

doc.cpp

```
template <class T> int DataQue<T>::Init(
    void
){
    int nStatus;
    nStatus = 0;
    nStatus = pthread_mutex_init ( &( this->mutex ), NULL );
    if ( nStatus != 0 )
    {
        return RT_E_INITIALIZE;
    }
    nStatus = pthread_cond_init ( &( this->PushCond ), NULL );
    if ( nStatus != 0 )
    {
        return RT_E_INITIALIZE;
    }
    nStatus = pthread_cond_init ( &( this->PopCond ), NULL );
    if ( nStatus != 0 )
    {
        return RT_E_INITIALIZE;
    }
    nStatus = this->oQue.Clear();
    ERR_RETURN(nStatus);
    this->nInitStatus = QUE_INITED;
    return RT_S_SUCCESS;
}

template <class T> int DataQue<T>::Push (
    T _oData
){
    int nStatus;
    int nQueSize;
    nStatus = 0;
    nQueSize = 0;
    if ( this->nInitStatus != QUE_INITED )
    {
        return RT_E_INITIALIZE;
    }
    nStatus = this->Lock();
    ERR_RETURN(nStatus);
    while ( ( nQueSize = this->oQue.GetElementSize() ) >= this->nMaxQue )
    {
        if ( this->nTimeout == 0 )
        {
            nStatus = this->Wait( &this->PopCond );
        }
        else if( this->nTimeout > 0 )
        {
            nStatus = this->TimedWait( &this->PopCond );
        }
        if( nStatus != RT_S_SUCCESS )
        {
            this->Unlock();
            return nStatus;
        }
    }
    nStatus = this->oQue.PushFirst( _oData );
    if ( ERR_CHECK( nStatus ) )
    {
        this->Unlock();
        return nStatus;
    }
    nStatus = this->Unlock();
    ERR_RETURN(nStatus);
    nStatus = pthread_cond_broadcast( &( this->PushCond ) );
    if ( nStatus != 0 )
    {
        return RT_E_INVALID;
    }
    return RT_S_SUCCESS;
}

template <class T> int DataQue<T>::Pop (
    T & _pData
){
    int nStatus;
    int nQueSize;
    nStatus = 0;
    nQueSize = 0;
    if ( this->nInitStatus != QUE_INITED )
    {
        return RT_E_INITIALIZE;
    }
    nStatus = this->Lock();
    ERR_RETURN(nStatus);
```