCCESS;
}

int GThread::StopThread(
    int _nMode
)
{
    int nStatus;
    nStatus = 0;
    if( this->pObj->isRunning() == false )
    {
        return RT_S_SUCCESS;
    }
    switch( _nMode )
    {
```