

```

    {
        return RT_E_INITIALIZE;
    }
    nStatus = this->Lock();
    ERR_RETURN(nStatus);
    nStatus = this->oQue.Clear();
    if ( ERR_CHECK( nStatus ))
    {
        nStatus = this->Unlock();
        return nStatus;
    }
    nStatus = this->Unlock();
    ERR_RETURN(nStatus);
    return RT_S_SUCCESS;
}

template<class T> DataQue<T> & DataQue<T>::operator =(
    DataQue<T> & _oObj
){
    if ( this->nInitStatus == QUE_INITED )
    {
        this->oQue          = _oObj.oQue;
        this->nMaxQue       = _oObj.nMaxQue;
        this->nTimeout      = _oObj.nTimeout;
    }
    return *this;
}

template <class T>
int DataQue<T>::TimedWait (
    pthread_cond_t *pCond
){
    int    nStatus;
    struct timeval oTv;
    struct timezone oTz;
    struct timespec oTimeout;
    nStatus = 0;
    memset( &oTv, 0x00, sizeof( oTv ));
    memset( &oTz, 0x00, sizeof( oTz ));
    memset( &oTimeout, 0x00, sizeof( oTimeout ));
    nStatus = -1;
    while( nStatus != 0 )
    {
        gettimeofday( &oTv , &oTz );
        oTimeout.tv_sec  = oTv.tv_sec + this->nTimeout;
        oTimeout.tv_nsec = oTv.tv_usec * QUE_TIMEVAL_U_N ;
        nStatus = pthread_cond_timedwait(
            pCond,
            &this->mutex,
            &oTimeout
        );
        switch( nStatus )
        {
            case 0:
                break;
            case ETIMEDOUT:
                return RT_W_TIMEOUT;
            case EINTR:
                continue;
            case EFAULT:
            case EINVAL:
                return RT_E_INVALID;
            default:
                return RT_E_ERROR;
        }
    }
    return RT_S_SUCCESS ;
}

template <class T>
int DataQue<T>::Wait (
    pthread_cond_t *pCond
){
    int    nStatus;
    nStatus = -1;
    while( nStatus != 0 )
    {
        nStatus = pthread_cond_wait(
            pCond,
            &this->mutex
        );
        switch( nStatus )
        {
            case 0:
                break;

```