SMARTSOFT

ONLINE YAZILIMCI EĞİTİM DÖKÜMANI - 1

GENEL KONULAR

Contents

GİRİŞ

Kapsam

Bu döküman Smartsoft Online konseptine ve online kodlamaya başlangıç dökümanıdır. Hedef kitle teknik personeldir. Yeni başlayanlar için hazırlanmış olmakla birlikte, anlatılan bazı püf noktaları ve sorunlara müdahale etme yöntemleri daha ileri seviyedeki kişilere de faydalı olabilir.

ONLINE nedir?

Smartsoft olarak genellikle sürekli ayakta olan, mesaj bekleyen ve mesajları işleyip cevap dönen programlara online diyoruz. Öte yandan web service'ler de bu tanıma uyuyor denilebilir. Kelime anlamı olarak bakılırsa, Batch olmayan tüm uygulamalara online denmelidir. Belki fark olarak online uygulamalarda cevap süresi ve availibilty'nin önemli olduğu söylenebilir. Tabii her online program da kritiklik seviyesi bir değil. Tek işi sahadan gelen verileri batch tarzda kaydetmek olan bir online program da olabilir. Online uygulamaların önemli bir özelliği de – istisnaları olmakla birlikte – bağlantılarının sürekli ayakta olması ve aynı bağlantı üzerinden asenkron olarak mesaj gönderilip alınabilmesidir.

Kimi zaman C++ ile yazılmış tüm kodlara online diyenler de olmakta. Bu görüş ilk başta garip görünse de kendi içinde tutarlıdır. Nihai tanım olarak, dış sistemlerle sürekli bağlantıda olan, real-time'a yakın, kritik, user interaction gerektirmeyen uygulamalardır diyebiliriz.

Neden C++ (yada C)

Online uygulamalarda C / C++ kullanıyoruz. Aslında tam olarak C++ dememiz haksızlık olur. Pek çok C++ konseptiyle alakamız yoktur. C ile C++'ın bir karışımı da diyebiliriz. Bu diller gerek performans gerek taşınabilirlik açısından tercih edilmekte. Zaten işin başında "pratik olarak" başka bir alternatifimiz yoktu. C/C++ online uygulamalar için genel olarak kabul görmüş bir dildir. Bazen dikkatsizlik sonucu bu dillerle sorunlar yaşanabiliyor ama böyle kritik uygulamalar yazılırken dikkatli olmak gerekir.

Öte yandan iyi yazılmış bir c# programının da bu işi görebileceğine söylenmekte. Argüman olarak Server kapasitelerinin artmasıyla ufak miktardaki performans farklarının önemsiz olduğu dile getirilmekte. Bu konuyu incelemek ayrı bir çalışma konusu olabilir. Eğer c# uygulamalarının c++ programlarından tek farkı, aynı işi daha fazla cpu cycle ile yapmak ise, cpu kapasitelerini artırarak c# ile online program yazmak mümkündür denilebilir. Ancak internal olarak alt planda neler yapıldığı irdelenmelidir.

Başlamadan önce bilinmesi gerekenler

Online yazmaya başlamadan önce ne bilmek gerekir? Aşağıda gerekli olabilecek bilgiler kısmen listenmiştir. Bilgiler zamanla öğrenilebilir, yeni başlayanların özellikle business bilgisi olmaması normaldir. Ancak ne kadar çok bilgi ve pratik varsa o kadar rahat bir başlangıç

olacaktır. En azından kişide algoritmik, analitik zeka olmasında fayda vardır. Genel olarak problem çözbilme, mühendis bakış açısıyla meselelere bakabilme yeteneği de işe yarar. Gerçi sadece online programlar için değil, tüm ciddi yazılım projeleri için bunlara ihtiyaç duyulabilir. Ayrıca yeri geldiğinde eksik analizlerin ve kullanıcı testlerinin de yazılımcı tarafından yapılması gerekebilmektedir.

Yoğun olarak ihtiyaç duyduğumuz / kullllandığımız bilgileri şu şekilde gruplayabiliriz:

- C / C++ bilgisi : syntax ve keyword öğrenmek kısa sürer. Ancak bunları düzgün ve yerinde kullanabilmek zamanla pratik yaparak olur. Diğer dillerden geçiliyorsa dikkatli olunmalıdır. C dili siz ne derseniz onu yapar, ne fazla ne de eksik. Pointer kullanımında dikkatli olmak gerekir. Karakter array'lerinin string gibi çalışmadığının farkında olmak gerekir (mesela). Gerçi C++ string yapısı da var ama biz kullanmıyoruz.
  Smartsoft online uygulamaların genel bir structure'ı vardır. Bu sınırlar içinde kaldığınızda hata yapma şansınızı da azaltmış olursunuz. Zaten yapılan çoğu iş artık standart hale gelmiştir.
- Sistem Bilgisi : Alt seviyeye inilecekse bazı kavramları bilmek ve anlamak gerekir. Process, Thread, Mutex, Event, Memory Management, TCP/IP , senkron/asenkron haberleşme ...
  Ancak çöğu zaman bunlarla bir işimiz olmuyor. Nadiren library seviyesinde düzeltmeler yapılsa da çoğunlukla işimiz business kısmındadır.
  Tabii bu kavramlarla hergün uğraşmasak bile bilmekte fayda vardır. Örneğin TCP/IP'nin nasıl çalıştığını bilirsek, bir 3rd party ile tcp konuşurken ortaya çıkabilecek sorunları anlamamız kolaylaşabilir.
- Business bilgisi : Business çok geniş bir kavram. Tüm business kavramları bilmek yıllar alacaktır. Tabii bu aşamaya geldikten sonra çoğunlukla kodlama yapılmıyor. Öte yandan tüm business'i bilmeye gerek de olmayabilir. Örneğin pos ile uğraşan bir kişi pos kısmını çok iyi bilmeli. Bazen diğer kısımların bilinmesi fayda getirebilir. Business bilgisi de kendi içinde bölümlere ayrılabilir. Bazı international standartlar da business bilgisi kapsamına girebilir, bir bankanın kendine özel ekstre yapısı da.

Kodlamaya Başlangıç (yazılımcı kurulum ve ayarlar)

Yeni bir bilgisayarda online kodlamaya başlayabilmek için şu adımlar izlenmelidir.

➔ Visual studio kurulur ve ayarlanır.
Sadece C++ ile kod geliştirecekseniz kurulum esnasında Environment seçiminde C++ seçmenizde fayda var. bazı şeyleri daha kolay yaparsınız.

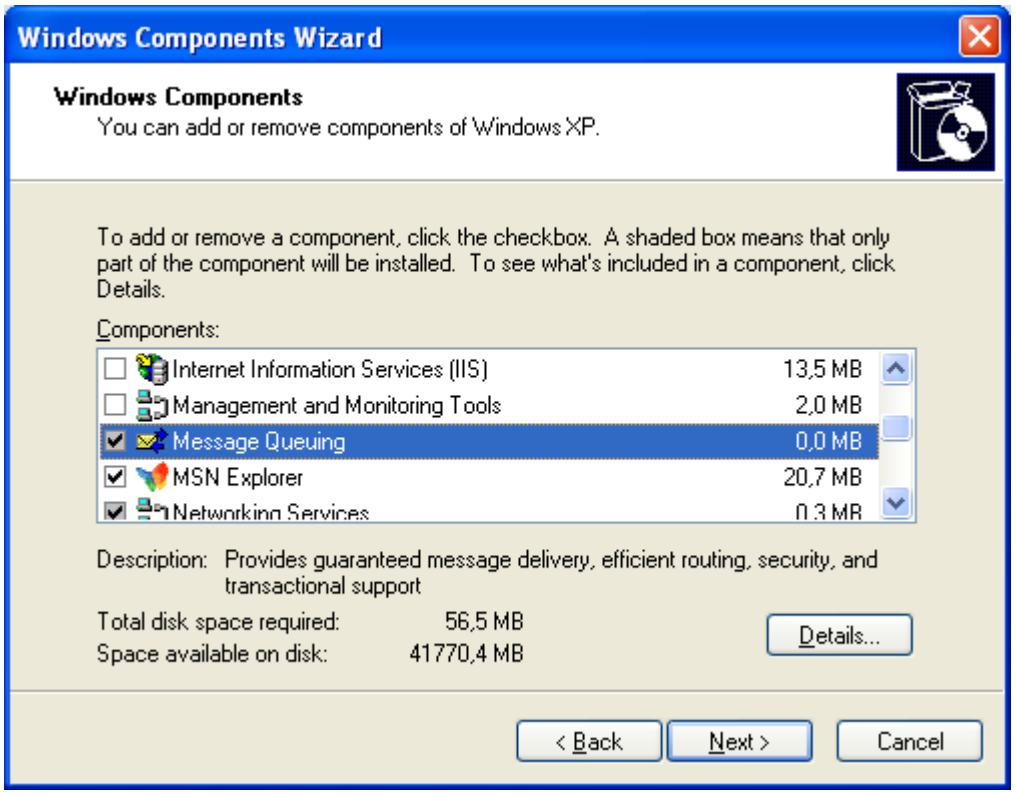
Bir de benim kişisel favotim olarak şunları da yapabilirsiniz. “Tools” – “Customize” – “KeyBoard” ekranından :

- Keyboard shortcut’larından view.PopBrowseContext -> (ctrl + *) olarak set edin. Aslında bu zaten set edilmiş ama keypaddeki * karakteri atanmış. Tabii laptoplarla çalışınca keypad karakterlerine erişim biraz sıkıntılı oluyor. Bunu normal * karakterine ile tekrar tanımlarsınız. Ne işe yarar : F12 ile gittiğiniz fonksiyondan ctrl+* ile kolayca dönersiniz. Alışkanlık haline getirdiğinizde çok işe yarar.
- Keyboard shortcut’larından view.FindResults1 -> ctrl+2 yapın. view.output -> alt+2 yapın. Zamanla bu tuşları yerinde kullanırsanız çok işe yarayacak.

➔ Online kodlar local’e alınır.
Projeler incelendiğinde görülecektirki bankaya ait tüm online kodlar “Online” isimli bir dizin alıtanda yer alır. Başka bir dizine bağlantımız yoktur (external libraryler hariç). Projeler birbirinden bağımsızdır, sadece Olbase isimli ortak kullanılan bir kütüphane vardır.

➔ Online kodları derlemek yada çalıştırmak için makinanıza msmq kurmanız gerekebilir. “Add/Remove Programs” – “Windows Components” kısmından kurulabilir. (Msmq kurmadan derlenebiliyor olabilir). Programı çalıştırmak yada debug etmek için msmq kesinlikle gerekiyor. Aksi halde açılışta dll bulunamadı hatası alabilir yada göçebilir.

Windows XP, Windows 2003:
Control Panel → Add Remove Programs → Add/Remove Windows Components Altından Message Queuing bulunup eklenir. Sadece “Common” parçası eklense yeterli olur.



Windows 2008
Server Manager → Features → Add Feature altından Message Queuing eklenir.

➔ Online kodları derlemek için DB client kurulumu gerekebilir.
Projeler Oracle ile derlenecekse, OCI library'si olmalıdır (oracle client kurulumu yapılır). Sql Server ile derlenecekse Sql Server client kurulumu gerekir. Çoğu bankada oracle kullanıldığı için bu dökümanda oracle ile devam edeceğiz. Oracle kurduğunuzda OCI dizininin oluştuğundan emin olun.
OCI dizini başka bir makinadan kopyalanarak da derleme yapılabilir ancak programı çalıştırmada sorun yaşanabilir.

➔ Oracle OCI için directory'lerin set edilmesi. Oracle client kurmak tek başına yetmeyecek. Projelerin Oracle OCI header ve library'leri alması için biraz ayar yapmak lazım. Directories ayarlarına girip, include ve library dizinlerini değiştirmek gerekiyor.

VS2005 : Tools → Options → Projects And Solutions → VC++ Directories
Include directory -> C:\Oracle\product\11.1.0\client_1\oci\include
Library directory -> C:\Oracle\product\11.1.0\client_1\oci\lib\msvc
(Oracle'in kurulduğu path'e göre oci dizininden öncesi değişebilir)

VS2010 için directory set etmede biraz fark var. aşağıdaki dizine gidip dosyalara ilgili dizinleri ekleyin. Böylece her proje için uğraşmanıza gerek kalmaz.

“C:\Documents and Settings\yusuf\Local Settings\Application Data\Microsoft\MSBuild\v4.0”
Microsoft.Cpp.Win32.user.props -> 32 bit derleme için dizinler
Microsoft.Cpp.x64.user.props -> x64 bit derleme için dizinler
Microsoft.Cpp.Itanium.user.props -> itanium derleme için dizinler

Buraya kadar olan kurulumları başarıyla geçtiğinize göre artık bir online programı derlemeyi deneyebilirsiniz. Bir şey bilmenize gerek yok sadece derlendiğini ve çalıştığını görmeniz yeterli.

Öncelikle Olbase projesini açın ve rebuild edin. Ardından herhangi bir projeyi açıp rebuild edin ve F5 ile çalıştırın. Program hemen açılıp kapanacaktır ama command promt ekranına kadar geldiyseniz kurulumunuz tamam demektir.

PROJE YAPISI

Tipik olarak bir bankanın online projelerini açtığımızda şöyle bir yapı ile kaşılarız.

..\ Online\OCAuth\
..\ Online\OCGateATM\
..\ Online\OCGatePOS\
..\ Online\OCHsmd
..\ Online\OCLogger
..\ Online\OCMonCore
..\ Online\OCTask
..\ Online\OLBase

Bankanın kullandığı online modüllere bağlı olarak buradaki dizin sayısı artabilir yada azalabilir. Her bir dizin içinde bir proje bulunur ve sonuç olarak bir exe dosyası üretilir (Olbase projesi hariç, Olbase projesi bir lib dosyası üretir).

Olbase Library

Olbase, statik library olarak tüm diğer projeler tarafından kullanılır. Olbase Library'si programlarda kullanılabilecek genel fonksiyonaliteyi sunar. Çok kullanılan obje ve fonksiyonlar burada toplanmıştır. Ayrıca işletim sistemi fonksiyonlarına yada external library'lere mümkün olduğunca buradan erişilir. Uygulamalar mümkün olduğunca platformdan izole edilmiştir. Teorik olarak olbase'in değiştirilmesiyle platformlar arası taşıma mümkün olmalıdır.

Projeler Olbase library'sini şu şekilde kullanır.

a. Projeler Olbase projesindeki header dosyaları ile compile edilir. Tüm header dosyalarını ayrı ayrı eklemek yerine sadece "OLBaseFw.h" isimli dosya eklenir. Bu header dosyası da Olbase içindeki tüm diğer header'ları include etmektedir.

   ```
   #include "..\\..\\OLBase\\OLBase\\OLBaseFw.h"
   ```

   Projelerin içinde yukarıdaki gibi bir include ile olbase header'ları eklenmiş olur. En baştaki dizin yapısına göre düşünülürse relative adres verilmektedir.
   Yukarıdaki include stdafx.h dosyası içine eklenerek tüm projede kullanılması sağlanmış olur.

b. Olbase derlendiğinde “OLBaseFw.lib” dosyası oluşur. Projelerin Link aşamasında da Olbase library’si linker’a parametre olarak verilmektedir

```
#ifdef _DEBUG
#pragma comment(lib,
"..\\..\\OLBase\\OLBase\\Debug\\OLBaseFw")
#else
#pragma comment(lib,
"..\\..\\OLBase\\OLBase\\Release\\OLBaseFw")
#endif
```

Projeler sadece bu satırları eklemek suretiyle olbase library’sini kullanabilir. Projenini debug/release derlenmesine göre ilgili OLBaseFw.lib dosyası eklenmektedir.

Olbase projesinde çok değişiklik yapılmaz. Herhangi bir değişiklik yapıldığında bu proje tüm config’ler için baştan rebuild edilmelidir. Header ile lib dosyasının içeriklerinin farklı olması durumunda istenmeyen sonuçlara yol açabilir.

Olbase neleri içerir

- Auxiliary fonksiyonlar
  - Dönüşüm fonksiyonları (hex, bcd, ascii, ebcdic, binary,integer, base64, base94)
  - Date / Time fonksiyonları
  - Floating point yuvarlama
  - CRC hesaplama
  - Data compression algoritması
  - Config reader
  - Crypto library (DES, DES3, PKCS5, ECB, CBC,)
- İşletim sistemi fonksiyonları
  - Thread, Thread Pool
  - Windows Service
  - Mutex
  - Event
  - Config / registry
  - File I/O
  - Shell process başlatma
- Db erişimi
  - OCI -> Oracle
  - OleDB -> SQL Server
- Mesaj çözme ve oluşturma
  - ISO8583 (Pos, BKM,Visa BASEI, MC online, IPM ... parametrik)

    - TLV mesaj (parametrik)
- Netwrok library'leri
    - TCP / IP
    - TCP / IP Connectivity objects
    - Seri Port
    - MSMQ
    - IBM MQ
    - Custom Send/Receive Protocol Over MSMQ (or TCP/IP)
    - Network monitoring
- Log library
- Tanımlamalar
    - Bankaya özel sabitler ve structure'lar
    - BKM / VISA / MC sabitleri
    - Data Tip tanımalamarı : Standart tipler typedef ile tekrar tanımlanıştır.
    - Makrolar

Olbase bir fonksiyon kütüphanesi gibi düşünülebilir. Nesne ve fonksiyonların ne işe yaradığı Olbase içindeki header dosyalarında anlatılmıştır. Buradaki commentler doxygen formatında olup ayrıca html olarak bir output da üretilebilir. Ancak olbase'in nasıl kullanılacağı bu commentlerden çok anlaşılamayabilir. Bunlar referans olarak düşünülmelidir.

Örnek Olbase fonksiyon comment stili:

```
/// \brief Compares two double value to check they are equal.
/// \param[in] d1 First value to compare.
/// \param[in] d2 Second value to compare.
/// \return If first number is equal to second number, returns (#B1), otherwise (#B0).
       OSI4 IsEqu(const ODBL d1, const ODBL d2);
```

Olbase objelerinin kullanım ayrıntıları bu dökümanda ayrı bir bölüm olarak eklenmiştir.

Uygulama Projeleri

Proje listesinde Olbase dışında kalan her bir dizin bir uygulamaya ait projeyi içerir. Genel olarak uygulama ismi ne işe yaradığını da göstermektedir. Örneğin “OCGatePos” sahadaki fiziksel pos terminallerini karşılayan uygulamadır. Uygulamarı 3 gruba ayırabiliriz.

1. Gate Uygulamaları : Bunlar dış dünya ile konuşan uygulamalardır. Her bir bağlantı noktası için bir gate uygulaması oluşturulur. Genel olarak dış dünya ile TCP/IP üzerinden konuşulur. Ancak MSMQ, IBMMQ ile de mesaj alıp göndermesi mümkündür. Örnek :
   Pos Gate
   Vpos Gate
   ATM Gate
   BKM Gate
   VISA Gate
   Mastercard Gate
   Core Bank Gate
   Şube Gate
   ..

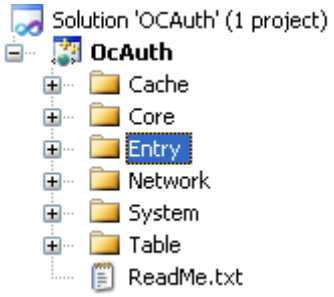
2. Authorisation Uygulamaları : Karta onay veren uygulamalardır.
   OcAuth : Kredi Kartı otorizasyon
   OCDAuth : Debit Kart otorizasyon
   PRAuth : Prepaid authorisation

3. Host Uygulamaları : Diğer uygulamaların aldıkları servisleri ifade eder.
   OCLogger : Merkezi Loglama Uygulaması
   OCMonCore : Merkezi Monitoring Uygulaması
   OCHSMD : Thales HSM Server
   OCHSMFc : Safenet HSM Server
   OCTask : Task Server (scheduler)

Sahada kurulumlarda sadece Gate/Host olarak ayrıştırılmıştır. Authorisation uygulamaları da gate tipinde kurulmaktadır. Ama bunun kafa karıştırmasına gerek yok. Zamanla Gate ve host kavramları birbirine karıştığı için bu şekilde oldu.

Uygulama Projelerinin Yapısı

Genel olarak tüm uygulama projelerinde source kodlar benzer bir hiyerarşi içindedir.



Solution içinde aşağıdaki filtrelere göre dosyalar gruplanmıştır:

Cache : Bazı tabloların periyodik olarak hafızaya alınması ve değerlerin buradan getirilmesi için kullanılır. Tabloların nasıl cache'lendiği ve ileride nasıl ulaşıldığı anlatılacak. Cache altyapısı programlar için ortak olmakla birlikte cache'lenen tabloların listesi farklı olacaktır.

Core : Business objesini içerir. **Programdan programa değişen kısımdır.** Diğer bölümler neredeyse tüm uygulamalarda aynıdır.

Entry : bu bölümde şu dosyalar bulunur:

- Stdafx.h stdafx.cpp : precompiler header. Olbase ve çok değişmeyen headerların her seferinde baştan derlenmesini önler.
- OC"ProgramName".cpp : main fonksiyonu vardır sadece.
- OC"ProgramName".h : programda kullanılan global tanımlar ve include'lar.

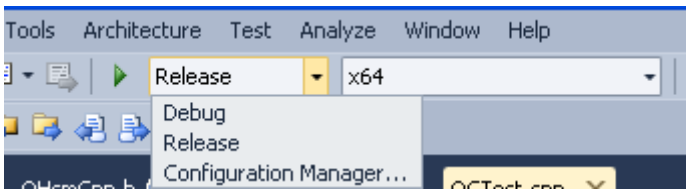
Network : Programların birbirleri arasında mesaj alışverişini sağlayan BMsgDsp nesnesini içerir. Bu nesne programlar arasındaki path tanımlarına göre mesajın doğru programa gitmesini sağlar. Programlar msmq ile konuştuğu için karşıdaki programın gerçekten açık olup olmadığını anlayan mekanizmalar da vardır. BMsgDsp objesi aynı zamanda yük dağılımı ve priority gibi işleri de halleder.

System : Programın altyapısı bu nesnelerin içindedir. Programın başlangıcında thread'lerin ayağa kaldırılması, initializastion işlemleri, programın düzgün kapanması, servis olarak çalıştırma, Config okuma, thread yönetimi gibi işler buradadır. Yapısı çok değişmemekle birlikte projenin içeriğine göre yeni thread'ler tanımlanıp buradan başlatılması gerekebilir. Yada yeni bir host uygulaması ile senkron haberleşmek için buraya kodlar eklenebilir.

Table : Tablolar için üretilmiş kodlardır. Erişilen her tablo için bir nesne bulunur. Bu kodlar "ORMapper" uygulamasıyla otomatik olarak üretilir. Daha sonra query'lerde yada parametrelerde elle değişiklikler yapılabilmektedir. OrMapper sadece basit query'ler ürettiğinden genelde değişiklik yapılır.

Debug vs Release derleme

Projeleri derlerken tool'ların arasında görüldüğü gibi debug & Release derleme seçmi yapılır. Bazı projelerde Win32/X64 seçimi de yapılmaktadır. 64 bit konuları daha sonra anlatılmıştır.



Bazı Visual studio kurulumlarında yukarıdaki gibi bir seçim toolbar da gelmiyor. Sanırım C++ olarak kurmaya bağlı olarak geliyor bunlar. Bu durumda da başka şekillerde de olsa debug release geçişi yapılabiliyor.

Debug : Bu modda sadece geliştirme aşamasında derliyoruz. Trace etme ve değişkenlkerin içeriğini görme daha kolaydır. Programda buffer / değişken ezilmesi gibi durumları tespit etmek debug modda daha kolaydır.

Release : Production yada test ortamlarına exe verirken release olarak derliyoruz. Test ortamında az işlem olduğundan debug versek olmazmı diye bir soru gelebilir. Release modda ortaya çıkabilecek aksilikleri yakalayabilmek için test ortamı da dahil release verilmeli. Bazen debug modda soruna yol açmayan bir geliştirme release modda göçebilmekte. Test ortamında binary olarak da prod ile aynı olmakta fayda var.

Release modda iken trace etmek mümkün fakat biraz zahmetli olabilir. Çünkü değişken içeriklerini göstemeyebilir yada hatalı gösterebilir. Satır atlatma gibi fonksiyonlar düzgün çalışmayabilir. Bunların sebebi release modda yapılan optimizasyonlar sonucu bazı değişken ve satırların hiç derlenmemesi gibi sebeplerdir.

Release mod çok daha hızlı olduğu için prod ortamına release veriyoruz. Özellikle std librarysi içindeki fonksiyonların debug versiyonları performans açısından çok kötüdür.

Kullanılan olbase library'sinin de rebuild edildiğinden emin olunmalıdır. Genellikle Olbase içinde bir değişiklik yapıldığında, hem debug hem release olarak "rebuild all" yapılmalıdır. Bazen olbase debug modda derlenip uygulamalar düzgün olarak debug modda test edilmekte, ancak uygulama release olarak derlenirken eski olbase lib dosyası kalabilmektedir.

Yeni Uygulama Projesi oluşturulması

Çoğunlukla yeni proje oluşturmak yerine mevcut projeler üzerinde geliştirme yapılıyor. Yeni bir projeye başlandığında bile mevcut projelerin en yakınından kodlar alınarak üzerinden devam ediliyor. Fakat proje guid'inin çakışması yada elle editlemek istenmezse, boş bir proje oluşturulup, diğer projeden gerekli kod dosyalarının kopyalanması daha mantıklı olabilir.

Yeni bir proje oluşturulmasında şu adımlar izlenebilir:

- Proje tipi olarak, Visual C++ → Win32 Console Application seçilir. Oluşturulacak dizin seçilir.
- Precompiled Header seçeneği seçilir. Az dosyalı utility'ler için seçilmeyebilir
- Menüden : Project → Properties ekranına girilir. C/C++ / Code Generation altında "Runtime Library" seçeneği değiştirilir.
  Debug için : Multi-threaded Debug (/MTd)
  Release için : Multi-threaded (/MT) seçilir.
  Bunlar seçilmezse olbase ile uyumsuzluk olacağından derlenmez.
- Olbase header include edilir. Projenin bulunduğu path'e göre relative path değişebilir. Genellikle stdafx.h içinde include ediyoruz. Ancak herhangi bir yerden include edilebilir:

```
#include "..\\..\\OLBase\\OLBase\\OLBaseFw.h"
```

- Olbase library linkera verilir. Bu işi genllikle stdafx.cpp içinde yapıyoruz. Herhangi bir cpp içinde yada project properties'den additional dependencies kısmından verilebilir.

```
#ifdef _DEBUG
#pragma comment(lib, "..\\..\\OLBase\\OLBase\\Debug\\OLBaseFw")
#else
#pragma comment(lib, "..\\..\\OLBase\\OLBase\\Release\\OLBaseFw")
#endif
```

- Proje içinde dizin yapısı oluşturulur (Cache, Core, Entry, System, Table ..)
- Online servis ise mutlaka gerekli olan dosyalar başka bir projeden eklenir
  - System\ASysCfg.cpp, .h
  - System\ASysCmd.cpp, .h
  - System\ASysMgr.cpp, .h
  - System\ASysWrkAdm.cpp, .h
  - System\ASysWrkCch.cpp, .h
  - System\ASysWrkDsp.cpp, .h
  - System\ASysWrkTxn.cpp, .h
  - System\ASysGlb.h
  - Cache\ACchPrm.cpp, .h
  - Cache\ACchPrmFill.cpp
  - Cache\ACchPrmFind.cpp
  - Core\BMsgCfg.cpp
  - Core\BMsgClr.cpp
  - Core\BMsgDbm.cpp
  - Core\BMsgPrc.cpp
  - Core\BMsgPrc.h
  - Core\....

- Table\Sys\F_SYS_SEQ.cpp, .h
- Table\Sys\T_SYS_DUAL.cpp, .h
- Table\Sys\T_SYS_GATE.cpp, .h
- Table\Sys\T_SYS_HOST.cpp, .h
- Table\Sys\T_SYS_PATH.cpp, .h
- Table\Sys\T_SYS_PRM_ONLINE.cpp, .h
- Table\....

"baz alınabiliecek boş bir proje yapalım - TODO"

Yukarıdaki liste genel olarak her projede olabilecek dosyaları içeriyor. Projenin yapısını göre çok sayıda db yada business kodu kopyalanabilir.

- Bazı bankalar için, Global objeler içinde mutlaka oLogSys objesi bulunmalıdır. Loglama yapılmasa bile Olbase içinde bu objeye external olarak refere eden kodlar olabiliyor. Ancak yeni yapılan projelerde bundan kurtuluyoruz genellikle. Olbase oLogSys objesi kullanmıyor

```
OLogSys oLogSys;
```

- Gerekli dosyalar kopyalansa bile epey bir temizlik yapmak gerekebilir. Gereksiz ve zahmetli bir prosedur maalesef. Daha şık yöntemler tasarlanabilir.

Örnek bir online kod bloğunun incelenmesi

Ileride geniş kullanımları verilecek olmakla birlikte, burada basitçe bir online kodunun neye benzediği gösterilecektir. Bazı çok kullanılan yapılara da giriş yapılmış olacaktır:

```
OSI4 BMsgPrc::CheckCardInfo(OLGC fiCheckFlag)
{
    OSI4 iRet = ORC_ERR;

    try
    {
        // find mbr
        iRet = FindIssMbrId();
        if (iRet != ORC_OKI)
            IRCR("B961101", "FindIssMbrId error.", ORC_ERR);
```

Yukarıda bir fonksiyonun belli bir kısmı listelenmiştir. Buna bakarak şu bilgileri verilebilir.

OSI4 : ocean'da primitive tipler tekrar tanımlanmıştır. OSI = "Ocean Signed Integer", en sondaki rakam ise 4 byte'lık bir integer olduğunu gösterir. Windows makinada "int" ile aynı anlama gelmektedir. Genel olarak şu tipler sıkça kullanılmaktadır:

OSI8, OSI4, OSI2, OSI1 -> integer. Sonunda length vardır.

OUI8, OUI4, OUI2, OUI1 -> unsigned integer. Sonunda length vardır. Bunlar integer kadar çok kullanılmaz. Genelde ihtiyaç duyulmuyor.

OSC -> karakter. Genelde karakter array olarak kullanılır.

OUC -> byte. Genelde byte array olarak kullanılır.

ODBL -> floating point number. Tutarlar da bu tipte tutulur

OLGC -> true / false anlamında kullanılır. B1 , B0 makrolarını değer vermekte fayda var.

BMsgPrc : Core business class. Tüm işlem akışı bu class içindedir. Her worker thread bu class'in bir instance'ı ile çalışır.

ORC_ERR : Fonksiyonlar genel olarak 3 değerden birini döner :

ORC_OKI : başarılı

ORC_ERR : başarısız

ORC_NON : data bulunamadı.

ORC_NON değişkeni bir şeyi arayan yada getiren fonksiyonlar için anlamlıdır. Çoğu fonksiyon sadece ORC_ERR yada ORC_OKI çiftinden birini de dönebilir. Ancak bunlar kural değildir. Bir fonksiyon buradan çok farklı değerleri de dönebilir ihtiyaca göre.

Try : işin aslı exception handling kullanmıyoruz. Bu try catch blokları hiç bir işe yaramıyor. Biz birşey throw etmediğimiz için. Visual studio ayarları ile bazı access violiation alınan durumlarda buralara düşebiliyor. Ama bunu tercih etmiyoruz. Büyük ihtimalle o durumda devam etmek daha kötü sonuçlara yol açar.

IRCR : Programlarda sıkça kullandığımız bir makrodur. Buna benzer pek çok makro bulunmakta. Bu makro kısaca şu işleri yapar:

- Loglara bir satır error mesajı basar
- Response code set eder
- Fonksiyondan ORC_ERR döner

IRCR ile fonksiyondan çıkılmış ve bir üst fonksiyona düşülmüş olur. Bir üst fonksiyonda da muhtemelen buna benzer bir kontrol vardır ve ana fonksiyona kadar hiyerarşi içinde error dönülmüş olur. Bu haliyle exception throw etmeye benzetilebilir belki.

Benzer şekilde __ERRB, __ERRD, __ERRx, IRCI, _ERRB gibi makrolar daha sonra anlatılacaktır. Tüm bu makrolar log satırına bir satır log atar ama farkları vardır.

B961101: Verilen hatanın uygulamanın hangi satırından verildiğini anlamamıza yarayan IRC değeri (Internal Response Code). Bu değerler de otomatik olarak üretilir. IRC değerleri tablolarda da saklanarak hangi kontrolden red dönüldüğü kolayca anlaşılabilir. Ayrıca IRC değerine göre dönlecek cevap kodu ve alın alınabilecek aksiyonlari OC_SYS.SYS_SEC tablosunda tanımlanmıştır.

OLBASE Library Kullanımı

Olbase dökümantasyonu sadece referans amaçlı bilgiler verdiğinden burada pratikteki kullanımına örnekler verilmiştir. Programları inceleyerek zamanla library ocean geliştirme standartlarına alışılır.

Ocean standart Veri Tipleri ve temel sabitler

Basit veri tipleri de ocean'da tekrar tanımlanmıştır. İnteger tipi değişkenlerde kapladığı byte sayısı isimle birlikte kullanılmıştır. Projelerde hep bu değişkenler kullanılır. İlk başlarda farklı gelse bile zamanla alışılmaktadır. Alıştıktan sonra da gerçek değişken tipleri garip gelebiliyor.

- OSI1, OSI2, OSI4, OSI8 -> Ocean Signed Integer (byte sayısı ile)
- OUI1, OUI2, OUI4, OUI8 -> Ocean Unsigned Integer (byte sayısı ile)
- OSC -> char (genelde array şeklinde kullanılır)
- OUC -> unsigned char (genelde array şeklinde kullanılır, buffer)
- ODBL -> double (tutarlar da bu tipte tutulur)
- OLGC -> logical. B1 ve B0 makrolarını değer olarak veriyoruz.

FILLM : Bufferi temizlemek için şu makro sıkça kullanılır. Fakat memset kullanılan yerler de çoktur. Bu konuda bir zorlama yok.

```
FILLM (myStruct, 0);

FILLM (myarray, 0);
```

KB1 : 1024 yerine kullanılan sabit. 1KB'lık yer ayrıldığını gösterir.

```
OUC uBuffer[KB1];     → 1 KB'lık buffer
```

os_NL : satır sonu göstergeci. Windows için (0x0D 0x0A) şeklinde 2 byte satır sonunu gösterirken unixde 0x0D olarak 1 byte satır sonunu ifade eder. Dosyalara satır atılırken dikkate alınabilir. os_NL_size değişkeni de uzunluğunu gösterir.

> Not: bazı yerlerde bu değişkeni kullanmak yerine "\r\n" gibi string içinde geçiyor olabilir. Bu tip kodlar taşınabilirliği çok düşünmeden yazılmıştır.

Genel fonksiyon dönüş değerleri : fonksiyon dönüş değerleri için herhangi bir kısıt olmasa bile çoğu fonksiyon aşağıdaki 3 değeri dönmektedir:

- ORC_OKI : başarılı
- ORC_ERR : başarısız
- ORC_NON : data bulunamadı

Bir data getirmesi beklenen select cinsi fonksiyonlarda ORC_NON anlamlıdır. Aksi gerekmedikçe business fonksiyonlardan bu değerleri dönüyoruz.

Ocean Uygulamaları arası data taşınması

Uygulamalar arası haberleşmeyle ilgili kullanılan classlar yeri geldikçe anlatılacaktır. Burada daha çok kullanılan structure ve definition'lar açısından konu incelenecektir.

Ocean uygulamaları kendi aralarında çoğunlukla msmq üzerinden konuşur. Gönderilen datalar sabit structure (lar) şeklindedir. Programlar aynı sistem üzerinde çalıştığı için aiignment, byte order gibi bir problemimiz yok. Bir mesajda bir çok structure gidebilir, bunlar da TLV (tag-length-value) şeklinde taşınmaktadır (ayrıntılı anlatılacak). Aralarında TCP konuşan programlar da aynı struct yapısını kullanabilir. Protokol değişse bile data içeriği aynıdır.

Structure taşınmasının avantaj ve dezavantajlarından bahsedilebilir. Ancak dikkatli kullanılırsa dezavantajlar etkisiz hale getirilebilir. Bahse konu olabilecek en önemli dezavantaj, structure'lara field eklenmesi, çıkarılması, uzunluk değişikliği gibi konulardır. Mümkünse yeni alanlar en sona eklenir. Böylece bu alanı beklemeyen programlar etkilenmez. Alan çıkarma ve değiştirme çok fazla yapılan bir şey olmamakla birlikte bu durumda structure ismini değiştirerek eski ve yeni structure'ı belli bir süre paralel kullanmak yada, kullanan tüm programları aynı anda değiştirmek düşünülebilir. Alternatif olarak çıkarılan alanın yeri, ileride kullanılmak üzere boş bırakılır. Değişecek program sayısı ve etkisine göre karar verilir. Kimi zaman işe yaramayan field'lar çıkarma maliyetinden dolayı yerinde kalabilmektedir.

<u>OBox structure yapısı:</u>

Gate'ler arasında OBox dediğimiz ana structure taşınır. Bu structure içinde pek çok alt alan ve structure yer almaktadır:

```
typedef struct OBox
{
      OBoxHdr oHdr;           // message header
      OSI2  iMsgLen;          // binary message length
      OUC   uMsgSgm[2*KB1];   // binary segment of the message
      OSI2  iUsrLen;          // tlv message length
      OUC   uUsrSgm[1900];    // tlv segment of the message

}OBox;
```

oHdr alanı genel bir headerdır. Gönderen ve alan uygulamaların bilgilerini ve yönlendirme amaçlı dataları içerir. Bu structure'da değişiklik yapılmaz. İçeriğine bakılacak olursa:

- `OSI8 iGUID;` → Mesaj GUID alanı. Ocean sistemine giren her mesaj için unique bir numara üretilir ve mesaj takibinde kullanılır. Ayrıca işlem tablolarında da bu numara referasn olarak tutulur.
- `OSI8 iMUIR;` → Original GUID alanı. Reversal / void gibi işlemlerde orijinal işlemin GUID'i referans olarak burada taşınır. Diğer işlemlerde bu alan GUID ile eşittir.
- `OSI8 iTcpRcvTime;` → Mesajın dış sistemlerden Ocean sistemine girdiği anı gösterir (milisaniye cinsinden). Cevap dönülme anında toplam işleme süresini gösterebilir. Ayrıca ilk giriş noktasında queue'da bekleme süresi hesabında da

kullanılır. Makinalar arasında saat farkından dolayı, alan mesajı ilk program için anlamlıdır.

- `OSI2 iSrcGateCode;` → Gönderen uygulamanın kodu. Ocean sisteminde her uygulamayı tanımlayan unique bir numara bulunur. Başka bölümlerde anlatılacak.
- `OSI2 iDstGateCode;` → Mesajı alacak uygulamanın kodu.
- `OSC iSrcGateNtw;` → Gönderen uygulamanın tipi.
- `OSC iDstGateNtw;` → Mesajı alacak uygulamanın tipi.
- `OSC iIntTraceNo;` → trace no. Geçilen her uygulama bu değeri bir artırır. Bu mesajın kaç uygulamadan geçtiğini gösterir.
- `OSI2 iFwdGateSsnIdx;` → Dış sistemlerle Ocean TCP Client olarak konuşuluyorsa, mesajın geldiği yada gideceği session numarası. Gönderen uygulama için bir IP-Port çiftine denk gelir. ONwmTcm objesine bakılabilir. Bu bilgiden bağlantı kurulan socket no'ya ulaşılarak mesaj gönderilir.
- `OSC sFwdGateAddr[40+1];` → Dış sistemlerle Ocean TCP Server olarak konuşuyorsa, mesajın geldiği yada gönderileceği adres bilgisini içerir. ONwmTcc objesine bakılabilir. Bu bilgiden bağlantı kurulan socket no'ya ulaşılarak mesaj gönderilir.

uMsgSgm alanında IMF (Internal Message Format) dediğimiz genel bir structure taşınır. Bu structure, çok kullanılan alanları içermektedir. iMsgLen alanı da Imf structure uzunluğunu göstermiş olur. Bazen işin kolayına kaçmak adına çok kullanılmayan alanlar da bu structure'a eklenmektedir. uMsgSgm alanı aşılmadığı sürece sorun olmamakla birlikte, nadiren kullanılan alanların TLV dataya eklenmesi tercih edilir.

```
typedef struct OImf
{
      OSI2 otc;
      OSI2 ots;
      OSI2 ote;
      ...

}OImf;
```

Imf Sadece gate'ler arasında taşıma amaçlı kullanılmaz, aynı zamanda uygulamalar içinde pImf pointer'ına cast edilir ve çok yerde kullanılır. Uygulamalanın bu alanları ayrıca bir local struct'a alması gerekmez. Uygulama kodları incelendiğinde, pImf pointer'inin binlerce yerde geçtiği görülebilir. Zaten kısa süre kodlama yapıldıktan sonra fonksiyonu anlaşılablir.

Dışarıdan mesaj geldiğinde, parse edilerek Imf içinde mümkün olduğunca çok alan doldurulmaya çalışılır. Bazı Imf alanları, tablolardan data alınarak yada belli dataların birleşitirilmesi, dönüştürülmesi ile de doldurulur.

Imf datasının maximum boyutu oBox.uMsgSgm alanının boyu kadar olmalıdır Genellikle Imf bu alandan daha kısadır. Bu nedenle uMsgSgm buffer'in sonunda bir miktar boşluk kalır.

uUsrSgm alanı ise Imf içinde yer almayan dataların gate'ler arasında taşınabilmesini sağlar. Burada taşınan data TLV formatında ucuca eklenmiş structure'lar şeklindedir. iUsrLen alanı burada taşınan bilginin boyunu gösterir. İşlemde hiç bir struct taşınmayabilir bu durumda iUsrLen = 0 olarak gider. TLV data formatı şu şekildedir:

TAG (ASCII – 4 byte) + LENGTH (BCD – 2 BYTE) + DATA (LENGTH BYTE)

Tagler Olbase içindeki Def dosyaları içinde tanımlanmıştır, genelde program tipine göre ayrı def dosyalarındadır ve unique değerleri vardır. Gönderen ve alan programlar olbase içindeki aynı tag değerlerini kullanır:

Örnek bonus tagleri (OBnsDef.h)

```
#define IT_BNS_ORIG_TX        "0B36"
#define IT_BNS_CAVV           "0B42"
#define IT_BNS_INST_REQ       "0B98"
```

Örnek Pos Tagleri (OPosDef.h)

```
#define IT_POS_FREE_MSG       "PFRE"
#define IT_POS_MERCHANT       "PMRC"
#define IT_POS_CHEQUE         "PCHQ"
```

Her tag'e karşılık gelen bir structure bulunur. TLV'nin Data kısmında bu structure gönderilir. Dolayısıyla TLV Length kısmında da structure length'i gönderilmektedir. Örnek olarak, yukarıdaki 2 tag'in karşılığı olan strructure'lar şunlardır:

➔ IT_BNS_ORIG_TX

```
typedef struct  BSOrigTxData
{
      OSI4 iOrigDate;
      OSI4 iOrigStan;
      OSC  sOrigRrn[12+1];
      OSC  sOrigAuthCode[6+1];
      OSI8 iGuidOrg;
      ODBL fOrigAmount;

}BSOrigTxData;
```

➔ IT_POS_MERCHANT

```
typedef struct PSMrcLyl
{
      OSI8 iGlbLoyaltyProfileGuid;
      OSI8 iMrcLoyaltyProfileGuid;
      OSI8 itxn_group_guid;         // auth
      OSI8 itxn_detail_guid;        // auth
      OSC  sMccGrpCode[4+1];
      OSC  sMrcCityCode[3+1];
      OSI2 iBranchId;
```

```
        OSI8 iChainGuid;
        OSI8 iCommissionProfileGuid;
        OSI8 iMrcCustomerNo;
        OSC  sMrcTaxNo[11+1];
        OSC  cMcTccCode;

}PSMrcLyl;
```

TLV data içinde IT_BNS_ORIG_TXN tag'i ile, BSOrigTxData structure gönderilir. Gönderen ve alan uygulamalar tag ve strucutre değerlerinde anlaşmıştır. Structure içeriğini değiştirmenin etkileri yukarıda anlatılmıştır.

Uygulamalarda TLV datanın oluşturulması ve parse edilmesi standart şekilde yapılmaktadır. Aşağıdaki örnekte TLV objesinin configure edilmesi, parse edilmesi ve bir tage karşılık gelen datanın alınması görülmektedir.

```
// *** startup config ***
oTlvInc.Config(OFIRST_TAG, 4, 2, OLEN_TYPE_BCD, OTAGLEN_EXC);

// *** işlem öncesi TLV objesi temizlenir ***
oTlvInc.TagClr();

// *** TLV data parse ediliyor ve TLV objesi dolduruluyor ***
iRet = oTlvInc.FldRes(oBox.uUsrSgm, sizeof(oBox.uUsrSgm));
if (iRet == ORC_ERR)
      // err

// *** TLV data içinden tag alınıyor ***
iRet=oTlvInc.TagGet(IT_POS_MERCHANT,&sMrcLyl,sizeof(PSMrcLyl),&iLen);
if (iRet == ORC_ERR)
      // err
if (iRet == ORC_NON)
      // tag not found
```

Gönderilen taglerin eklenmesi ve mesaja konulması da benzer bir şekilde yapılır. TLV objesine hiç bir tag eklenmemişse uUsrSgm boş kalır.

```
// ****** TLV objesi tanımı ********
OMsgTlv oTlvOut;

// ****** startup config ********
oTlvOut.Config(OFIRST_TAG, 4, 2, OLEN_TYPE_BCD);

// *** işlem öncesi TLV objesi temizlenir ***
oTlvOut.FldClr();

// *** TLV objesine yeni tag ekleniyor ***
iRet=oTlvOut.TagAdd(IT_BNS_STMT,&bSStatement, sizeof(bSStatement));
if (iRet == ORC_ERR)
      // err

// *** TLV objesinde TLV data üretilip oBox içine konuyor ***
iRet = oTlvOut.FldGen(oBox.uUsrSgm, sizeof(oBox.uUsrSgm));
if (iRet == ORC_ERR)
      // err
oBox.iUsrLen = iRet;
```

Gelen ve giden tag'ler için 2 farklı TLV objesi tanımlanmıştır. Çünkü bir işlemin herhangi bir aşamasında gelen TLV objesinden tag alınabilir yada giden TLV objesine eklenebilir.

TLV içindeki data yapıları iki yönlü kullanılabilir. Bazıları acquirer'dan issuer'a giden request alanlarını taşırken bazıları cevap mesajına ait alanları taşır. Bazıları ise hem giden hem de dönen mesajda bulunur. Örneğin datanın bir kısmını acquirer uygulama, bir kısmını da issuer uygulama doldurabilir.

Gönderilen structure'ın aynen geri dönülmesi gereken durumlar da vardır. Gönderen uygulama dataları hafızasında saklamadığı için dönüşte aynı bilgilere ihtiyaç duyabilir (tabloda da saklamıyorsa). Mutlaka geri dönülmesi gereken structure'lara programsal olarak karar verilir ve kodlanır. Zaten Imf yapısında da benzer şekilde gönderilen bilgiler aynen geri dönülmekte ve kullanılmaktadır.

Aşağıda örnekte değişmeden dönülen bir data örneği görülüyor. Gelen TLV data içinde ATM_MSG_HDR tag'i varsa otomatik olarak cevap TLV datasına ekleniyor

```
iRet = oTlvInc.TagGet(IT_ATM_MSG_HDR, &uBff1, sizeof(uBff1), &iLen);
if (iRet == ORC_ERR)
{
	_ERRF("F968297", "IT_ATM_MSG_HDR oTlvInc.TagGet error..");
	iErrCnt ++;
}
else if (iRet == ORC_OKI)
{
	iRet = oTlvOut.TagAdd(IT_ATM_MSG_HDR, &uBff1, iLen);
}
```

OHsm structure yapısı

Programlar ile Hsm server arasında bilgi taşınması bu structure ile yapılır (normalde burada da obox yapısı olabilirdi - YB). Normal program akışında bu structure ile hiç işimiz olmamaktadır. Olbase Hsm objesinde gizli olarak bulunur ayrıca Hsm Server programında da kullanılır. OBox yapısına benzer şekilde header kısmı ve data kısmı vardır. Data kısmı başka custom structure'ların taşınmasında kullanılır.

```
typedef struct OHsm
{
	OHsmHdr sHdr;		// hsm hdr
	OSC	sMsgDat[KB1+256];  // HSM message structure

}OHsm;
```

Header yapısı oBox headerdan farklıdır. Öenmli alanlara bakacak olursa, sSrcHst ve sSrcBox alanları cevap mesajının atılacağı queue bilgilerini içerir. Böylece hsm server path tanımlarından bağımsız olarak istenen queue'ya cevap dönebilir. iThrId değişkeni gönderen threadin ID'sidir. HSM protokolü Senkron olarak çalıştığı için bir thread aynı anda tek requestte bulunmuş olabilir. CEvap mesajlarının doğru thread'e ulaştırılmasını sağlar (aslında

quid yeterli olurdu - YB). Header içinde bazı timer değerleri de taşınır. Gelen cevabın zamanında gelip gelmediği kontrol edilir (bu da gereksiz - YB).

OHsm yapsındaki sMsgDat alanında, Hsm komutunda ihtiyaç duyulan structure taşınır. Örneğin, Generate Pin komutu için, request ve reply mesajlarında sırasıyla aşağıdaki structure'lar taşınır. Her komut için benzer structure'lar Olbase içindedir. Bu strucutre'lar da Hsm server programı dışında programlardan kullanılmaz. Olbase library içinde hidden olarak kullanılır.

```
typedef struct SGenRndPin
{
	HSndHdr hSndHdr;
	OSC	sPan[19+1];
	OSI4	iPinLen;

}SGenRndPin;

typedef struct  RGenRndPin
{
	HRcvHdr hRcvHdr;
	OSI4	iPin;

}RGenRndPin;
```

Hsm library'nin kullanımı ileride anlatılmıştır. HSM'in daha ayrıntılı anlatılması başka bir döküman konusu da olabilir.

OHstMsg structure yapısı

Hsm struct yapısına çok benzer. OHsm struct için söylenen şeyler geçerlidir. Bankacılık sistemiyle konuşan H2H gate isimli gate ile diğer programlar bu structure ile haberleşir. ONwmHst objesinin kullanımı ileride anlatılmıştır.

```
typedef struct OHstMsg
{
	OHstHdr sHdr;			// host hdr
	OSC	sMsgDat[KB2];		// H2H message structure

}OHstMsg;
```

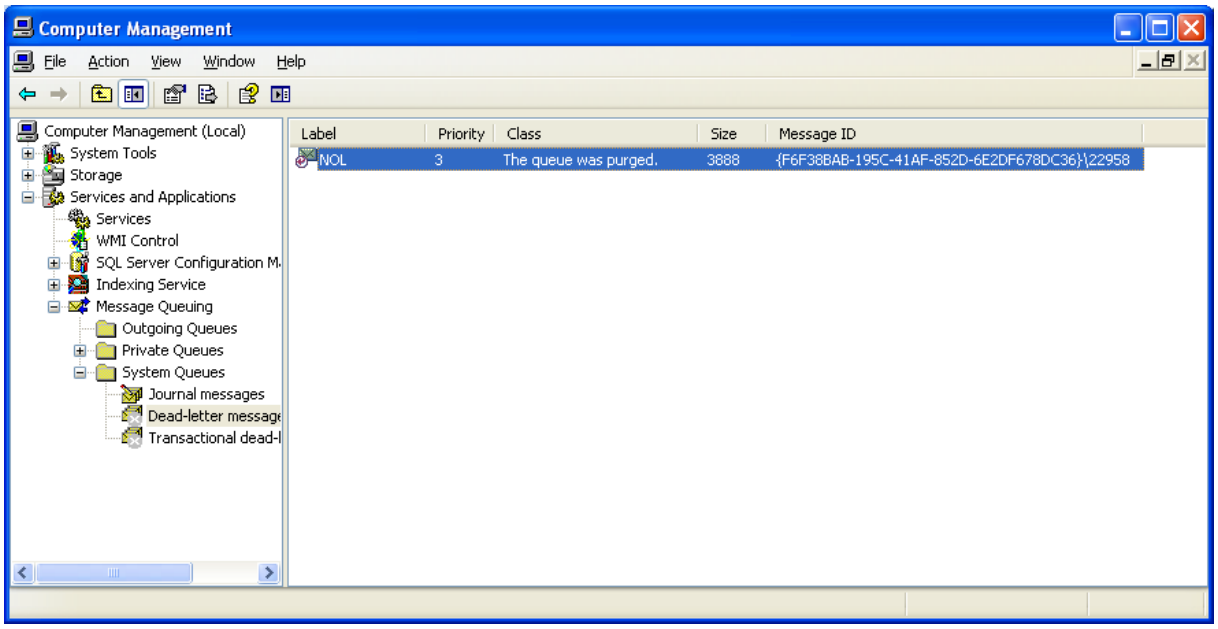
Hsm ve H2h structlarına benzer olarak değişik hostlarla haberleşme amaçlı pek çok yeni struct oluşturulmuştur. Bunların yapıları hemen hemen aynıdır ve bu yapıları kullanan protokol objeleri de aynı mantıktadır. Normalde hepsi için tek bir struct yapısı ve ortak altyapı kullanılabilirdi.

MSMQ kullanımı, genel özellikleri ve sorunlara müdahale (ONwmMmq)

- Msmq objesinin kullanımı oldukça kolaydır. IP address ve Queue ismi ile config yapılıp istenen queue'ya mesaj gönderilir yada alınır. Aşağıdaki örnekte mesaj gönderilmektedir.

```
ONwmMmq oNwmMmq;

iRet = oNwmMmq.Config(sDstHst, sDstBox, MSQ4_SEND);
if (iRet != ORC_OKI)
	// error

iRet = oNwmMmq.Push(&sEvt, sizeof(OEvt));
if (iRet != ORC_OKI)
	// err

oNwmMmq.Free();
```

- Bir obje sadece mesaj göndermek yada almak için config edilebilir (MSQ4_SEND, MSQ4_RECV). Aynı anda ikisini birden yapamaz. Gönderen fonksiyon adı "Push", alan fonksiyon adı ise "Pop"'tur, gönderme amaçlı config edilmiş obje ile Pop komutu hata alır yada tersi.
- Bir kez config edildikten sonra istenen sayıda push / pop yapılabilir. Program kapanırken Free fonksiyonu çağrılarak resource'lar geri verilir. Free çağrıldıktan sonra gönderip alınamaz. Tekrar config çağrılmalıdır.
- Queue aynı makinada ise, Host IP adresi yerine, MSQ4_LOCAL parametresi geçilebilir.
- Maximum mesaj boyu 4K'dır. Config fonksiyonunda 4 nolu opsiyonel parametre 1 olarak geçilirse, 64K'ya kadar mesaj alınıp gönderilebilir. Ancak 4K'dan büyük ihtiyaç olmayacaksa set edilmemelidir. Boşuna memery allocate eder.
- Aynı makinadaki queue'ya memory üzerinden mesaj gönderildiği için çok hızlı mesajlaşma olur. Farklı makinada ise TCP/IP üzerinden mesajlaşılacağı için biraz sade TCP/IP'ye nazaran overhead olabilir.
- Farklı makinadaki bir queue config edilirken yada gönderip alma fonksiyonlarında, **queue olmasa bile hata almaz**. Hatta karşı makinanın kapalı olması durumunda bile hata alınmaz. Aynı makinada queue yoksa fonksiyonlar hata döner.
- Send fonksiyonunun hata dönmemesi, mesajın gittiği anlamına gelmez. Karşı makinaya ulaşılamıyorsa, mesaj **outgoing queue**'da bekler. Makinaya ulaşılırsa mesaj gönderilir ama karşı makinada queue bulunamaması, yada izin olmaması gibi hatalardan dolayı **dead-letter queue**'ya mesaj düşebilir.



Yukarıdaki Computer Management ekranında Outgoing ve Dead-Letter queue'lar görünmektedir. Dead-Letter queue'da mesajın neden buraya düştüğü de Class kısmında görülebilir. Outgoing Queue ekranında da mesajların neden bulunduğumuz makinadan çıkamadığı anlaşılabilir. Tipik olarak karşı makinaya erişemediğinde buraya düşer.

- Aşağıdaki registry parametreleri set edilmelidir. Aksi halde default değerler bazı sorunlara yol açmakta. TcpNoDelay parametresi, msmq'nun küçük mesajları bekletip sonraki mesajlarla birlikte göndermesini önlüyor. Burada 100 ms gibi ufak bir beklemeden bahsediyoruz. Aslında bu güzel bir özellik, network overhead'ı düşürür ancak bizim online uygulamalarda tercih etmiyoruz. İkinci parametre olan "WaitTime" ise, başka bir makinayla msmq bağlantısı uzun süre inaktif kaldığında ne kadar süre sonra bağlanacağını gösteriyor. Sanırım bu parametreyi de bağlantı kurulamıyorsa boşuna bağlantı kurmayı denemesin diye koymuşlar. Ancak uzun süre işlem gelmediğinde gelen ilk işlemin 20-30 saniye beklemesine yol açtığından bu parametre de bizim için kabul edilebilir değil.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSMQ\Parameters]
"TcpNoDelay"=dword:00000001
"WaitTime"=dword:000007d0
```

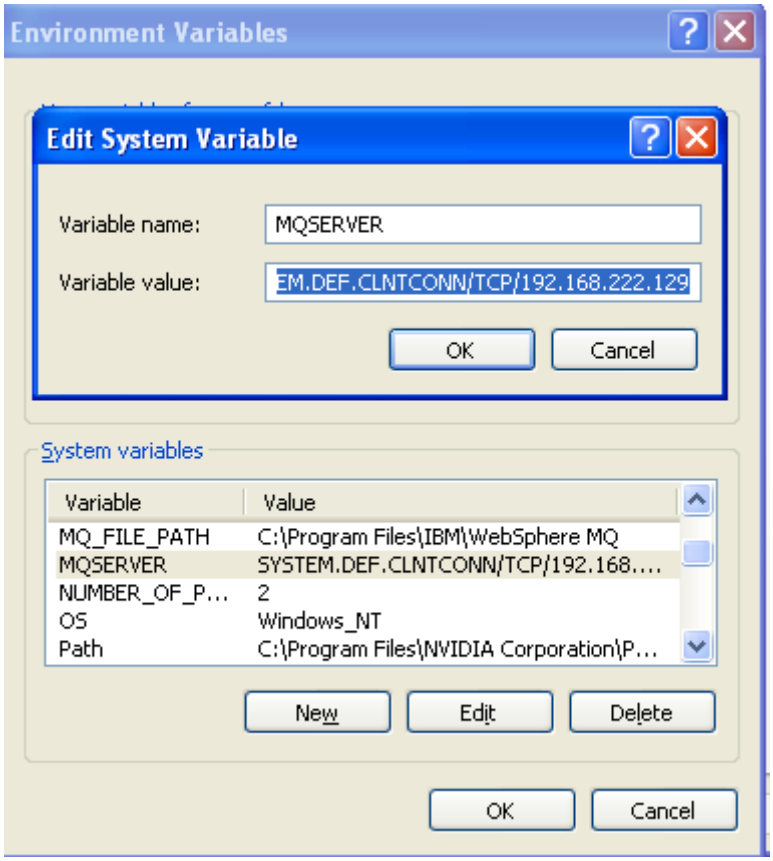
- Kullandığımız queu'lar transactional değildir. Queue oluşturuken dikkat edilmelidir. Queue'larda Journal opsiyonu disable olmalıdır. Aksi halde local diskte de bir kopyasını saklayacağı için diski dolduracaktır.
- Bir queue'ya sürekli mesaj geldiği halde mesajları alan olmazsa (alan program kapalı olabilir), dead-letter queue'ya çok fazla mesaj düşerse, vb durumlarda msmq disk alanı dolar. Bu durumda bu makinaya yeni mesaj gelmesi ve bu makinadan mesaj gidişi imkansız hale gelir. İlgili queue'lar truncate edilerek yada msmq servisi restart

edilerek bu durumda kurtulunabilir. Esas olarak queu'larda birikmeye yol açan sebepler bulunup düzeltilmelidir. MSMQ storage alanı default 1GB'dır. Sorun yaşanırsa bu alan da artırılabilir.

- Online programlarda karşı uygulamanın ayakta olduğu kontrol edilip daha sonra mesaj gönderildiği için genel olarak queue'larda şişme olmaması gerekir. Ancak bağlantının kesilmesi gibi problemlerden dolayı nadiren msmq disk alanı dolabilir. Prod ortamında yaşanmış bir durum değil.

IBM MQ Library Kullanımı, Kurulum ve Sorunlara müdahale (ONwmImq)

IBM mq objesinin kullanımı msmq objesine benzer. Fonksiyon isimleri aynıdır. Ancak config fonksiyonun server ismi yerine Queue Manager ismi verilir. Server bağlantı parametresi Environment variables altında MQSERVER parametresiyle girilir. (Not : Config fonksiyonunda ip port verilebiliyormu araştırmakta fayda var).



Server bilgisi şu formatta verilir "SYSTEM.DEF.CLNTCONN/TCP/192.168.222.129". Baştaki "SYSTEM.DEF.CLNTCONN" server tarafındaki channel tipidir. Daha ayrıntılı bilgi için Ibm dökümanlarına bakılabilir. Bir server'a bağlanıp queue'ya mesaj atmak yada almak için karşı taraftan şu bilgilerin alınması gerekir:

- Channel adı
- Server Ip adresi (default port kullanılmıyorsa port no da gerekir).
- Queue Manager adı
- Queue Adı

IBM mq'nun çalışma mantığında msmq'dan önemli bir farkımız var. Burada local queue kullanımı yoktur. Tüm queue'lar server üzerindedir. Client uygulamaları serverdaki queue'lara mesaj atar cevap alır. Dolayısıyla msmq'da olduğu gibi local haberleşme için kullanılması efektif değildir. Sadece gerektiğinde mainframe ile haberleşmede kullanıyoruz.

Client tarafında IBM mq client kurulumu yapılmalıdır. Hem geliştirme hem de uygulamaların çalıştırılması için gereklidir. Derleme yapabilmek için aşağıdaki dizinlerin C++ dizinlerine eklenmesi gerekir:

Include dizini : C:\Program Files\IBM\WebSphere MQ\tools\c\include "cmqc.h"

Library dizini (32 bit) : C:\Program Files\IBM\WebSphere MQ\tools\Lib "MQIC32.lib"

Library dizini (64 bit) : C:\Program Files\IBM\WebSphere MQ\tools\Lib64 "MQIC.lib"

IBM mq bağlantı be mesajlaşma sırasında pek çok hatalar çıkabilir. Öncelikle hata kodu alınıp "cmqc.h" dosyasında anlamına bakılabilir (internetten de aratılabilir). Hata kodunu almak için. `oNwmImq.GetLastError()` fonksiyonu kullanılır. Bazı durumlarda ayrıntılı hata bilgisine aşağıdaki dizin ve dosyadan alınabilir:

```
C:\Program Files\IBM\WebSphere MQ\errors\AMQERRxx.LOG
```

Gerektiğinde IBM mq ile ayrıntılı bilgi için ibm dökümanları incelenebilir.

DB Library kullanımı (ODbmCnn, OdbmCol, ODbmCrs, ODbmPrm)

Db librarysine erişim genelde otomatik üretilen kodlarla sağlandığı için günlük ekstra kullanım yada bilgi ihtiyacı olmaz. Projelerde connection kuran kısımlar da standart olarak eklenmektedir. Burada ne ne işe yarar kısaca anlatılmıştır. ODbmCol ve OdbmPrm objeleri cursor objesi içinde gizli olarak bulunur ve programda tanımlanmasına gerek yoktur. Cursor objesinin parametre ve kolonları get / set eden fonlisyonları içeride bu objeleri kullanır.

- Database bağlantısı kurulması ve koparılması:

```
ODbmCnn oDbmCnn;

oDbmCnn.CnnConfig(sDbmHost, sDbmUser, sDbmPswd);
oDbmCnn.ConfigExecTimeout(10000);
iRet = oDbmCnn.CnnStart(3000);
if (iRet != ORC_OKI)
       // err


iRet = oDbmCnn. CnnStop();
```

Yukarıdaki örnekte db'ye bağlantı kurulup akabinde koparılmıştır. Normalde program başında bağlantı kurulup çıkışa kadar bağlantı açık kalmaktadır. Örnekte kullanımı göstermek açısından Stop fonksiyonu da çağrıldı.

CnnConfig fonlsiyonunda ilk parametre TNS adıdır. Bunun yerine herhangi bir bağlantı string'i de kullanılabilir. Daha sonra DB user ve password parametreleri yer alır.

CnnStart fonksiyonu ile DB bağlantısı kurulmuş olur. Parametre olarak verilen değer connection timeout süresidir. Normalde kısa sürede bağlantı kurulur.

Query Timeout Verilmesi

"ConfigExecTimeout" fonksiyonu ile, connection üzerindeki default query execution timeout süresi verilir. Bu parametre verilmezse default değeri 300 saniyedir. Ayrıca her bir query için özel timeout süresi verilebilir. Örneğin yukarıdaki örnekte connection üzerindeki default timeout süresi 3 saniye verilmiştir. Bazı query'ler 8-10 saniye sürebilecekse, o query'lere özel timeout süresi verilebilir (bkz: oDbmCrs objesinin CmdInit fonksiyonuna parametre olarak).

Bir connection üzerinde hiç query timeout olması istenmiyorsa aşağıdki fonksiyon çağrılır. Bu şekilde connection gitmediği yada oracle kesmediği sürece query çalışmaya devam eder.

```
oDbmCnn.DisableTimeout();
```

- Query tanımlanması (query config aşaması)

Query cümleleri otomatik üretiliyor ancak çoğu zaman sonradan elle edit edilmektedir. Şu sebeplerden elle query'leri editlenme zorunluluğu oluşuyor.

  - ORMapper uygulaması tablodaki tüm kolonları query'lere ekler. Oysa update cümlelerinde sadece 1-2 kolonun update edilmesi yeterli olmaktadır. Select cümlelerinde tüm kolonların yerine sadece ihtiyaç duyulanların alınması da performansı etkiliyor.
  - ORMapper where cümlesine tüm indexli kolonları ekler. O anda Index yoksa where cümlesi oluşmaz. İhtiyaca göre where kriterlerini vermemiz ve parametre tanımlarını yapmamız gerekir.
  - ORMapper sadece standart select/insert/update/delete cümleleri oluşturur. Gerçekte ise bir tabloda bir çok değişik query ihtiyacı olmakta. Bu yüzden yeni cursor'ler oluşturup elle editlenmekte. Sorun yaşanmaması için mevcut cursor objelerinden kopyalanarak oluşturulmasında fayda var.
  - Bazen 2 tabloyu joinlemek gibi daha karışık queryler gerekmekte, bunları da ORMapper üretemiyor.
  - Sp çağrıları için ORMapper'dan kod üretilmiyor. Bunları da elle üretmek gerekiyor.

```
ODbmCrs  oCrsSel;
```

```
oCrsSel.CmdInit(&oDbmCnn, OSA_SQL,
       "SELECT "
       "c.GUID, c.STATUS, c.LASTUPDATED "
       "FROM OC_CRD.CRD_CARD c "
       "WHERE c.CARD_NO = :card_no "
       "AND c.STATUS = 1 ");

oCrsSel.PrmNew("card_no", OVT_TXT, OPD_INP);
```

Yukarıdaki örnekte bir cursor initialize edilmiş ve bir query parametresi tanımlanmıştır. CmdInit fonksiyonuna start aşamasını geçmiş bir connection parametre olarak verilir. Connection resetlendikten sonra CmdInit tekrar çağrılır.

CmdInit fonksiyonunda son bir opsiyonel parametre daha vardır. fiTimeout parametresi verilirse, bu query'e özel timeout süresi milisaniye cinsinden verilebilir.

PrmNew fonksiyonunda da opsiyonel fiScale parametresi bulunuyor. Bu parametre noktalı sayılar için noktadan sonra kaç hane alınabileceğini gösterir. Default değeri 2 'dir ve string yada tam sayılar için bir anlamı yoktur.

Query'de geçen tüm parametreler PrmNew fonksiyonu ile eklenir. Sp tipindeki query'lerde outpu parametreleri de bulunaiblir. Outout parametreleri OPD_OUT parametresi ile eklenir.

Parametre tipinin OVT_TXT yada OVT_NUM olmasına dikkat edilmelidir. Yeni kolon eklemelerinde bazen hatalı değer verilebilmektedir. Ocean tablolarında sadece VARCHAR2 ve NUMBER tipinde kolon kullanıyoruz.

Query config edilmesi program içinde yukarıdaki gibi açık olarak yapılmaz. Cursor ve parametre configleri T_ ile başlayan tablo objelerinde yer alır. Bu objeler üzerinden config fonlksiyonu çağrılır. Örnek:

```
T_CRD_CARD    CRD_CARD;

iRet = CRD_CARD.Config(&oCnnTxn);
if (iRet != ORC_OKI)
       __ERRD("D968222", "CRD_CARD.Config error", ORC_ERR);


OSI4 T_CRD_CARD::Config(ODbmCnn *fiCnn)
{
       ...
       oCrsSel.CmdInit(fiCnn, OSA_SQL,
```

CRD_CARD objesinin config fonksiyonu içinde de cursor'ın init edilmesi ve parametrelerin oluşturulması vardır.

- Query Çalıştırılması (execute ve fetch aşaması)

Query çalıştırılmasının aşamaları şunlardır:

a. İnput parametre değerlerinin verilmesi
b. Execute
c. Select query'leri için Fetch işlemleri
d. Sp'ler için (varsa) output parametrelerinin alınması

Aşağıdaki örnek basit bir select query'sinin çalıştırılmasını göstermektedir. "card_no" input parametresi verilir. CmdExec fonksiyonu query'i çalıştırır. ResFetch fonksiyınu ilk satırı getirir. Select sonucunda kayıt gelmemişse ResFetch fonksiyonu ORC_NON döner. Diğer durumlarda kayıt içinderki değerler cursor objesinde tutulmaktadır. "ColGet" fonksiyonu ile o satır için ilgili kolona ulaşılıp değeri alınır. Değeri alınırken AsOSI8, AsOSI4, AsOSI2 fonksiyonları değeri integera çevirir. AsODBL fonksiyonu kolon değerini floating point sayı olarak alır. AsOSC değeri karakter pointer'ı olarak döner. Bu değer bir sonraki fetch komutuna kadar cursor objesi içinde kalmaya devam edecektir.

```
oCrsSel.PrmGet("card_no") = fiStr->card_no;

iRet = oCrsSel.CmdExec();
if (iRet != ORC_OKI)
	__ERRD("D968301", pDbmCnn->CnnError(), ORC_ERR);

iRet = oCrsSel.ResFetch();
if (iRet == ORC_ERR)
	__ERRD("D968302", pDbmCnn->CnnError(), ORC_ERR);

if (iRet == ORC_NON)
	return ORC_NON;

fiStr->guid = oCrsSel.ColGet("GUID").AsOSI8();
fiStr->status = oCrsSel.ColGet("STATUS").AsOSI2();
fiStr->lastupdated = oCrsSel.ColGet("LASTUPDATED").AsOSI8();
strncpy(fiStr->card_no, oCrsSel.ColGet("CARD_NO").AsOSC(), sizeof(fiStr-
>card_no)-1);
```

Bu query bir insert/Update/delete query'si olsaydı, CmdExec'den sonra fonksiyondan Ok dönülürdü.

Sp'lerden output parametresi alınmasına bir örnek verelim:

```
oCrsSpc.PrmGet("fi_seq_code") = fiStr->fi_seq_code;

iRet = oCrsSpc.CmdExec();
if (iRet != ORC_OKI)
	__ERRD("D968275", pDbmCnn->CnnError(), ORC_ERR);

fiStr->fo_val_new = oCrsSpc.PrmGet("fo_val_new").AsOSI8();
```

Execute aşaması başarıyla geçildikten sonra PrmGet ile output parametresine ulaşılıp değeri alınır.

Cursor'ların çalıştırılması, parametrelerin verilip alınmasıı satır değerlerinin doldurulması gibi işlemler programlarda business kısmında yapılmaz. T_ ile başlayan

table objeleri üzerinden yapılır. Bir programda Select cümlesinin business tarafta çağrılması şu şekildedir:

```
R_CRD_CARD rCRD_CARD;

strcpy(rCRD_CARD.card_no, pImf->pan);

iRet = CRD_CARD.Select(&rCRD_CARD);
if (iRet == ORC_ERR)
	// err

if (iRet == ORC_NON)
	// not found
```

Tablodaki bir satırı tutabilen R_ ile başlayan struct kullanılır. Sp'ler için bu structure tüm input ve output parametrelerini içerir. Db fonksiyonu çağrılmadan önce bu structta input parametreleri doldurulurve parametre olarak geçilir. Çıkışta select edilen satır, yada sp'nin output parametreleri de bu structure içinde yer alır. Update/Insert tipi cümlelerde structure değişmeden dönecektir.

ResFetch komutu arka arkaya çağrılabilir. Böylece query'den gelen tüm satırlar alınabilir. Yukarıdaki gibi tek satır beklenen query'lerde bir kez çağrılması normaldir. Son satır alındıktan sonra bir sonraki ResFetch komutundan ORC_NON döner.

TCP Library kullanımı (ONwmTcp, OnwmTcm-Server, OnwmTcc-Client, OnwmAtm - Server)

Programlarda OnwmTcp objesi pek kullanılmaz. Çünkü bu obje kullanıldığında receiver thread ayağa kaldırılması, flow kontrol gibi işleri uygulamanın halletmesi gerekir. Utility tarzı uygulamalarda bu obje kullanılabilir. Ancak gate uygulamalar daha gelişmiş olan diğerobjeleri kullanır.

OnwmTcm objesi belli bir portu dinleyen, gelen her connection için bir receiver thread açan ve receiver thread'lerin aldığı mesajları uygulamanın transaction queue'suna atan objedir. Uygulama cevap mesajını da bu obje üzerinden geldiği noktaya gönderir.

Örnek kullanım:

```
oNwmTcm.ConfigFormat(MSG_FRM_BLN, LEN_FORM_BIN, 2, 0, 0,"C:\\LOG\\");
oNwmTcm.ConfigBox(MSQ4_LOCAL, aCchPrm1.rSYS_GATE.gate_txn_box);
oNwmTcm.ConfigPort(pSYS_GATE_CNN->prm_host_port, 10);
iRet = oNwmTcm.Start(&oNwmMon);
```

Objenin ayağa kalkması için bu kadarlık bir config yeterli olmaktadır. Start fonksiyonu başatılı olduktan sonra ilgili porttan bağlantı kabul edilmeye başlanır.

ConfigPort fonksiyonu ile dinlenecek olan port parametre olarak verilir. Bir makinada bir portu sadece bir uygulama dinleyebilir.

ConfigBox fonksiyonu ile gelen mesajların hangi queue'ya atılacağı söylenir. Aynı queue'dan uygulama içinde mesajların alınması gerekir. Yoksa gelen mesajlar birikecektir.

ConfigFormat fonksiyonu ise gelen/giden TCP mesajlarının length ve header formatını gösterir. Tek hat üzerinden asenkron gönderim yapabilmek için mesajların başına length konulması gerekir. Alternatif olarak mesaj seperator kullanılabilir ancak binary mesajlarda pratik bir kullanım olmaz. Tüm mesajların aynı boyda olması gibi bir yöntem de pratik değildir. Mesaj başında length gönderilmesi en mantıklı seçenektir.

İlk parametre olan Message Format parametresinin alabileceği değerler ve anlamları şu şekildedir:

- `MSG_FRM_ISO : 2 byte binary length + 5 byte TPDU + mesaj. Alınan mesaj SIso struct içine oturtulur.`
- `MSG_FRM_BLN : 2 byte binary length + mesaj. Alınan mesaj SBin struct içine oturtulur.`
- `MSG_FRM_EST : 4 byte ASCII length + mesaj. Alınan mesaj SEst struct içine oturtulur.`
- `MSG_FRM_SPECIAL : Parametrik. Diğer parametreler ile format belirlenir. Alınan mesaj STcpData struct içine oturtulur.`
- `MSG_FRM_NOLEN : lengthsiz mesaj okunur ve raw olarak uygulamaya geçilir.`

Aslında tüm mesaj formatları MSG_FRM_SPECIAL ile ifade edilebilir. Nihai olarak tüm uygulamaları bu tipe çevirmeye çalışıyoruz.

ConfigFormat fonksiyonundaki diğer parametreler sadece MSG_FRM_SPECIAL için geçerlidir. Diğer formatlarda kullanılmaz. MSG_FRM_SPECIAL için şu parametreler set edilir:

```
fiLenFormat : LEN_FORM_NON, LEN_FORM_BIN, LEN_FORM_BCD, LEN_FORM_STR

fiLenLen : length of the length

fiPreLen : Length'den önce gelen byte sayısı. Bunlar length içine
dahil değildir.

FiPosLen : Length'den sonra gelen ve length'e dahil olmayan byte
sayısı. Örneğin VISA mesajlarında LLZZ formatından dolayı, length'den
sonra 2 byte fazlalık vardır.
```

MSG_FRM_SPECIAL formatında gelen mesaj StcpData struct içine şu şekilde yerleşir. iLen değişkeninde gelen length integer olarak yer alır. uMsg kısmında iLen uzunluğundaki mesaj binary array olarak yer alır. uPre kısmında ise mesajdan önceki kısım sağa dayalı olarak bulunur. Örneğin, len_type = binary, lenlen = 2, prelen = 0, postlen = 0 durumunda, uPre değişkeninin son 2 byte'ında binary length yer alır. İlk 8 byte'ı anlamsızdır.

```
typedef struct STcpData
{
      OSI2     iLen;                  // Length of data in uMsg
      OUC      uPre[10];       // right aligned
```

```
        OUC     uMsg[KB4];

}STcpData;
```

Bağlanmış olan client'lardan gelen mesajlar nerede alınmaktadır diye bir soru gelebilir. Objenin içinde bu mesajlar receive edilmekte ve queue'ya atılmaktadır. Uygulama ayrıca queue'dan mesajları almalıdır. Genellike business threadlerin Pop fonksiyonlarında aşağıdaki gibi bir kodla gelen mesaj alınır:

```
iPopLen = sizeof(oBox);
iRet = oMmqTxn.Pop(&oBox, &iPopLen);
if (iRet == ORC_OKI)
      // message received
```

Queue'dan alınan mesaj OBox structure tipindedir. OBox structure'ın uMsgSgm değişkeni de sIso, sBin, sTcpData structlarından birini içerir. Hangi struct'a cast edildiği configFormat'ta verilen formata bağlıdır.

```
typedef struct OBox
{
      OBoxHdr oHdr;
      OSI2  iMsgLen;          → uMsgSgm içindeki struct'ın length'i
      OUC   uMsgSgm[2*KB1];   → SIso, SBin, STcpData ...
      OSI2  iUsrLen;
      OUC   uUsrSgm[1900];

}OBox;
```

Box header içinde bağlanan adres bulunur. Cevap mesajı gönderilirken aynı adresin box header içinde geçilmesi gerekir. Hangi socket'e gönderileceği buradaki adres bilgisinden anlaşılır.

```
typedef struct OboxHdr
{
      ..
      OSC   sFwdGateAddr[40+1];      → sender address
```

Mesaj işlendikten sonra yine aynı TCP server objesi üzerinden cevap gönderilir. Cevap mesajında OBox yapısı içindeki uMsgSgm değişkeni yine ilgili mesaj yapısının gerektirdiği structure'ı içerir (SIso, SBin, STcpData ...). Header içindeki adres bilgisi de gelen değerle aynı olmalıdır.

```
iRet = oNwmTcm.Send(fibox);
```

Standart OnwmTcm objesi sınırsız sayıda bağlantıya izin vermekte. Ancak bunun özel bir case'i olarak tek bağlantılık server objesi de tanımlanabilir. Özellikle switch tarzı çift yönlü bağlantılarda sadece tek bağlantı kurulması tercih edilebilir. Bu modda kurulu bağlantı varsa ikinci bağlantıya izin verilmez. Askıda kalmış bir bağlantı vs. varsa hattın resetlenmesi gerekir.

```
oNwmTcm.SetSglCltSck();
iRet = oNwmTcm.Start(&oNwmMon);
```

Uygulamalarda zaten bu kullanımlar hazır olarak mevcuttur. Baştan yazılmayacak ancak burada nasıl çalıştığı öğrenilmiş oldu.

OnwmTcc objesi benzer bir işi client olarak yapar. İstenilen sayıda IP-Port çiftine bağlantı kurar. Her bir IP-Port çifti session olarak adlandırılır ve unique session numarası ile erişilir. Her session için bir receiver thread bulunur. Receiver thread'lerin aldığı mesajlar uygulamanın transaction queue'suna basılır. Session receiver objeleri aynı zamanda connection statüsünü de kontrol eder, her bir session için primary ve secondary IP-Port verilebilir. Primary porta bağlantı kurulamazsa diğerine bağlantı denenir.

```
oNwmTcc.ConfigFormat(MSG_FRM_SPECIAL, LEN_FORM_STR, 4, 1);
oNwmTcc.ConfigBox(MSQ4_LOCAL, aCchPrm1.rSYS_GATE.gate_txn_box);
while (pSYS_GATE_CNN != 0)
{
      iRet = oNwmTcc.AddSsn(pSYS_GATE_CNN);
      if (iRet != ORC_OKI)
            __ERRN("N008065", "TCC add session error..", ORC_ERR);
      pSYS_GATE_CNN = aCchPrm1.GetNextConn();
}
iRet = oNwmTcc.Start(&oNwmMon);
```

Örnekte görüldüğü gibi SYS_GATE_CNN tablosunda tanımlı olan her connection için objeye bir session eklenmiştir. Obje içinde her session ayrı bir thread tarafınfan takip edilir ve bağlantı kurulmaya çalışılır. Bağlantı kurulunca recv döngüsüne girilir.

ConfigFormat ve ConfigBox fonksiyonları, ONwmTcm 'de olduğu gibidir. Farklı olarak burada port verilmek yerine session bilgileri verilmiştir.

Gelen mesajlar yine ONwmTcm'de olduğu gibi receive edilir ve gönderilir. Farklı olarak OBox header içinde address yerine session Id kullanılmaktadır. Cevap mesajı dönüyorsak gelen box içindeki session id saklanmalı ve send fonksiyonunda da aynı session id olmalıdır.

```
typedef struct OboxHdr
{
      ..
      OSI2 iFwdGateSsnIdx; → client durumunda adres yerine session id
```

Request mesajı gönderiminde ise sessionId değerini uygulama set etmektedir. Genellikle tek session kullanıldığı için hangi session'a göndereceğiz gibi bir sorun yaşanmaz, çoklu session'larda round-robin ile ayakta olan session'lara gönderim yapılır. Ancak uygulamalar business ihtiyaçlara göre session seçimi de yapabilir. Kısaca `iFwdGateSsnIdx` değişkeni set

edilerek hangi session'a mesaj gönderileceği belirtilir. Session numaraları SYS_GATE_CNN tablosunda yer alan numaralardır ve obje config edilirken set edilir.

OnwmTcc yada OnwmTcm kullanılması uygulamaya göre değişebilir. Örneğin Pos, Atm gibi device'lar genelde client konumundadır ve host TCP Server olarak çalışır (ONwmTcc), Bkm, Visa gibi kuruluşlar genelde TCP server konumundadır ve biz client oluruz vb.

OnwmAtm (socket array kullanımı)

Şu anda sadece ATM projesinde kullanıldığı için tüm OLBase'ler içinde bulunayabilir. Bu class TCP server olarak çalışır ve ONwmTcc class'ının bir alternatifidir. oNwmTcc objesi bağlanan her client için bir thread ayağa kaldırmakta ve açılan socket'i dinlemektedir. Ancak ATM server uygulamasında tüm ATM'ler sürekli bağlı olduğundan binlerce socket açılmaktadır. Bunların herbiri için ayrı thread ayağa kaldırılması gereksiz bir yük getirecektir. Üstelik bir socketten 20-30 saniyede bir mesaj geldiğinden verimsiz bir çalışma olur. Bu sebeplerden, socket library'sinin bir özelliği olarak 64 socket'i bir thread dinlemektedir. Böylece 20 thread açılarak 1000'den fazla ATM rahatça dinlenebilir. Bu class, benzer çok bağlantılı projelerde kullanılmak üzere geliştirilebilir.

Program config okunması (OAuxCfg, OSysReg, DB)

Programlar config bilgilerini değişik noktalardan okuyabiliyor.

1. İlk başta tüm bilgiler cfg dosyasından okumaktaydı.
2. Sonradan sadece DB connection parametrelerinin registry'den alınması yöntemine geçildi. Burada db parametresi değiştirmek için tüm cfg dosyaları editlenmesin gibi bir amaç güdülmüştü. Esasında programlar sadece açılışta cfg dosyası okuduğu için tüm cfg dosyalarının değiştirilmesi sorun oluşturmaz.
   Bu yöntemde registry'de hiç tanım yoksa db tanımları da cfg dosyasından alınmaktadır. Yani ilk yöntemle aynı olur.
3. Daha sonra bir bankanın geliştirmeleri sırasında bankanın isteğiyle cfg dosyası kullanımı tamamen kaldırıldı. Cfg dosyasındaki tüm bilgileri SYS_PRM_ONLINE tablosuna alındr. Sonraki kurulumlarda da bu yapı devam etti.
4. Tüm parametrelerin db'den okunması yönteminde program kodu da tablodan alındığı için, bir IP adresinden aynı program tipinden sadece bir tane açılabilmesi gibi bir kısıt oluştu. Bunu önlemek için bazı parametreler yine cfg dosyasına alınmaya başladı. GATE_NTW, GATE_CODE vb.
   Ayrıca, sadece bir programda kullanılması gereken bir parametrenin bile tabloda bir kolon olarak bulunması gerekiyor. Bunun yerine yeni eklenecek programa özel parametreler de cfg dosyalarına alınacak (log programında arşivleme süresi vb).
   *** bundan sonraki projelerde bu şekilde çalışılacak ***

Cfg dosyası programın çalıştırıldığı dizinde olmalıdır. Cfg Dosya ismi exe ile aynı isimde ve farklı uzantıdadır. Windows için dosya standart ini formatında olmalıdır. Ancak format

değişse bile, “Bölüm – Key – Data” hiyerarşisi korunmalıdır. Dosya formatı değişirse okuyan class da ona göre değiştirlebilir. Windows için değişikliğe gerek yok.

Cfg dosyaları okunması OAuxCfg class’ından bir obje oluşturularak yapılır. Aşağıda basitçe çörneği görülmektedir. Sadece obje tanımı yapılıp, GetOSC fonksiyonu ile string değerler, GetOSI fonksiyonu ile de integer değerler alınabilir. Cfg objesine herhangi bir config yapılmıyor çünkü sadece bulunduğu dizinden ve programlar aynı isimli cfg dosyasını okur.

```
OAuxCfg oAuxCfg;

oAuxCfg.GetOSC(SCT_SYS, KEY_SYS_GATE_NTW,  sTemp, "P");
oAuxCfg.GetOSC(SCT_SYS, KEY_SYS_GATE_CODE, iGateCede, 10);
```

Okuma fonksiyonlarına son parametrede default değer verilebilir. Bu durumda dosyada bulamasa bile default değeri kullanır.

Config bilgilerini programlardaki `aSysCfg` isimli global obje okur. Okunan değerler de bu objenin içinde public değişkenler şeklinde tutulur ve tüm programdan erişilir.

Global aSysyCfg objesi ve config bilgilerine erişim örneği:

```
ASysCfg     aSysCfg;

sprintf(sFileName, " %s.log", aSysCfg.sSvcName);
```

Bazı config parametreleri sadece açılışta okunurken bazıları periyodik olarak okunabilir. Örneğin Thread sayısı program akışı sırasında değiştirilemez. Config değiştikten sonra programın kapatılıp açılması gerekir. Aşağıda okunan tüm config parametreleri listelenmiştir. Sadece açılışta okunanlar static, diğerleri live olarak gruplandı:

Örnek static config parametreleri :

- DBM_HOST : DB TNS adı yada connection string
- DBM_USER : Database user
- DBM_PSWD : Database password
- SYS_MODE (RUN_AS_SVC)

Conversion Library Kullanımı (OAuxCnv)

Conversion ve bazı utility fonksiyonlarını içerir ve uygulamalarda sıkça kullanılır. Local data kullanmadığından global bir obje oluşturulabilir. Fonksiyonları birbirinden bağımsız olduğundan fonksiyon isim ve açıklamalarına bakılması yeterlidir.

Ocean sistemin şu tanımların ne anlama geldiği bilinmelidir : BCD, ASCII, EBCDIC, binary,hex, base64, base94, Str, int. Örneğin 123 sayısının değişik karşılıkları şu şekildedir:

Bcd → 0x01 0x23 (2 byte)

Bin → 0x7B (1 byte)

Str (ASCII) → "123" = 0x31 0x32 0x33 (3 byte)

Str (EBCDICI) -> "123" = 0xF1 0xF2 0xF3 (3 byte)

İnt → 123 (0x00 0x00 0x00 0x7B)

Hex → çok değişik şekillerde çevrilebilir. Herhangi bir buffer hex'e çevrildiğinde boyutu 2 katına çıkar.

0x7B → "7B" = 0x37 0x62 (2 byte)

0x31 0x32 0x33 → "313233" = 0x33 0x31 0x33 0x32 0x33 0x33 (6 byte)

0x01 0x23 → "0123" = 0x30 0x310x32 0x33

Base64 ve Base94 binary datanın ascii string şeklinde taşınması için kullanılan yöntemlerdir. Hex dönüşümü de aynı işi yapar ancak hex'e çevirdiğimizde boyut 2 katına çıkmaktadır. Base64 için uzunluk %33 oranında artar. Yani 3 byte'lık bir veri 4 karakter ile temsil edilir. Base94 için oran 4'e 5'tir. Base64 ve Base94'ün ayrıntıları için araştırma yapılabilir.

Crypto library kullanım (OAuxCrp)

DES librarysi thread safe değildir. Bir obje ile aynı anda tek encrypt yada decrypt yapılıyor olabilir. Zaten her thread için bir obje oluşturulmaktadır.

DES simetrik bir şifreleme yöntemidir. Aynı key ile encrypt ve decrypt işlemi yapılır. Des keyleri 8 byte uzunluğundadır. Şifreli ve clear data da 8 byte uzunluğundadır.

```
oAuxCrp.DES(fiDat, fiKey, foRes, fiMode);
```

fiData, fiKey ve foRes 8 byte'lık bir dataya point etmelidir. Library buffer'in içeriğine bakmaz yani buffer içinde 8 byte'dan büyük bir data olsa dahi ilk 8 byte kullanılır. Aynı fonksiyon hem encrypt hem de decrypt için kullanılır. Son parametre DES_MODE_ENC ise fiDat içinde Clear data input olarak girer ve foRes içinde encrypted hali döner. DES_MODE_DEC için fiDat içinde encrypted data gelir ve foRes içinde Clear data döner.

DES3 fonksiyonu da aynı işi yapar sadece fiKey 16 byte'lık bir key'e point etmelidir. Kullanımında bir fark yoktur.

Buffer üzerinde çalışmak : DES ve DES3 fonksiyonları ile 8 byte'dan daha büyük dataları şifrelemek yada çözmek data 8 byte'lık parçalar halinde işlenir. Burada CBC, ECB gibi yöntemler vardır.CBC yöntemi library içinde de yer almaktadır. Projelerde ECB'nin kullanıldığı yerler de vardır ancak uygulama kodları içinde handle edilmiştir.

DES library'sinin kaynağı : Bu library internetten bulduğunuz bir kod üzerinde oynamalar yapılarak bize uygun hale getirilmesi ile oluştu. Orijinal kodla ilgili bilgiler library içinde mevcuttur.

DES librarysini hızlandırmak: DES library'sinde key'in initialize edildiği bir bölüm vardır. Normalde Key değişmediği müddetçe bu initialize aşaması atlanabilir. Ancak biz her enc/dec işleminde bu init işlemini yapmaktayız. Özellikle buffer şifreleme gibi işlemlerde buradan bir miktar zaman kazanmak mümkün olabilir (çok büyük bir kazanç değil).

Floating point library kullanımı (OAuxDbl)

OAuxDbl sınıfının içinde local bir data kullanılmadığı için global bir obje oluşturulup kullanılmasında bir sakınca yoktur.

- Ocean uygulamalarında ODBL olarak geçen veri tipi sıkça kullanılmaktadır. Özellikle tutar alanları bu tipte tutulmaktadır. Ancak bu veri tipinin özelliğinden dolayı sayıları tam kesinlikte tutmayabilmektedir. Bizim açımızdan sayının değeri kesain olmakla birlikte hafızadaki tutuluş şekli değişebilmektedir.
  Örneğin 3.00 sayısı, hafızada 2,99999999999 yada 3,00000000001 şeklinde duruyor olabilir. Bu 2 değeri içeren 2 değişkeni karşılaştırdığımızda eşit olmadığı gibi bir sonuç dönecektir ve hesaplamalarda ciddi sorunlara yol açacaktır. Bu yüzden karşılaştırma, yuvarlama işlemleri için oAuxDbl objesini kullanıyoruz. Bu obje 0.0000000001 den daha küçük farkları yok saymaktadır. Bu değere epsilon değeri diyoruz.
- oAuxDbl içindeki karşılaştırma fonksiyonlar doğru ise B1, yanlışsa B0 döner. Örneğin IsGrt fonksiyonu, ilk değer ikinciden büyükse B1 döner. Aşağdaıki örnekte fPntEarned değişkeninin 0'dan büyüklüğü halinde if içine girer.

```
if (oAuxDbl.IsEqu(fPntEarned, 0.0) == B1)
```

- ODBL cinsinden bir sayıyı integer'a çevirirken de Round fonksiyonundan geçirmekte fayda vardır. Örneğin 2,99999999 sayısını integer'a attığınızda 2 olarak alır. Round işlemi sayıyı aynı zamanda değerin üst tarafına taşımaktadır. Böylece 3 değeri doğru olarak alınır.
- Round fonksiyonu verilen digitten itibaren yukarı, aşağı yada ortadan yuvarlama yapar. Bu fonksiyon epsilon değerlerini de dikkate alır.

Date Time Library Kullanımı (OAuxDts)

Sistemden zamanın değişik formatlarda alınası ve date/time formatlarının birbirleri arasında çevrimi için kullanılır. Local datası olmadığından global olarak kullanılabilir.

Ocean sisteminde genel olarak tarihler 8 haneli sayı olarak YYYYMMDD formatında tutulur. Saatler de 6 haneli sayı olarak HHMMSS formatında tutulur. Bu ikisinin birleşiminden 14 haneli bir format da kullanılmakta. Bazı durumlarda 14 hanenin sonun 2 yada 3 hane milisaniye eklenebilmektedir.

En çok kullanılan fonksiyon GetDts fonksiyonudur. SDts cinsinden struct doldurulur.

```
OAuxDts oAuxDts;
SDts sDts;

oAuxDts.GetDts(&sDts);
```

sDts yapısı içinde bahsedilen date/time/ms yanında julian date ve haftanın günü dönülmektedir. Julian data formatı YGGG şeklinde, yılın son hanesi ve yılın kaçıncı günü olduğunu gösterir.

```
iDate = 20110201
iJuli = 1032 -> 2011'in 32'inci günü.
```

Tarih / saati getiren ve dönüştüren fonksiyonlar genellikle okunabilir formattadır. Fonksiyon isminde de bu format belirtilmektedir. Örneğin `GetDtsMMDDHHMMSS`, `GetDateDDMMYYYY`, `TrnsYYDDD2YYYYMMDD` vb.

Bunların yanında saniye yada milisaniye sayısını dönen fonksiyonlar da vardır (GetDtsScs ve GetDtsMsc). Bunların formatı dışarıdan anlaşılamaz ancak geçen zamanı ölçmek için kullanılabilir. Aşağıdaki örnekte milisaniye cinsinden geçen zaman ölçülmüştür.

```
OSI8 iStartMsc;
OSI8 iEndMsc;
OSI8 iTimeElapsed;

iStartMsc = oAuxDts.GetDtsMsc();

...

iEndMsc = oAuxDts.GetDtsMsc();

iTimeElapsed = iEndMsc – iStartMsc;
```

Windows timer hassasiyeti : Windowsta timer 15 ms hassasiyetindedir. Yani geçen süre 3-4 milsaniye olduğu halde 0 yada 15 milisaniyeden biri görülebilir. Alınan milisaniye bilgilerinde belli milisaniyelerde birikme olduğu görülür. Eğer çok hassas ölçümler gerekiyorsa kullanılamaz. Genellikle toplam işlem süresi ölçüldüğü için 15 milisaniyeye kadar olan fark önemsenmiyor. ölçülen süre 15 milisaniye ise 1-29 arası bir değerdir denebilir.

Ek olarak bazı tarih utility fonksiyonlarından bahsedilebilir:

- DaySpan : iki tarih arasındaki gün sayısını getirir
- AddMonth : YYYYMMDD formatında bir tarihe belli sayıda ay ekler.
- AddDay : YYYYMMDD formatında bir tarihe belli sayıda gün ekler.
- AddSeconds : SDts struct içindeki tarih/saate saniye ekler. Tarih de değişebilir.
- IsWeekEnd : verilen tarihin haftasonu olup olmadığı

Tüm tarih fonksiyonları, artık yıl kontrolünü yapar. Artık yıl kontrolünde, 4 yılda bir şubat 29 gündür, 100 yıl başlarında 28, 400 yılda bir yine 29 gündür.

Compression Library Kullanımı (OAuxLzss)

Sadece bazı pos projelerinde kullanılmıştır. Sınırlı kullanımı olduğundan bahse konu edilmesine gerek yok. Kısaca verilen buffer'ı bir algoritma ile sıkıştıran ve aynı buffer'ı açabilen fonksiyonları içerir.Fonksiyon ad ve parametreleri amaçlarını açık bir şekilde göstermektedir. Burada kullanılan kod da internetten indirilmiş ve bazı düzenlemeler yapılmıştır. Zamanla libraryden çıkarılabilir.

Process Library Kullanımı (OAuxPrc)

Bir process ayağa kaldıran ve opsiyonel olarak sonlanmasını bekleyen bir class'tır. Bu class'ın system bölümünde olması daha doğru olabilirmiş.

"Execute" fonksiyonu path ve exe adını parametre olarak alır. Exeye geçecek comman line argumanları da ikinci parametredir. Win32 fonksiyonları kullanılarak exe ayağa kaldırılır. Bitmesi beklenir. Herhangi bir aşamada hata alınası durumunda hata dönülür. Fonksiyonun üçüncü parametresi exe'nin dönüş kodunu içerir. Tabii ki exe çalışıtırılamaz yada sonlandırılmazsa bu değişkenin bir anlamı yoktur.

"ExecuteShell" fonksiyonu ise sadece bir process'i ayağa kaldırır sonlanmasını beklemez.

Global Memory Queue Objesi kullanımı (OCllCrc)

Online uygulamalarda pek kullanılmamakla birlikte bazı batch projelerde kullanılır. Hafızada global bir queue objesi tutularak thread safe şekilde veri aktarımını sağlar.

Genelde producer/consumer tarzı thread yapısında kullanılır. Belli sayıda thread queue'ya item koyarken belli sayıda thread de queue'dan iyem alır. Çok dahice dizayn edilmiş bir yapı değil fakat iş görüyor. İtem boyu ve maximum item sayısı baştan verilir ve gerekli memory ayrılmış olur. Aşağıdaki örnekte içinde bir struct tipinde değişkenler tutulacaktır ve max item sayısı 1000'dir.

```
      oCllCrcAcquirer.Config(rBKM_SEGMENT.countno, sizeof(R_BKM_ISSUER));
```

Queue'dan aşağıdaki gibi item pop edilebilir. Item alınırsa ORC_OKI döner.

```
      iRet = oCllCrcAcquirer.Pop((byte *)&(sCLRBKMACQ));
      if (iRet == ORC_NON)
            // no item in the queue
      if (iRet == ORC_OKI)
            // process item
```

Queue'ya aşağıdaki gibi item eklenir. Yer yoksa ORC_NON döner.

```
      iRet = oCllCrcAcquirer.Push(&sCLRBKMACQ);
      if (iRet != ORC_OKI)
            // no place to add
      if (iRet == ORC_OKI)
```

```
// item add
```

Bu objenin 2 tür kullanımı olabilir.

1. Tüm itemları alabilecek şekilde queue oluşturulur. Tüm itemlar doldurulduktan sonra process edilmeye başlanır. Bu durumda Pop ve Push komutlarında ORC_OKI dışında bir değer dönmesi halinde çıkılır
2. Push ve Pop eden thredler paralel çalışır. Queue dolu ise Push yapan thread (ler) boşalması için bir süre bekler. Queue boş ise Pop eden thread (ler) bir item gelene kadar beklemeye geçerler. Bu tip kullanımda Push yapan thread'in işinin bittiği bir şekilde diğer threadlere iletilir.

Log Library Kullanımı (OLogSys)

Programların Merkezi yada local olarak loglamalarını sağlayan fonksiyonları içerir. Ayrıca pek çok makro kullanılmaktadır. Monitoring servera data gönderilmesi de bu obje üzerinden yapılır.

Programlar global bir oLogSys objesi oluşturulur ve tüm threadler bu obje üzerinden loglama yapar. Global log objesi tanımı :

```
OLogSys oLogSys;
```

Log objesi içinde loglama yapan her thread için data tutulur. Thread log datası, thread ilk defa log attığında otomatik olarak oluşur. Bu datanın geri verilmesi thread çıkmadan önce thread kodundan yapılmalıdır. Thread kodundan _TR() makrosu ile log datası geri verilmiş olur. Bu makro sadece thread çıkarken çağrılmalıdır. Bu makrodan sonra tekrar loglama yapılırsa data baştan oluşturulacaktır.

Program açılışında log objesi configure edilir. Çalışma sırasında da bu fonksiyonlar çağrılarak log biçimi dinamik olarak değiştirelibilir. ConfigDir fonksiyonu, local log dizini parametrelerini verir. İlk 2 parametre base ve sub dizinleri ifade etmekle birlikte sadece birinin verilmesi yeterlidir. Local log dizininin altına tarih, onun altında da uygulama bazında folderlar açılır. Son parametre opsiyoneldir ve uygulama bazında açılan folderin ismini set etmek için kullanılabilir. Bu parametre verilmezse, GateCode ve GateNtw parametrelerinden dizin adı üretilir.

```
iRet = oLogSys.ConfigDir(
          ""
       , aSysCfg.sLogPath
       , aSysCfg.iGateCode
       , aSysCfg.cGateNtw
       , "MyLogName");
```

ConfigCat fonksiyonu loglama yapılan satırların filtrelenmesine yarar. Genel olarak tüm satırları loglayacak şekilde ayarlıyoruz. Loglar çok fazla şiştiği için bazı kategoriler kapatılmak

istenebilir. Ancak programlardaki kategori bilgileri de çok sağlıklı olmayabilir. Hangi Log kategorilerinin açık olduğu config okunurken alınır.

```
iRet = oLogSys.ConfigCat(aSysCfg.iLogMask, aSysCfg.iLsiMask);
```

LogMask parametresinde tanımlanmış katrgoriler şunlardır. Pek çoğu kullanılmıyor. Genelde database yada business tipinde log atılır. Özellikle ISO8583 parse dataları ve raw mesaj dumplar logların büyük kısmını kapsadığı için gerekirse bu loglar kapatılabilir (LC_MSG, LC_ISO).

```
LC_NTW        // network log
LC_BOX        // msmq log
LC_FNC        // function start / end log
LC_ISO        // iso parse log
LC_BSN        // business log
LC_DBM        // database log
LC_CLL        // collection log
LC_FMT        // formatter log
LC_FPR        // function params log
LC_MON        // monitoring
LC_MSG        // iso message dump
```

Lsi değişkeni kritiklik seviyesini gösterir. Genelde bunlar arasında bir ayrıma gitmiyoruz.

```
LS_INF        // Information
LS_WRN        // Warning
LS_ERR        // Error
```

ConfigDst fonksiyonu ile hangi hedeflere log atılacağı belirtilir. Config aşamasında belirlenir. 3 hedefin hepsi de açık olabilir. Hedef msmq olduğu halde queue bilgileri verilmemişse, msmq loglama yapılmaz.

```
iRet = oLogSys.ConfigDst(aSysCfg.iLdsMask, 0);
```

```
LD_SCR        // To screen
LD_FSY        // To Local file
LD_BOX        // To Log Host's Queue
```

Msmq loglama seçildiği halde log server kapalıysa yada queue erişiminde bir problem varsa local dosyalara loglama yapılır. Servis modda iken ekrana basılsın seçeneğinin bir anlamı yoktur.

NOT : console modda ekrana log basılması performansı ciddi olarak düşürür ve işlem cevap süresini oldukça uzatır. Stress testi vs. yapılıyorsa ekrana log basılmamalıdır aksi halde tıkanmaya yol açar.

ConfigBox fonksiyonu ile merkezi loglama yapılacak olan server ve queue adı parametreleri verilir. Hedef olarak box verildiyse anlamlıdır.

```
oLogSys.ConfigBox(aCchPrm1.rSYS_HOST_LOG1.host_host,
aCchPrm1.rSYS_HOST_LOG1.host_txn_box);
```

ConfigMon fonksiyonu ile monitoring server adresi ve queue adı verilir. Paranetreler verilmediği takdirde monitoring servera veri gönderen fonksiyonlar hata veriri.

```
oLogSys.ConfigMon(aCchPrm1.rSYS_HOST_MON1.host_host,
aCchPrm1.rSYS_HOST_MON1.host_txn_box);
```

Log objesi config edildikten sonra loglama işleri genellikle makrolar ile yapılır. Pek çok makro loglara 1 satırlık data atar. Aşağıdaki örneği incelersek, ERR kısmı bunun bir error mesajı olduğunu, ERR'den sonra gelen N ise network error olduğunu gösterir. İlk parametre olan "B961234" IRC dediğimiz internal response Code değeridir. Çoğunlukla otomatik üretilir. İkinci parametre log basılacak olan satırdır. ERR'den önce gelen __ karakterleri fonksiyondan return edileceğini gösterir. Son parametre ise fonksiyondan return edilecek değerdir. ERR'den önce tek _ karakteri olursa, fonksiyondan dönülmeden akışa devam edilir. _ERRN makrosunda son parametre yer almaz.

```
__ERRN("B961234", "error line...", ORC_ERR);

_ERRN("B961234", "error line...");
```

Bazı makrolar hata dönmenin yanında işlem dataları içinde bulunan irc code ve response code değerlerini de set eder. Bunlardan IRCR makrosu hem log atar, hem irc ve rc değerlerini doldurur hem de fonksiyondan çıkar.

```
IRCR("B961234", "error line...", ORC_ERR);

IRCI("B961234", "error line...");
```

IRCR ve IRCRI makroları business thread içinden ve işlemi reddetmeye karar verirsek kullanılır. Örneğin hata alındığı halde işlem reddedilmeden akışa devam edilmesi isteniyorsa, __ERRx tipi makrolar kullanılmalıdır.

- _ERRx : loga hata satırı bas ve devam et
- __ERRx : loga hata satırı bas ve fonksiyondan çık
- _INFx : loga bilgi satırı bas
- _WRNx : loga warning satırı bas

x olarak verilebiliecek değerler şunlardır ve kategory mask ile sınırlandırılabilirler:

- D : database
- N : network
- S : System
- H : Hsm
- F : Format
- B : Business

Diğer makrolar ve kullanımları şöyledir:

ISOD : parametre olarak bir ISO8583 parser objesi alır. Öncelikle ISO mesajı dump halinde dosyaya basar, daha sonra mevcut olan tüm iso field'larını satır satır listeler. Bu makro ile basılan mesajlar loglarda önemli bir yer tutar. Yukarıda nasıl disable edildiği anlatılmştır.

ISOD makrosunun son parametresi mask parametresidir. Bu parametre 1 olarak verildiğinde, kritik alanlar ** karakterleri ile maskelenir. Kartno, track2, track1, cvv2, pin block gibi bilgilerin loglara açık olarak basılması güvenlik açığı oluşturduğundan production ortamında mask parametresi 1 olarak verilir. Burada önemli bir husus, cvv2 alanının yerinin mesajın tipine göre değişik olabileceğidir. Bu yüzden iso parser içinde hardcoded olarak, mesaj network tipine göre gizleme işlemi yapılır.

Loglardaki örnek ISOD makrosu sonucu aşağıdaki gibidir. Binary data içeren alanlar hex'e çevrilerek yazdırılır.

```
l20004:241 | [       ]  ------------------------------------------------------------------
l20004:241 | [       ]  Incoming ISO8583 Message MTI : 0800, LEN : 79 bytes
l20004:241 | [       ]  ------------------------------------------------------------------
l20004:241 | [MSGDUMP]  08002020000000E000009000000000033030333030303033393134323030303030
l20004:241 | [MSGDUMP]  303035303030300002485649373135313530323531323131383030363439012O
l20004:241 | [MSGDUMP]  00000000000034E4E00010000000000
l20004:241 | [       ]  ------------------------------------------------------------------
l20004:241 | [       ]  MTI = [004] [0800]
l20004:241 | [       ]  BM1 = [008] [2020000000E00000]
l20004:241 | [       ]  003 = [006] [900000]
l20004:241 | [       ]  011 = [006] [000003]
l20004:241 | [       ]  041 = [008] [00300003]|
l20004:241 | [       ]  042 = [015] [914200000050000]
l20004:241 | [       ]  043 = [040] [000248564937313531353032353132313138303036343901200000(
l20004:241 | [       ]  ------------------------------------------------------------------
```

BIND : bir buffer'i loglara hex stringe çevirerek ve bir satırda 16 byte olarak şekilde basar. Örnek bir BIND makrosu sonucu aşağıdadır. Bu makroda data maskeleme işlemi yapılmaz, çağıran program gerekiyorsa kritik dataları maskelemelidir.

```
143820:439 | [       ]  Total [269] byte(s)
143820:439 | [       ]  --------------------------------------------------------------------------
143820:439 | [       ]  0000: [06 01 00 32 50 53 31 34 39 37 38 37 00 00 00 10] [...2PS149787....]
143820:439 | [       ]  0016: [70 56 68 05 88 01 97 00 14 85 00 00 00 01 00 30] [pVh.ˆ.—.….....0]
143820:439 | [       ]  0032: [30 30 30 30 30 30 30 30 31 30 32 38 39 39 14 B4] [0000000010289 9.´]
143820:439 | [       ]  0048: [93 84 11 09 63 74 00 2D 12 03 20 11 93 91 74 50] ["„..ct.-.. .“‘tP]
143820:439 | [       ]  0064: [00 00 FF 01 01 02 04 35 2E 30 30 01 97 00 13 00] [..ÿ....5.00.—...]
143820:439 | [       ]  0080: [00 04 30 2E 30 30 20 01 01 69 9F 06 07 A0 00 00] [..0.00 ..iŸ.. ..]
143820:439 | [       ]  0096: [00 03 10 10 9F 26 08 4C EE 6A 81 D4 6A CE 38 9F] [....Ÿ&.LîjÐjÎ8Ÿ]
143820:439 | [       ]  0112: [27 01 80 9F 02 06 00 00 00 00 05 00 9F 37 04 1C] ['.€Ÿ.........Ÿ7..]
143820:439 | [       ]  0128: [18 19 27 5F 34 01 00 9F 36 02 01 5A 9F 10 17 06] [..'_4..Ÿ6..ZŸ...]
143820:439 | [       ]  0144: [01 0A 03 A4 00 00 0F 04 00 00 00 00 00 00 00 00] [...¤............]
143820:439 | [       ]  0160: [25 00 C0 AC C4 BD 82 02 5C 00 95 05 00 80 00 80] [%.À¬Ä½‚.\.•...€.€]
143820:439 | [       ]  0176: [00 9C 01 00 9F 34 03 41 03 02 9F 33 03 60 F0 C0] [.œ..Ÿ4.A..Ÿ3.`ðÀ]
143820:439 | [       ]  0192: [9F 41 04 00 00 29 51 9F 09 02 00 83 5A 08 49 38] [ŸA...)QŸ...ƒZ.I8]
143820:439 | [       ]  0208: [41 10 96 37 40 02 5F 2A 02 09 49 9F 1A 02 07 92] [A.–7@._*..IŸ...’]
143820:439 | [       ]  0224: [9A 03 09 11 04 9F 21 03 14 38 11 8E 12 00 00 00] [š....Ÿ!..8.Ž....]
143820:439 | [       ]  0240: [00 00 00 00 00 02 01 41 03 1E 03 02 03 1F 00 9F] [.......A.......Ÿ]
143820:439 | [       ]  0256: [53 01 52 00 BA BE CE CE 00 00 00 05 05 07 07 07] [S.R.º¾ÎÎ........]
143820:439 | [       ]  --------------------------------------------------------------------------
```

_TP() makrosu : Log objesi gelen satırları anında dosyaya yada log servera göndermez. Thread'e ait buffer dolduğunda yada thread _TP() makrosunu çağırdığında o ana kadar birikmiş olan loglar topluca gönderilir. Thread logları push ettiğinde, en sona PUSH Thread ile başlayan bir satır log daha eklenip gönderilir. Loglara yazılırken 2 thread'in log dataları birbirine karışmaz. İki Push Thread arasında tüm loglar aynı thread'e aittir. Push Thread satırınıda thread ID'si de yazılmıştır. Böylece bir threadin aktivitesi de izlenebilir. Yani thread ne zaman işlemi bitirmiş, ne zaman sonraki işleme başlamış gibi. Thread Id'ler yeri geldiğinde incelemede oldukça yardımcı olur.

```
143822:893 > PUSH Thread 3472
143822:924 | [SSS8300] <SYS><INF> [BMsgPrc::MsgCore] PROCESS START
143822:924 | [0008288] <NTW><INF> [BMsgPrc::ResolvePosMsg] Incoming Message [172.
143822:924 | [0008289] <NTW><INF> [BMsgPrc::ResolvePosMsg] TCP Header: F0016 RmtH
143822:924 | [0008291] <NTW><INF> [BMsgPrc::ResolvePosMsg] Remote Confirmation: +
143822:924 | [B008649] <BSN><INF> [BPrcConfirm::ConfirmRemove] Confirmation remov
143822:924 | [SSS8307] <SYS><INF> [BMsgPrc::MsgCore] PROCESS END:[99-99-99] GUID[
143822:924 > PUSH Thread 3548
```

Yukarıdaki örnekte, bir thread 6 kez log satırı basan makroları çağırmış (information tipinde), en sonunda ies TP makrosuyla bu logları msmq yada dosyaya göndermiştir.

TP makrosunun hiç log atılmadan çağrılmasında bir sakınca yoktur. Herhangi bir log gönderilmez. Hiç bir etkisi olmaz.

_TR() makrosu : Öncelikle _TP makrosunun yaptığı işin aynısını yapar yani logları push eder. Daha sonra çağıran thread için ayrılmış olan tüm log buffer'larını geri verir. Thread çıkmadan önce bu makroyu çağırmalıdır.

TLV data parser Library Kullanımı (OMsgTlv, OMixedTlv)

Programlar arasında data taşınması kısmında kısmen kullanımı verilmiştir. Parametrik şekilde TLV yada LTV formatında gelen dataları parse eden yada üreten class'tır. Parse edilen yada üretilecek tag-value çiftleri obje içinde tutulur.

OMixedTLV class, OMsgTlv class'ından türetilmiştir ve bazı taglerin length formatının farklı set edilebilmesine olanak sağlar.

Config fonksiyonu ile TLV formatı verilir. Burada verilecek bilgileri şunlardır:

- Önce tag (TLV), yada önce Length (LTV) gelebilir. İlk parametre bunu set eder.
- Tag uzunluğu : TLV datada Tag kısmının byte cinsinden uzunluğudur. Ber-TLV için anlamsızdır ve ignore edilir.
- Length uzunluğu : TLV datada length kısmının byte cinsinden uzunluğudur. Ber-TLV için anlamsızdır ve ignore edilir.
- Length tipi : length kısmı, Bcd, Binary, ascii string olabilir.
  Datanın BER-TLV formatında olduğu bu parametre ile set edilebilir.

- Length bilgisi TAG uzunluğunu da içeriyor mu? Genel kullanımda LTV tipinde length, tag bilgisini de içerir ancak içermesi zorunlu değildir.

Config aşamasında, MixedTLV objesi için gerekiyorsa bazı tag'lerin length tipleri ayrıca set edilebilir. Bu garip yapı bir bankada ortaya çıkan bir durumu handle etmek üzere eklendi. İlgili bankada length'ler ASCII string ve 2 uzunluktaydı. Bu durumda bir tag'in maximum uzunluğu 99 byte'ı geçemiyordu. Bu yüzden bazı tag'lerin uzunlukları BCD olarak set edilmekteydi. Mixed TLV objesi ile, bazı taglerin length formatları değiştirildi. Aşağıda örnekte görüldüğü gibi, 9600 tag'inin uzunluğu BCD, diğerleri ise ASCII dir.

```
OMixedTlv oTlvF48;

oTlvF48.Config(OFIRST_LEN, 4, 2, OLEN_TYPE_ASC, OTAGLEN_INC);
oTlvF48.SetCharType(CHAR_TYPE_EBCDIC);
oTlvF48.AddTagLenProp("9600", OLEN_TYPE_BCD);
```

Sadece MixedTLV objesinde, Config aşamasında SetCharType ile karakter tipi EBCDIC olarak set edilebilir. Bu durumda sadece Tag bilgileri için EBCDIC-> ASCII dönüşümü yapılır. Config aşaması dışında TLV ve mixed TLV objeleri aynı çalışır.

FldRes fonksiyonu ile verilen bir buffer'daki tag'ler ile obje doldurulur. Obje temizlenmeden yeni bir buffer parse edilebilir. Yeni gelen tag'lerin daha önceden değerleri varsa ezilmiş olur.

```
iRet = oTlvF48.FldRes(uBuffer, iBufferLen);
if (iRet == ORC_ERR)
      // err
```

TagAdd fonksiyonu ile TLV objesine bir tag eklenebilir. Parse edilmiş tag'ler olduğu halde yeni tag eklemek mümkündür. TagAdd ile aynı tag'i tekrar tekrar eklemek de sorun olmaz ancak, Obje içinde aynı tag'in tek değeri olabilir. TagAdd yada FldRes fonksiyonlarında eklenen tag(ler) daha önceden varsa değeri son verilen değer olur. TagAdd fonksiyonu ile byte array olarak bir değer ve uzunluğu verilir. Data içeriği önemsizdir.

```
oTlvF48.TagAdd("9401", uBff1, 6);
```

FldGen fonksiyonu ile TLV objesindeki tüm tag ve value bilgileri bir buffera yazılır. Buffer maximum uzunluğu da verilir. Dönüş değeri diğer fonksiyonlardan farklı olarak buffera yazılan byte sayısıdır. Obje içinde hiç tag yoksa, data üretilmeyeceğinden 0 döner. Üretilen buffer'da tag'lerin sırası belirsizdir. Tag'lerin eklenen sırada buffer'a yazılacağı garantisi yoktur. Çünkü tag'ler içeride bir map yapısında tutuluyor. Map yapısının implementasyonuna bağlı olarak eklenen sırada veriyor da olabilir. TLV class içinde değişik yapılar kullanılırsa bu sorun çözülebilir.

```
iLen = oTlvF48.FldGen((OUC *)sBff1, sizeof(sBff1));
if (iLen > 0)
      // iLen bytes of buffer generated
```

TagRem fonksiyonu ile TLV objesindeki bir tag silinir. Bunun amacı üretilecek TLV buffer'da bu tag'in yer almaması olabilir. Mevcut Bir TLV buffer'dan tag çıkarmak için önce FldRes ile buffer çözülür, TagRem ile tag çıkarılır ve FldGen ile yeni buffer oluşur. Oluşan bufferdaki tag sırası orijinalden farklı olabilir.

İşlemin başında TLV objesi temizlenmelidir. Aksi halde önceki işlemden kalan tagler hatalı işlenmesine yol açabilir.

```
oTlvF48.FldClr();
```

EMV data parser Library Kullanımı (OMsgEmv)

EMV chip datasının parse edilmesi ve üretilmesini sağlar. Mantık olarak TLV objesine benzer. Hatta TLV objesinin BER-TLV özelliği kullanılarak da aynı iş yapılabilir.

EMV parser'in ek özelliği olarak EMV tagleri önceden tanımlanmıştır. Parse aşamasında tag length uyumu kontrol edilir. Tanınmayan taglerin değerleri obje içinde saklanmaz ve output buffer'a yazılamaz. EMv objesinin bir özelliği de internal data ile çalışmayan fonksiyonlarının olmasıdır. Yani bir external bir buffer'i işler ancak internal datası hiç değişmez. EMV objesi parse edilmiş Emv tag'lerinden çok kullanılanları genel bir strucuture (OEmv) içindeki alanlara doldurarak programa dönebilir. Böylece programda her tag için ayrı get fonksiyonuna gerek kalmaz.

EMV objesi constructor fonksiyonunda tüm tag bilgilerini set eder. Genel olarak kullanılabiliecek tüm tagler eklenmeye çalışılmıştır. Burada fazla bir değişiklik olmuyor.

```
NewTagInfo("5A",    10,          ETL_VAR);
NewTagInfo("57",    19,          ETL_VAR);
...
```

*** Esasında programlara da custom tag ekleme yada bir tagin özelliğini değiştirme imkanı verilebilirmiş, NewTagInfo public yapılarak – bunu değerlendirelim ***

ETL_VAR tanımlanan taglerin uzunluğu değişebilir. Parametrede verilen uzunluk maximum uzunluktur. ETL_FIX tanımlanan taglerde ise verilen uzunluk fix uzunluktur ve daha kısa gelmesine izin verilmez.

oMsgEmv objesinin config fonksiyonu yoktur ancak buffer parse / generate ederken tag uzunluklarınını BCD / Binary olması seçilebilir. Henüz BCD length kullanan bir projemiz olmadı.

Buffer TLV objesine benzer şekilde parse edilir. Fark olarak length tipi config aşamasında değil parse aşamasında dinamik olarak verilir. Son parametre, tanınmayan tag'lerin gelmesi durumunda ne yapılacağını gösterir. filgnoreUnknownTags = 1 verilirse, bu tagler atlanarak devam edilir. 0 verilmesi durumunda parse kesilir ve hata döner. 1 verilmesi tavsiye edilir.

```
iRet = oMsgEmv.FldRes(pImf->icc_data, pImf->icc_data_len
```

```
      , FLD_LEN_BIN, 1);
if (iRet <= ORC_ERR)
      // parse failed.
```

Bu fonksiyon da diğer fonksiyonlardan farklı olarak parse edilen data uzunluğunu geri döner. Hata durumunda -1 (ORC_ERR) dönülür. Parse edilen uzunluk verilen buffer uzunluğundan az olabilir çünkü, bir tag'den sonra binary 0x00 gelirse, parse işlemi kesilir. Bu şekilde uzunluk düzeltilebilir. Hata dönülmesi tag'in maximum yada fixed uzunluğununu aşılmasından olabilir, tag uzunluğu gelen buffer'ı aşıyorsa olabilir. Ayrıca tanınmayan bir tag gelip ignore paranmetresi set edilmemişse hata döner. Hata dönülmesi durumunda, GetErrorTag() fonksiyonu ile en son kalınan tag görülebilir.

3 şekilde tag eklenebilir. Byte array ve length verilerek, string ve pad type verilerek, integer değer verilerek. String yada byte array verilirken dikkat edilmelidir. Geçen parametre string olduğunda son parametre length değildir. Explicit olarak buffer cast edilmesi tavsiye edilir.

```
oMsgEmv.TagAdd("CB", (OSC *)sStrVal, TAG_PAD_RIGHT);

oMsgEmv.TagAdd("CB", (OUC *)uBuffer, iBufferLen);

oMsgEmv.TagAdd("CB", iVal);
```

*** String ekleyen fonksiyon kullanılmıyor ve sakat. Bunun çıkarılmasında fayda var, yada en azından fonksiyon isminde string olduğunu belirtelim -TODO

2 şekilde tag değeri alınabilir. Mesajda geldiği şekilde byte array olarak yada hex string olarak. Her bir tag'in içeride mesajda gelen hali ve bunun hex string karşılığı tutulur. Hex karışılığı 2 kat uzunluktadır. Hex değeri getiren fonksiyon tag bulunamazsa boş bir stringe pointer döner. Hex değer içinde 0..9 A..F arası değerler olabieceğinden tag bulunadığı anlaşılabilir.

```
iRet = oMsgEmv.TagGet("CB", uBff1, &iLen);
if (iRet == ORC_OKI)
      // tag exists.

pCB = oMsgEmv.TagGet("CB");
if (p9F10[0] >= '0')
      // tag exists.

İlk örnekte 6 byte buffer gelmişse, 0x00 0x00 0x00 0x01 0x00 0x00
İkinci örnekte şu stringe bir pointer döner "000000010000"
```

Birde tüm tagleri struct array ile dönen bir fonksiyon vardır. Online uygulamalarda kullanılmıyor. İlk parametre struct array'inin başlangıç adresi, ikinci parametre ise arraydeki item sayısıdır. Item sayısı aşılırsa tag alınması kesilir ve ok dönülür.

```
OSI4 TagGet(STlvStr *fiData, OSI2 fiMaxCount);
```

Obje içindeki EMV taglerinden chip data buffer üreten 2 fonksiyon vardır. İlki kuralsız olarak tüm tagleri ekler. İkincisi ise belirli tagleri verilen sırada ekler. Özellikle switch tarzı uygulamalarda ikinci fonksiyon tercih edilmelidir çünkü dış kurumlarda kullanılacak tag

listeleri önceden belirlenmiştir. Aşağıdaki örnekte eskiden kullanılan FldGen fonksiyonu commentlenip yeni formatta sıralı üretime geçilmiştir.

```
FILLD(pImf->icc_data, 0); pImf->icc_data_len = 0;
//iRet = oMsgEmv.FldGen(pImf->icc_data, sizeof(pImf->icc_data));
iRet = oMsgEmv.FldGenOrdered(pImf->icc_data, pImf->icc_data_len,
      "82;84;95;9A;9C;5F2A;9F02;9F03;9F09;9F10;9F1A;9F1E;9F26;"
      "9F27;9F33;9F34;9F35;9F36;9F37;9F41;9F53;C0");
if (iRet <= ORC_ERR)
      IRCR("F963218", "CHIP fields generate error..", ORC_ERR);
pImf->icc_data_len = iRet;
```

FldGen yada FldGenOrdered fonksiyonlarına hedef buffer ve maximum buffer boyu geçilir. Dönüş değeri üretilen buffer uzunluğudur. Obje içinde hiç tag yoksa 0 döner.

FldGenOrdered fonksiyonuna geçilen listedeki taglerden değeri set edilmemiş olanlar atlanır. Bir tagin değeri olsa bile listede yoksa eklenmez. Listedeki sıra ile buffer üretilir.

TagRem fonksiyonu ile obje içindeki bir EMV tag'i çıkarılabilir. Eskiden gönderilmek istenmeyen tag'leri çıkarmak için kullanılıyordu. Ancak artık buffer üreten fonksiyona tag listesi geçildiği çok için önemi kalmadı. Yine de işlem bazında tag çıkarmak gereken durumlar olabiliyor.

Internal Data kullanmayan fonksiyonlar : EMV objesinde 2 fonksiyon sadece fonksiyon parametreleri üzerinde çalışır ve internal data üzerinde hiç bir etkisi olmaz.

FindMaxAvaliableLen fonksiyonu buffer'i parse etmeye çalışır ve parse edebildiği son noktayı döner. Bu fonksiyon bir buffer'in uzunluğunu düzeltmek için kullanılabilir. Örneğin Aşağıdaki örnekte buffer uzunluğu 2 fazla verilmiştir ve bu fonksiyonla düzeltilir.

```
uBuffer → 0x8a 0x05 0x01 0x02 0x03 0x04 0x05
iBufferLen → 100;

iBufferLen = oMsgEmv.FindMaxAvaliableLen(uBuffer, iBufferLen, FLD_LEN_BIN, 1)
```

sonuçta iBufferLen → 7 olur

Bu fonklsiyonun çıkış noktası bir bankada script hostunun dönüşte emv data uzunluğunu bazen 2 kat olarak dönmesiydi. Bunu handle etmek için eklendi.

BuffReplaceTag fonksiyonu ise verilen buffer'daki bir tag'i başka bir tag ile değiştirir. Sadece tag değerini değiştirir. Length ve value kısımlarına dokunmaz. Değiştirlen tag'in uzunluğunun farklı olmasını da handle eder. Garip bir fonksiyon olarak görülebilir. Normalde bu işi tagGet TagAdd fonksiyonları ile yapmak daha güzel olabilirdir. Ancak bir pos projesinde gerekli oldu. Aşağıdaki örnekte 74 tagi varsa tag değeri 9f9029 yapılır ve buffer uzunluğu 2 uzamış olur.

```
iRet = oMsgEmv.BuffReplaceTag(
      pImf->icc_data, pImf->icc_data_len, "74", "9F9029");
if (iRet > 0)
      pImf->icc_data_len = iRet;
```

Son olarak EMV parser objesinden çok kullanılan tag'lerin struct ile alınmasına bakalım. Normalde tüm tagler TagGet ile custom olarak alınabilir. ancak bu şekilde alınması daha kolaydır. oEmv struc içine genel tagler kopyalanır. En aşağıda struct içindeki tag değerleri kullanılmakta.

```
typedef struct OEmv
{
      OSC   t82[4+1];
      OSC   t84[32+1];
      OSC   t95[10+1];
      OSI4  t9a;
      ...


OEmv   oEmv;

iRet = oMsgEmv.FldRes(...
iRet = oMsgEmv.Fld2Str(&oEmv);

rTXN_ACQUIRER.f55_t9a                    = oEmv.t9a;
rTXN_ACQUIRER.f55_t9c                    = oEmv.t9c;
```

EMV objesindeki tag'lerin de işlem başında temilenmesi gerekir. Aksi halde bir sonraki işlemde hatalı değerler gösterilebilir

```
      oMsgEmv.TagClr();
      oMsgEmv.ResetStruct(&oEmv);
```

ISO8583 parser Kullanımı (OMsgIso, OMsgDef)

ISO8583 finansal mesajların iletilmesinde kullanılan standart bir mesaj formatıdır.

*** Devam etmeden önce ISO8583 formatı ve çok kullanılan alanlarla ilgili dökümanlara göz atılmalıdır. Yoksa buradan sonraki kısım anlaşılamayabilir. Ayrıca online kodlama için de bu araştırma gerekli olabilir.

ISO8583 mesajında standart olarak tanımlanmış alanlar vardır (örneğin Field2- Kartno, Field4-Tutar gibi). Bir ISO8583 mesajında 128 farklı alan gönderilebilir. Bitmap3 kullanılırsa sayı 192'ye çıkabilir ancak bitmap3 içindeki az sayıda alan, sadece VISA'da ve nadiren kullanılmaktadır. Standart alanların dışında private yada national kullanılabilecek alanlar ayrılmıştır, bunların içeriği de network'e göre değişebilir. Bir alanın içeriği mastercard ve VISA formatlarında çok farklı olabilir.

Uygulamada çok farklı ISO8583 mesaj formatları ortaya çıkabilmektedir. Kimisnde length bilgileri kimisinde data içeriğii farklıdır. Library pek çok mesaj formatını destekleyecek şekilde çalışmaktadır. Hazırda, Mastercard'ın online ve takas formatları, VISA baseI, değişik pos mesaj formatları, EST yada innova VPOS formatları, BKM, BonusNet, işbank Safir gibi formatlarla konuşuyoruz. Yeni bir format eklenmesi de kolaydır.

Örnek mesaj formatları (daha sonra ayrıntıya inilecektir):

BKM :

```
60000000003031303076 7F464128E2B60831363531353834393437393836373938333730303330303030303030303030303031333030303030303030303031333030...
```

VISA :

```
160102000000000023792300000000000000000000000100723D668128E09A12104761739001010010000000000000001235012616012129391 0...
```

Bonus : `6000000000F0F1F0F0767F44C128E3B400F1F6F5F1F2F8F0F3F0F0F0F5F6F3F0F5F4F9F0F0F3F0F0F0F0F0F0F0F0F0F0F0F0F2F0F0F0F0F0F0F0F0F0F0F0F0F0F2F0F0F0...`

POS :

```
60040122860200703C058020E0820416550472573905571500000000000000040000023170159 12...
```

*** POS için standart bir format yok. farklı pos formatları da olabilir.

ISO8583 mesajları incelendiğinde nümerik tanımlı alanlarda önemli bir fark görülecektir. Bazı mesajlarda bu alanlar BCD olurken, bazılarında ise string şeklindedir. Örneğin F3 alanı ISO standartlarında nümerik 6 olan bir alandır. Pos ve VISA mesajlarında bu alan 3 byte'lık bir BCD data ile ifade edilmiş. BKM ve bonus mesajlarında ise 6 byte karakter olarak ifade edilmiştir. Nümerik alanlar string olarak ifade edilirken, bu string ASCII yada EBCDIC olabilir. Bonusta EBCDIC, BKM'de ise ASCII string halindedir.

Örnek F3 alanı (000000 değerinin gösterimi):

Pos, Visa → 0x00 0x00 0x00

BKM, MC → 0x30 0x30 0x30 0x30 0x30 0x30 → "000000"

Bonus → 0xF0 0xF0 0xF0 0xF0 0xF0 0xF0 → "000000"

Variable length field'ların uzunluk bilgileri de değişik formatlarda verilebilir. BKM ve Bonus'ta length bilgileri yine string, Pos için BCD, Visa için Binary'dir.

Örnek F2 length bilgisi (16 değerinin gösterimi):

Pos → 0x16 (16)

Visa → 0x10 (16)

BKM, MC → 0x31 0x36 → "16" (16)

Bonus → 0xF1 0xF6 → "16" (16)

MsgCfg fonksiyonunun ilk 2 parametresi nümerik alanların formatını ve length formatını belirler. İlk parametre aynı zamanda string ve binary alanların formatına da etki etmektedir. (Normalde bunları ayırmak daha doğru olurdu).

- MSG_CHS_ASC → Nümerik alanlar ASCII string, string alanlar ASCII string, Binary alanlar byte array.
- MSG_CHS_BIN → Nümerik alanlar BCD, String alanlar ASCII yada EBCDIC, Binary alanlar BCD.
- MSG_CHS_EBC → Nümerik alanlar EBCDIC string, string alanlar EBCDIC string, Binary alanlar byte array.

String alanlar tüm mesajlarda aynı uzunluktadır. Formatı ise, library'nin bir özelliği olarak, nümerik olanlar ASCII yada EBCDIC string tanımlı ise, string alanlar da aynı formatta kabul ediliyor (yeni yazılacak library'de bunu değiştirebiliriz, gerçi aksi bir örnek görülmedi). Nümerik alanlar BCD ise, string alanlar default ASCII kabul edilir. Bu durumda string alanların EBCDIC olması isteniyorsa MsgCfg'nin beşinci parametresinde söylenir.

ISO objesinin config fonksiyonunda üçüncü parametre ise field tanımlarının hangi formatta olacağını içerir. 192 adet field için network bazında uzunluk ve tip tanımları farklılaştırılmıştır. Bunun olbase içinde olması bir handikap olabilir. Normalde her uygulama bunu dışarıdan verse daha düzgün olabilirdi.

Ön tanımlı field setleri şunlardır:

```
MSG_FPR_POS – POS
MSG_FPR_NSW – BKM online
MSG_FPR_IPM – Mastercard Takas dosyası
MSG_FPR_VIS – VISA BASEI
MSG_FPR_BNS – BonusNet
MSG_FPR_EST – EST VPOS formatı
MSG_FPR_MCR – Mastercard Online
```

Her bir field seti için 192 item'lik 2 array tanımlıdır, OFT_ ile başlayan Field Type array'i OFA ile başlayan length attribute arrayi.

```
static OSI4 OFT_POS[ISO_FLD_CNT + 1] =
{             OFT_BIN,
/* 001-005 */ OFT_BIN, OFT_BIN, OFT_BCD, OFT_BCD, OFT_BCD,
/* 006-010 */ OFT_BCD, OFT_BCD, OFT_BCD, OFT_BCD, OFT_BCD,


static OSI4 OFA_POS[ISO_FLD_CNT + 1] =
{             16,
/* 001-005 */ 16,   OFA_2LV,     6,     12,    12,
/* 006-010 */ 12,   10,          8,     8,     8,
/* 011-015 */ 6,    6,           4,     4,     4,
```

ISO parser verilen field seti ve parametrelere dayanarak mesajları parse eder ve mesaj üretir. Parse etme mekanizması, yada bu mekanizmanın düzgün olup olmadığı ayrıca incelenebilir. Bilinmesi gereken config parametrelerinin düzgün set edilmesidir.

Field değerleri obje içinde tutulur. Mesaj parse edildiğinde field değerleri dolacağı gibi, dışarıdan obje içindeki bir field set edilebilir. Mesaj oluşturulurken obje içinde mevcut olan değerler kullanılır. Bunun anlamı, parse edildikten sonra, field ekleme/silme/değiştirme yapılıp mesaj tekrar oluşturulabilir.

MsgRes fonksiyonu ile mesaj parse edilir. Öncesinde MsgClr yapılarak içeriğin boşaltılması gereklidir. İşlem başında da bu yapılabilir.

```
oMsgIso.MsgClr();
iRet = oMsgIso.MsgRes(pBin->uMsg + iIndex, pBin->iLen - iIndex);
if (iRet != ORC_OKI)
	// ERR
```

MsgRes fonksiyonuna örnekte olduğu gibi buffer ve buffer length verilebilir. Buffer MTI değeri ile başlamalıdır. MTI'dan önceki her türlü header (tpdu, visa header vs.) program içinde ayrıştırılmalı ve parser'a mti sonrası buffer adresi geçilmelidir. Örnekte iIndex değişkeni önden parse edilen datanın uzunluğu olmuştur.

MsgRes fonksiyonuna bunun dışında sBin, sIso gibi structlar verilebilir. Bu durumda TCP'den alınan struct direk olarak basılabilir. Bunlar pek tavsiye edilmemekle birlikte, sBin struct geçiliyorsa, 2 byte binary length'den sonra MTI gelen bir mesaj formatı olmalıdır. SIso struct kullanılıyorsa, 2 byte Binary length + 5 byte TPDU ve sonrasında MTI ile başlamalıdır.

MsgGen fonksiyonu ise MsgRes'in tam tersi gibi düşünülebilir. Header, mti gibi alanlar için söylenen her şey burada da geçerlidir.

```
iRet = oMsgIso.MsgGen(pBin->uMsg + iIndex, sizeof(pBin->uMsg) - iIndex, &iLen);
if (iRet != ORC_OKI)
	// ERR
```

ISO library içinde tutulan bir alanın alınması GetFld, GetFldOUC ve GetFldOSC fonksiyonlarından biri ile olabilir. Bu fonksiyonlar yerine popüler makrolar kullanılmaktadır.

```
OSC_POP (2,   pImf->pan);
OSI4_POP(3,   pImf->pcode_sys);
ODBL_POP(4,   pImf->txn_amt);
OUC_POP (55,  pImf->icc_data, pImf->icc_data_len);
...


--- Makro kullanmadan data alınası:
iRet = oMsgIso.GetFldOUC(57, sizeof(uBff1), uBff1, &iFldLen);
if (iRet != ORC_OKI)
```

OUC tipinde data alınıp verililen length de ayrıca alınıp verilir. Diğer alanlarda datanın sonlandığı kendiliğinde belli olmaktadır.

Library'e data verilmesi de benzeri fonksiyon ve makrolar ile yapılır:

```
OSC_PSH (2,  pImf->pan);
OSI4_PSH(3,  pImf->pcode_sys);
OUC_PSH (55, pImf->icc_data, pImf->icc_data_len);
```

Bunun dışında mesaj oluşturulurken filtreleme amaçlı SetFldExsPrf fonksiyonu kullanılır. Mesaj tipine bağlı olarak bazı alanların olmaması istenebilir. OC_SYS.SYS_GATE_FLD_PRF tablosundaki tanımlar kullanılarak bu fonksiyon tüm field'lar için çağrılır. Fonksiyona

geçilebilecek değerler M-Mandatory, C-Conditional, N-None'dir. N değeri geçilirse, field set edilmiş olsa bile gönderilmez.

```
oMsgIso.SetFldExsPrf(y + 1, rSYS_GATE_FLD_PRF.fld_prf[y]);
```

Hassas alanların maskelenmesi de ISO objesi tarafından yapılır. F2, F35, F52 gibi alanlarda standart maskeleme yapılırken, bazı alanlar network tipine göre custom olarak içeride maskelenmektedir. Loglara iso mesaj ve field'ları basılırken maskeli halleri alınarak basılır.

```
// Maskeli mesaja pointer döner. Mesaj Uzunluğu GetMsgLen ile alınabilir.
oMsgIso.GetMsgMasked();

// field maskeli olarak dönülür. Maskelenmeyen field'lar için orijinal datadır.
bRes = oMsgIso.GetFldOUCMasked(y, 512, uBff, &iLen);
```

FTP Library Kullanımı (ONwmFtp)

Bu library'i hiç bir projede kullanmadık. Bir bankada banka elemanları geliştirmişdi. İhtiyaç olduğunda incelenip kullanılabilir. Gerekirse üzerine eklemeler yapılabilir. Anlaşıldığı kadarıyla bir ftp servera login olunup file gönderme / alma işlemleri yapılıyor.

Seri Port Library Kullanımı (ONwmSerial)

Seri port üzerinden data gönderip almaya yarayan fonksiyonları içerir. Aktif olarak kullanıldığı bir proje olmadı. Belki seri porttan çalışan HSM'ler olsaydı ihtiyaç olabilirdi. Artık onlar da kalmadı. Online dışı uygulamalarda zaten seri port fonksiyonları internal olarak kullanılmaktadır. Bu class'in implementasyonunda win32 içindeki Api'ler kullanılmış. Config/Start/Send/Recv fonksiyonları isimlerinin gerektirdiği gibi çalışır. Diğer objelerde de sıklıkla kullanılan fonksiyonlardır.

Seri port haberleşmede Recv fonksiyonu istenen sayıda byte dönmeyebilir. Flow kontrol için programda ayrıca bir kodlama gerekir (STX, length vb.)

HSM Library Kullanımı (OHsmCnn)

Hsm Server programına basit ve genel bir yapı ile senkron bağlanılmasını ve mesajların gönderilip alınmasını sağlayan classtır. Thales Hsm server için geliştirilmiştir.

"Bu class'ta kullanılan mekanizmalar pek çok diğer class'ta benzer şekilde kullanılmaktadır (emv, nfc, h2h, vb.). Esasında tüm senkron bağlantılar için ortak bir altyapı olsaydı daha anlaşılır olabilirdi."

Programlar HSM cihazına direk bağlanmazlar. Arada Hsm Server (yada HSM driver) olarak adlandırdığımız başka bir online uygulama bulunur. Bu tip bir yapı kurulmasının sebebi, Hsm bağlantılarının kısıtlı olması ve bu bağlantıların hsm server'dan daha verimli kullanılabilmesidir. Ayrıca güvenlik açısından, her noktadan hsm'e erişilmesi yerine, sadece Hsm Server'dan erişilmesi kontrolü kolaylaştırır. Bir diğer nokta, hsm mesaj formatının business tarafındaki programlar tarafından bilinmesine gerek kalmaz. Tüm formatlamayı Hsm Server yapar. Eskiden seri port ile bağlanılan hsm'ler için de Hsm server gerekli

olabilmekteydi. Şu anda kullandığımız tüm Thales hsm'ler TCP çalışıyor. Gerçi Safenet hsm'lerin PCI kart olarak kullanılanlarında da aynı durum oluşmakta.

Programlarda global bir oHsmCnn objesi oluşturulur ve Hsm server ile tüm mesajlaşma bu obje kullanılarak yapılır.

"ASysGlb.h" (Globals)

```
OHsmCnn      oHsmCnn;
```

HSM servera gidişler senkrondur ve cevap gelene kadar business thread bekler. Normalde hsm'e gidilip gelinmesi birkaç milisaniyelik bir işlem olduğundan thread'lerin bu kadarlık bir süre beklemesi sorun olmaz. Bizim açımızdan database işlemleri gibidir.

Hsm kullanımından önce obje Config edilmelidir. 2 şekilde config edilebilir.

1. Hsm Servera msmq üzerinden gidilecekse ConfigMq fonksiyonu ile msmq olduğu söylenir. Hsm server'ın dönüş yapacağı queue adı da verilir.

   ```
   oHsmCnn.ConfigMq(local_ip, local_hsm_box);
   ```

   Msmq ile bağlanılacak Hsm server bilgileri eklenir. İstenen sayıda Hsm servera bağlantı kurulabilir. Yeri gelidiğinde anlatılacaktır.

   ```
   oHsmCnn.AddMqHost(hsm_server_ip1, hsm_Server_txn_box1, hsm_host_code1);
   oHsmCnn.AddMqHost(hsm_server_ip2, hsm_Server_txn_box2, hsm_host_code2);
   ```

2. Hsm Server'a TCP ile bağlantı kurulacaksa ConfigTcp fonksiyonu çağrılır. Tcp modda iken sabit bağlantı olacağı için queue tanımına gerek olmaz

   ```
   oHsmCnn.ConfigTcp();
   ```

   İstenen sayıda TCP ile bağlanılacak hsm server tanımı eklenir. IP/port bilgileri verilir.

   ```
   oHsmCnn.AddTcpHost(hsm_server_ip1, hsm_Server_port1, hsm_host_code1);
   oHsmCnn.AddTcpHost(hsm_server_ip2, hsm_Server_port2, hsm_host_code2);
   ```

Config edildikten sonra Start fonksiyonu ile bağlantı sağlanır. Arka planda Hsm cevap mesajlarını alacak bir thread ayağa kaldırılır. Bu thread config tipine göre queue'dan alabilir yada arka planda TCP ile bağlantı kurup mesaj almaya başlayabilir. TCP bağlantısını yönetmek de Hsm objesinin işidir. Tcp yada Mq üzerinden protokol olması programın işleyişini etkilemez. Buradan sonrası aynıdır.

```
iRet = oHsmCnn.Start();
```

TCP bağlantı seçeneği özellikle dinamik olarak ayağa kalkan ve kapanan programlar için daha uygundur. Çünkü bu programlarda dönüş queue'sunun set edilmesi çakışmalardan dolayı zor olabilir. Sabit online servis programlarda ise msmq üzerinden haberleşilmekte ve her

programın “Hsm Reply Queue” tanımı yapılmaktadır. Öte yandan Online servislerde de Tcp kullanılsa bir sakıncası olmaz.

Global Hsm objesini kullanmak isteyen her thread ayrı bir fonksiyonla objeye register olmalıdır. Bu fonksiyonu çağırmayan threadler hsm çağrısı yapamaz. Fonksiyon içinde thread’e özel bazı data allocationları yapılır. Global hsm objesi thread safe’dir. Aynı anda pek çok thread Hsm server ile haberleşiyor olabilir.

```
iRet = oHsmCnn.NewPrc();
```

Bundan sonra Hsm çağrıları ilgili hsm business fonksiyonlarını çağırarak yapılır. Aşağıdaki örnekte HSM objesi kullanılarak hsm’e Pin block çevrimi yaptırılmıştır. Tüm hsm fonksiyonları bu şekilde input/output parametreleri ile çağrılır. Parametre sayısı ve tipi fonksiyona göre değişir. Ancak ortak parametreler de vardır. İlk guid parametresinde işlemin guid değeri verilebilir. Sondan ikinci parametre her zaman timeout parametresidir. Cevap bekleme süresini milisaniye cinsinden verir. En sonraki Receive Header parametresi kullanılarak hsm cevap kodu gibi alanlar alınabilir.

```
HRcvHdr hRcvHdr;

iRet = oHsmCnn.TrsPINTPK2LMK(oBox.oHdr.iGUID
       , pImf->term_pin_key
       , pImf->pin_c0
       , "01"
       , pImf->pan
       , &iLmkPin
       , 3000, &hRcvHdr);
```

Hsm komut fonksiyonları, senkron olarak mesajı gönderip alır. Hsm’e bağlantı yoksa hata döner. Timeout süresi aılırsa hata döner. Buna benzer onlarca fonksiyon vardır.

“Hsm ile ilgili daha kapsamlı bir döküman hazırlanacak”

Fraud Connection Library Kullanımı (OSsfCnn, OSsfMsg)

Smartsoft Fraud hostuna bağlanıp, mesaj gönderip almayı sağlayan classtır. Fraud server’a TCP olarak connect kuran, connection’ları yöneten, mesajı gönderip alan objedir. Çalışma mantığı olarak Hsm objesine benzer. Fraud bağlantısı uygulamalar kısmında ayrıntılı anlatılacaktır. Burada olbase library’siyle alakalı konular anlatılacaktır.

Programlarda global bir oSsfCnn objesi oluşturulur ve Fraud server ile tüm mesajlaşma bu obje kullanılarak yapılır.

“ASysGlb.h” (Globals)

```
extern OSsfCnn oSsfCnn;
```

Program startup akışında Fraud objesi config edilir ve başlatılır. Bu obje arka planda TCP bağlantılarını açar ve yönetir. Görüldüğü gibi 2 tane Ip ve port çifti verilebilir. İkincisi yedek Fraud server’dır. Hsm için 2’den fazla server girilebiliyordu ve bunlar arasında yük dağılımı da

yapılmaktaydı. Fraud tarafında ilk server kapalıysa ikincisine bağlanılır. ConfigTov fonksiyonu ile default timeout süresi milisaniye cinsinden verilir.

```
oSsfCnn.ConfigTov(aCchPrm1.rSYS_HOST_FRD.host_tov_msec);
oSsfCnn.ConfigSsf(Fraud_ip1, Fraud_Port1, Fraud_ip2, Fraud_Port2);
iRet = oSsfCnn.Start(&oNwmMon);
if (iRet != ORC_OKI)
       // err
```

Fraud'a bağlanmak isteyen thread'ler Register fonksiyonunu çağırmalıdır.

```
iRet = oSsfCnn.RegSlot();
if (iRet != ORC_OKI)
       // err
```

Business Thread'ler ProtFrd fonksiyonu ile fraud tarafına mesaj gönderip alır. Bağlantı olmaması yada cevabın timeout süresinde gelmemesi durumunda bu fonksiyon hata döner

```
if (iMsgSrc == RSP_MSG_ISS)
  iRet = oSsfCnn.ProtFrd(pImf, &sImfEx, 0); // no wait
else
  iRet = oSsfCnn.ProtFrd(pImf, &sImfEx, -1); // default wait timeout
```

ProtFrd fonksiyonuna daha önce bahsedilen pImf pointeri geçilmiştir. Point edilne OImf structure çok kullanılan ve programlar arasında her zaman taşınan alanları içerir. Imf içinde olmayan field'lar için SSsfImfEx isimli internal bir structure daha oluşturulmuştur. Bu structure uygulamalar arasında taşınmaz sadece frauda gerekli Imf harici field'ları içerir. Fraud objesi içeride bu structure'lardan aldığı bilgilerle fraud mesajını oluşturur.

Gelen cevap mesajı da obje içinde parse edilip SSsfImfEx structure içindeki dönüş alanları set edilir. Dönüşte sRspCodeFrd alanı fraud response code değerini içerir.

ProtFrd fonksiyonunun son parametresi timeout değeridir. 0 verilirse hiç bekleme yapılmaz. Acquirer Fraud tarafına işlem sonuçları bu şekilde gönderilir. Genelde -1 verilir, bu durumda fraud objesinin default timeout süresi kadar beklenir.

Asenkron Library kullanımı (ONwmProtHst, ONwmProtMlc, ONwmProtMqWs, ONwmProtTsmGate)

Yukarıda bahsedilen Hsm objesi ile msmq üzerinden senkron mesaj alışverişi yapan bir protokolden bahsedilmiştir. İlk başladığımız bankada sadece bu class mevcuttu. Sonradan değişik ihtiyaçlar için bu class'ın kopyalanıp rename edilmesiyle aynı işi yapan çok benzer yapılar ortaya çıktı. Tüm yapılar için ayrı bir structure ve bir reply queue alanı tanımlandı. Notmalde bunların hepsini tek bir class içinde yönetmek mümkün olabilirdi. Bu şekilde daha basit yapılar olabilirdi. Ancak bu değişiklik için biraz geç kalınmış olabilir.

→ bu bölüm eksik kaldı. Library tekrar elden geçirilip devam edilecek.

Event Library Kullanımı (OSysEvt)

Event library threadler arası senkronizasyon için kullanılabilecek bir objedir. Aynı zamanda bir thread'in başka bir thread'I bir olay hakkında bilgilendirmesini (ayağa kaldırmasını) de

sağlar. Kimi zaman Event objesi bir mutex gibi de kullanılabilir. Bu kullanım özellikle birden fazla paralel resource'ın daha fazla sayıda thread tarafından kullanıldığı durumda faydalıdır. Tek resource'a eişimde mutex tercih edilir.

Tipik bir event kullanımı örneği aşağıdaki şekilde olabilir. Thread1 thread2'nin bir işi tamamlamasını bekleyecekse:

Thread1 – Wait for event (olay gerçekleşene, yada iş bitene kadar uyu)

Thread2 – Set event (olay gerçekleşti, diğer thread'e haber ver)

Thread1 – Wake-up (olay gerçekleşti, ayağa kalk)

Bu örnekte Event kullanımı olmasaydı, Thread1 olayın gerçekleştiğini poll'leyerek sürekli check edebilirdi, Olay gerçekletiğinde thread2 buna dair bir flag'I set ederdi. Thread1'de Sürekli pollemek cpu usage'I artıracağı için arada sleep fonksiyonları konup periyodik olarak polleme yapılacaktı. Bu durumda olay gerçekleştiği halde thread uykuda olacağı için belli bir beklemenin sonunda olayın gerçekleştiği anlaşılabilirdi. Event kullanımında ise bekleyen thread anında ayağa kaldırılmış olacaktır. Üstelik gereksiz mekanizmalar, thread'ler vs. ile uğraşmaya gerek kalmayacaktır.

Thread2'nin yaptığı işi Thread1 neden yapmıyor denebilir. Ancak thread1 ile aynı pozisyonda pek çok thread olabilir. Thread2 burada tek bir kaynağı kullanan thread olaabilir. Yada thread2'nin yaptığı iş belirsiz bir süre bloke edici bir iş olabilir, fakat biz thread1'in takılmasını istemiyor olabiliriz vb.

Tipik bir thread kullanımı aşağıda verilmiştir. Event objesinin her 2 thread tarafından görülebilmesi gerekir. Bunun için event objesi global olabilir yada paylaşılan bir struct içinde olabilir. Event objesinin Start işlemi kullanımdan önce yapılmalıdır. Stop işlemi de çıkışta yapılmalıdır. Thread1 ve thread2 deki kullanımlar verilmiştir.

```
OSysEvt oEvt;

oEvt.Start()
...


*** thread1 ***
       ...
       iRet = oEvt.Wait(iWait);
       if (iRet == ORC_OKI)
```

```
                // event occured
        else if (iRet == ORC_NON)
                // timeout
        ...

*** thread2 ***
        ...
        oEvt.Set()
        ...


...
oEvt.Stop()
```

Thread1 wait aşamasındayken, Thread2 Set fonksiyonun çağırırsa, Thread1'in wait fonksiyonu ORC_OKI döner. Thread1 wait fonksiyonuna timeout süresi vermiştir. Bu süre zarfında Thread2'den herhangi bir Set çöğrısı yapılmazsa, Thread1 Wait fonksiyonundan ORC_NON ile çıkar.

Thread1 wait aşamasına gelmeden thread2 Set fonksiyonunu çağırırsa wait fonksiyonu anında ORC_OKI döner. Bunun anlamı, daha önceden event set edilmişse ve eventi bekleyen kimse yoksa, bir sonraki bekleyene hatalı mesaj gitmesi demektir. Bu yüzden bun tip kullanımda Wait fonksiyonundan önce Event resetlenmelidir.

```
oEvt.EvtRst();
iRet = oEvt.Wait(iWait);
```

Event objesi mutex gibi kullanılıyorsa resetleme yapılmaz. Çünkü eventin setli olması erişilecek kaynağın sahipsiz olduğu anlamındadır. Wait fonksiyonu ile kaynağın sahipliği alınır, set fonksiyonu ile sahiplik geri verilir. Mutex olarak event kullanımı:

```
OSysEvt oEvt;

oEvt.Start();


// Wait resource (Acq)
iRet = oEvt.Wait(iWait);

        // use resource

// Give resource (Rel)
oEvt.Set();
```

Event objesinde timeout verilebilmesi mutex objesine göre bir üstünlük olabilir. Kaynak meşgulse tanımsız olarak beklenmez. Ayrıca birden fazla resource varsa herhangi bir resource'ın beklenmesi muliple event yapısı ile sağlanır. Multiple event durumu aynı objenin farklı bir şekilde kullanılmasıyla sağlanabilir. Aşağıdaki örnekte 10 tane paralel event vardır, bunun anlamı 10 thread aynı anda Wait fonkisyonundan girebilir demektir. Wait

fonksiyonunun dönüş değeri 0..9 arası ise 10 eventten birisi alnmıştır. Hangi eventin alındığı dönüş değeridir. Iş bittiğinde sadece alınan event set edilmelidir.

```
oEvt.Start(10);

iRet = oEvt. WaitMulti(iWait);

oEvt.Set(iRet);
```

Bu 10 eventten belirli bir tanesi de beklenebilir. Örnekte sadece ikinci event beklenmiştir.

```
oEvt.Start(10);

iRet = oEvt. Wait(iWait, 1);
```

Wait, Set,Rst, fonksiyonlarına event no verilerek arraydeki eventler için çalıştırılabilir. Single eventlerde event no sadece 0 olabilir. Single event multiple eventin özel bir halidir (count=1).

EVent library'nin etkin kullanımı, db library'sinde (thread haberleşmesi şeklinde) ve hsm server uygulamasında (çoklu mutex şeklinde) detaylı görülebilir.

File IO Library Kullanımı (OSysFio)

Dosya erişimini sağlayan fonksiyonları içerir. Projelerde bu class kullanılmadan direk dosya erişimleri de görülmektedir. Fopen, fwrite vb komutlarla. Çok fazla dosyalarla işimiz olmadığı için bu kullanımların şu anda çok bir önemi yok.

Aşağıda anlatılan standart dosya işlemlerinin yanında verilen bir path'i oluşturan bağımsız bir CreateDir() fonksiyonu da vardır. Bu fonksiyon obje data yada statüsünü etkilemez. Verilen parametrede root dizinden itibaren tüm dizinleri recursive olarak oluşturur. Path içinde dosya adı da olabilir. Dosya oluşturulmaz. En sondaki "\" karakterine kadar olan dizinler oluşturulur.

```
sprintf(sFilePathName, "c:\\SsfLog\\%d.log", sDts.iDate);
oSysFio.CreateDir(sFilePathName);
```

Okuma yazma yapmadan önce Config ve Open fonksiyonları çağrılır. Config fonksiyonu sadece dosya adını (full path) ve hangi modda açılacağını söylediğimiz fonksiyondur. Dosya modu şu anda windowstaki fopen fonksiyonuna geçilen değerleri kapsıyor. Ancak ileride bunu kendimiz handle edebiliriz aynı şekilde çalışır. Aşağıdaki modlar sıklıkla kullanılmaktadır. "r" → read, "w" → write, "a" → append. Bunlara ek olarak ikinci karakterde "b" → binary flag'i geçilebilir. Bu modlar uygulamalar için yeterli olmalıdır.

```
oSysFio.Config(sFile, "ab");
iRet = oSysFio.Open(B1);
if (iRet == ORC_ERR)
        // err
```

Open fonksiyonuna B1 parametresi geçilirse, dosya path'indeki bulunmayan tüm dizinler baştan oluşturulur. Özellikle verilen path'in olmama ihtimali varsa B1 verilmelidir.

Config ve Open aşamasından sonar okuma yazma yapılabilir.

```
iRet = oSysFio.Write(pData, iDataSize);
if (iRet == ORC_ERR)
	// err

iRet = oSysFio.Read(pData, iDataSize);
if (iRet == ORC_ERR)
	// err
```

*** Read fonksiyonu verilen boyut kadar okuyamazsa hata dönüyor. Bunun yerine okuduğu kadar byte dönse daha iyi olur – TODO

GetLastPos fonksiyonu, son yapılan Read/Write işleminden sonar dosyada hangi pozisyonda kalındığını gösterir. Append modda iken (dosyanın sonuna data yazılıyorsa) bu aynı zamanda dosya boyutuna eşittir.

Mutex Library Kullanımı (OSysMtx)

Mutex thread senkronizasyonu için kullanılan bir objedir. Global bir dataya erişirken yada bir resource kullanılırken bu objeden faydalanılır. Bir thread mutex objesinin Acq metodunu çağırdığında mutexi almış olur. Rel() fonksiyonunu çağırana kadar aynı mutex objesini başka bir thread alamaz. Diğer thread'ler aynı mutexin Acq satırına gelirlerse beklerler. Acq fonksiyonunda bir çok thread bekliyor olabilir. Her biri sırayla mutex'I alır ve bırakır.

```
OSysMtx oMtxReg;

oMtxReg.Acq();

... only one thread executes this block of code

oMtxReg.Rel();
```

Aynı mutex objesi farklı kod bloklarından kullanılabilir. Bu durumda tüm kod blokları aynı senkronizasyona tabii'dir. Aşağıdaki örnekte, bir thread "code block 1" içindeyken diğer threadler her 2 kod bloğuna da giremez. Bir –objeye- ait Acq ve Rel fonksiyonları nereden çağrılırsa çağrılsın aynı etkiye sahiptir.

```
OSysMtx oMtxReg;

oMtxReg.Acq();
... code block1
oMtxReg.Rel();

...

oMtxReg.Acq();
```

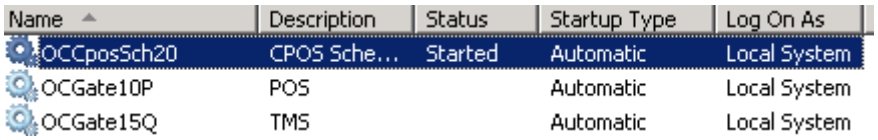
```
... code block2
oMtxReg.Rel();
```

Mutex objelerinde dikkat edilecek konu tüm thread'lerin aynı mutex objesine eriştiğinden emin olmaktır. Örneğin bir thread'in local datasında bir mutex oluşturulmasının hiç bir etkisi olmaz. Böyle yapılırsa her threadlerin kendi mutex objeleri olur ve birbirlerinden bağımsız olarak aynı kodu işletmiş olurlar. Uygulama kodlarındaki BMsgPrc objesinin içinde de mutex tanımlanmaz. Genel olarak mutex objesi, korunmaya çalışılan global data yada objectle birlikte tanımlanır ve kullanılır.

Servis Library Kullanımı (OSysSvc)

Online uygulamalar serverlar üzerinde arka planda windows servis olarak çalıştırılmaktadır. Bu şekilde sistem açıldığında ayağa kaldırılmakta ve herhangi bir user'in logon olmasına bağlı kalmamaktadır. Servis olarak çalışma esnasında console olmadığı için programın izlenmesi dosya ve tablolardaki loglardan yapılmaktadır.

Bir uygulamanın servis olarak ayakta olduğu "services" ekranından görülebilir. Tabii online uygulamalar için tablolardaki verilerden de ayakta olduğu anlaşılmakta.



Servisler default olarak "Local System" account ile çalışır. Servise sağ tıklayarak "Properties" seçildiğinde gelen ekrandan Logon user değiştirilebilir. Özellikle domain içinde olan makinalarda, başka bir makinadaki dosyalara erişilecekse domain user ile çalıştırılıp ilgili user'a yetki verilmesi faydalı olacaktır. Msmq yetkilerinde de domain user kullanımına geçilebilir.

Servisin register / unregister edilmesi :

Bir uygulama dışarıdan bir uygulama yardımıyla register edilebileceği gibi, kendi kendini de register edebilir. Burada kendi kendini eklemesi / kaldırması anlatılacak. Uygulamanın kendini register etmesi için ayrı bir fonksiyon yok. Uygulamalarda main fonksiyonuna command line argument gelirse bunu servis library'de geçiyoruz. Bu argument "i" olursa exe register ediliyor, "r" olursa exe unregister ediliyor.

Örnekte görüldüğü gibi argc ve argv değişkenleri servise parametre olarak geçilmiştir. Argv'nin ilk elemanı exe ismini içerir. İkinci elemanı ise ilk parametreyi. Servis objesi şu anda sadece ikinci elemanın değerine bakmaktadır.

```
int main(int argc, char* argv[])

oSysSvc.Config(aSysCfg.sSvcName, C_StartSys, C_StopSys
	, aSysCfg.sDependencies, aSysCfg.sStartupLogDir, aSysCfg.sDescription);

if (argc > 1)
{
	oSysSvc.Command(argc, argv);
	return ORC_OKI;
}
```

Command fonksiyonundan önce Config fonksiyonu çağrılmış olmalıdır. Bu fonksiyondaki bazı parametreler servisin özelliklerini belirlemektedir. Config fonksiyonu sadece içerideki bazı değişkenleri set eder. Servisin Start edilmesi öncesinde de config fonksiyonu çağrılmış olmalıdır. Bu fonksiyonu command mode'da olsak bile çağırıyoruz. Çapırmanın zararı yok. Comfig fonksiyonu parametreleri ve işlevleri şöyledir.

- fiName : Servis adı. Servis listesine bu adla çıkar. Exe adını servis adı olarak vermiyoruz çünkü aynı exe dosyasını farklı servis adları ile çalıştırabiliyoruz. Servis adında uygulama tipi ve kodu bulunur, online uygulamalar için standart bir servis adı yapısı vardır. Bazı utility tarzı uygulamalarda custom servis adları da verilebilmektedir.

  Örnek Servis Adı = "OCGate10P" (10 numaralı Pos Gate servisi)

- fiStart : Servis başlangıç fonksiyonu. Servis start edildiğinde bu fonksiyona girer. Bir "C" fonksiyonu olmalıdır. Genellikle bunun için ufak bir C fonksiyonu yazıp, Manager objesinin StartSys fonksiyonunu çağırıyoruz.
-

```
OSI4 C_StartSys()
{
	OSI4 iRet;
	iRet = aSysMgr.StartSys();
	_TR();
	return iRet;
}
```

- fiStop : Servis bitiş fonksiyonu. Servis stop edildiğinde yada servisin start aşamasında bir hata alınırsa bu fonksiyona girer. Bir "C" fonksiyonu olmalıdır. Genellikle bunun için ufak bir C fonksiyonu yazıp, Manager objesinin StopSys fonksiyonunu çağırıyoruz.

```
OSI4 C_StopSys()
{
	OSI4 iRet;
	iRet = aSysMgr.StopSys();
	if (iRet != ORC_OKI)
		aSysMgr.StartUpLog("StopSys Has Errors");

	_TR();
	return iRet;
}
```

- fiDepends : Servisin bağlı olduğu diğer servislerin listesi. Bu liste verildiğinde, servis başlatılmadan önce diğer servisler başlatılır. Şu anda MSMQ servisine bağlantı veriyoruz. Özellikle makina restart edildikten sonra msmq servisi başlamadan bizim servisler başlatıldığından açılışta hata alma olayları oluyordu. Bundan dolayı msmq servisine dependecy tanımladık.
  Msmq servisine tanımlanan dependency'nin bir etkisi de şudur. Msmq servisini restart ettiğinizde tüm ocean servislerini de ona bağlı olarak restart edecektir. Bazen teker teker restart etmek yerine bunu yapıyoruz.
  Birden fazla servise dependency verilebilir. Servis adları "#" karakteri ile ayılmış string olarak verilmelidir.
- fiLogPath : servis objesinin log atacağı dizindir. Atılan log ismi belirtilen dizinde "<service_name>_svc.log" şeklindedir. Nadiren sorun incelemek için gerekebilir. Ancak incelenen bu sorunlar servisin başlaması, takılması gibi sorunlar olabilir. Bir bankada windows2008'in service manager'i ile ciddi takılmas sorunları yaşamıştık, sonradan Norton güvenlik uygulamalarından kaynaklandığı anlaşıldı ve versiyon upgrade ile sorun çözüldü.
- fiDescription : Servis listesinde çıkan Description bilgisini burada set edebiliriz. Opsiyoneldir.

Servisi Register / Unregister etmek için en az Servis adı parametresi verilmiş olmalıdır. Dependency, log path ve description bilgileri boş verilebilir.

Servis olarak çalışan bir programdaki kod akışı aşağıdaki şekildedir. Config aşamasından sonra Dispatch fonksiyonuna girilir. Servis stop olduktan sonra dispatch fonsiyonundan çıkar. Config aşamasında verilen Start / Stop fonksiyonları içeriden çağrılacaktır.

```
int main(int argc, char* argv[])

oSysSvc.Config(aSysCfg.sSvcName, C_StartSys, C_StopSys
       , aSysCfg.sDependencies, aSysCfg.sStartupLogDir, aSysCfg.sDescription);
oSysSvc.Dispatch();
```

Thread / Thread Pool Library kullanımları (OSysThr, OSysTpl)

Online uygulamalar doğal olarak multithreaded bir yapıdadır. Main thread, programın start/stop akışlarını yürütür ve diğer threadleri ayağa kaldırır. Diğer tüm işler ayağa kaldırılan thread'ler tarafından yapılır.

Bir threadin başlangıç fonksiyonu C fonksiyonu olmalıdır. Zaten main() fonksiyonu da bir C fonksiyonudur. Thread start fonksiyonları aşağıdaki yapıda olmalıdır:

```
static void* FunctionName(void *fiPrm)
```

Thread'i oluşturan yerden fonksiyona tek bir parametre geçilebilir. Bu parametre genellikle bir struct yada objeye pointer olur. Ancak böyle bir zorunluluk yoktur. Parametre olarak bir integer yada 0 değeri geçilebilir pointer gibi geçilebilir. Geçilen değerin içeriği çağıran ile thread fonksiyonu arasında ihtiyaca göre belirlenir.

Thread'I başlatmak için şu fonksiyonlar kullanılır: Init fonksiyonu ile başlangıç fonksiyonu belirtilir. Ikinci parametresi, thread'e geçilen parametredir. Bu parametre thread fonksiyonuna aynen geçilir. İçeriği library açısından önemsizdir. Pointer tipinden olmakla birlikte, aynı alana sığabilen her şey geçilebilir. Üçüncü parametresi ise thread stack size'dır. 0 verilirse default stack size kullanılır. Ancak thread'in daha çok stack datası olacaksa artırılabilir.

```
OSysThr oSysThr;

oSysThr.Init(FunctionName, my_arg, 0);
iRet = oSysThr.Start();
if (iRet == ORC_ERR)
       // err
```

Init fonksiyonundan sonra çağrılan Start fonksiyonu thread'I ayağa kaldırır.

Stop fonksiyonu ile başlatılmış thread'in bitmesi beklenir. Dikkat edilmesi gereken husus, Stop fonksiyonu thread'in bitmesi için herhangi bir aksiyonda bulunmaz. Thread sonsuz bir loop içinde dönüyorsa, Stop fonksiyonu süresiz olarak bekler. Thread'in çıkması için ayrıca bir flag tutulmalıdır yada thread belirli bir işi tamamlayıp sonra çıkabilir.

```
oSysThr.Stop();
```

Stop fonksiyonu hiç çağrılmayabilir. Bu durumda program main fonksiyonu çıktığında thread otomatik olarak ölmüş olur. Kontrolsüz bir şekilde çıkış yapmıyoruz her zaman thread'lere stop sinyali gönderip, sonra da stop fonksiyonu ile bitmelerini bekliyoruz.

Stop fonksiyonu çağrılmadan thread'ler çıkıyorsa, thread handle açık kalır ve handle sayısında artma meydana gelir. Dinamik thread üreten yerlerde soruna yol açabilir. Bu tip yerler için thread'I başlattıktan sonra ClearHandle fonksiyonu çağrılır ve thread bağımsız hale getirilir. Ancak bu thread'i artık Stop fonksiyonu ile bekleyemeyiz.

```
iRet = oSysThr.Start();
oSysThr.ClearHandle();
```

Thred Pool objesinin kullanımı thread objesine benzer. Tek farkı birden fazla threadin aynı fonklsiyondan birlikte başlatılmasıdır. Ekstra parametre olarak thread sayısı da geçilir. Pool içindeki tüm thread'lere aynı parametre geçilmektedir. GEnellikle bu parametre çağıran objenin pointer'i olur.

```
OSysTpl oTplWrk;

oTplWrk.Config(FunctionName, my_arg, 0);
```

```
oTplWrk.Config(ThreadCount, B1);
iRet = oTplWrk.Start();
if (iRet != ORC_OKI)
	// err

iRet = oTplWrk.Stop();
if (iRet == ORC_ERR)
	// err
```

Thread ve Thread pool kullanımına uygulama kodlarında ayrıntılı inilecek

ONLINE UYGULAMALAR

Uygulamanın Start – Stop akışları

Tüm uygulamalar doğal olarak main fonksiyonundan başlar. Proje ile aynı isimdeki cpp dosyasında main fonksiyonu yer alır. Ama burada hiç bir işimiz olmayacak. Main fonksiyonunda hiç bir şey yapılmadan AsysMgr objesine gidilir.

```
iRet = aSysMgr.Start(argc, argv);
```

AsysMgr objesi programın açılış/kapanışını yöneten objedir. Tüm global objelerin, threadlerin yönetimi buradan yapılır.

AsysMgr::Start fonksiyonuna baktığımızda en başta bir configurasyon okunduğu görülür. Programın config bilgileri değişik yerlerden okunabilmektedir. Registry, cfg dosyası, db..

```
iRet = aSysCfg.Start();
```

Genel olarak Database bağlantı parametreleri registry'den alınır. Bu parametreleri DB server, Kullanıcı adı ve password' dir. Diğer config bilgileri ise kimi zaman cfg dosyasından, kimi zaman da OC_SYS.SYS_PRM_ONLINE tablosundan alınabilir.

Config bilgileri alındıktan sonra 3 tip akış vardır.

1. Uygulamaya parametre verilerek, servisin register, unregister edilmesi ve çıkılması. Aşağıdaki kod parçasından anlaşılacağı gibi programa arguman verildiğinde normal çalışma akışına girmemektedir. Sadece aSysSvc objesi kullanılarak servisin register edilmesi, silinmesi gibi operasyonlar yapılır ve çıkılır. geçersiz bir argüman verilmesi durumunda hiç bir aksiyon alınmadan çıkılmış olur.

```
oSysSvc.Config(aSysCfg.sSvcName, C_StartSys, C_StopSys
	, aSysCfg.sDependencies, aSysCfg.sStartupLogDir, aSysCfg.sDescription);

if (argc > 1)
{
	oSysSvc.Command(argc, argv);
	return ORC_OKI;
}
```

2. Programın console olarak çalışması. Console olarak çalıştırmak Exe'ye çift tıklayarak yada shell komutları ile exe'nin çalıştırılmasıdır. Bu durumda bir console ekranı olur ve program output'u bu ekrandan görülür. Yazılımcıların debug modda programı trace etmesi de aynı şekildedir.
   Exe'yi console olarak açıyorsak, configrasyon aşamasında da console olduğu belirtilmelidir (SYS_PRM_ONLINE tablosundaki run_as_svc alanı 0 olmalıdır). Aksi halde program açılmayacaktır.

NOT : Console olarak açılmış bir uygulama, kullanıcı logoff olunduğunda kapanır.

Örnek : Console modda program akışı

```
if (aSysCfg.iSvcMode == NO_SVC)
{
	iRet = StartSys();
	if (iRet == ORC_OKI)
	{
		sprintf(sBff1, "OCEAN Service [%s] Started..", aSysCfg.sSvcName);
		MOND(ML_TYP_INF, ML_CAT_NTW, ML_SEV_INFORMATION, sBff1);
		aSysCmd.Start();
	}
	iRet = StopSys();
	if (iRet != ORC_OKI)
		aSysMgr.StartUpLog("StopSys Has Errors");
}
```

Örnekte görülen StartSys fonksiyonu çağrıldığında program initialize edilir ve threadler ayağa kalkar. Start işlemi başarılı olursa, aSysCmd.Start fonksiyonu çağrılıyor. Bu fonksiyon while döngüsü içinde kullanıcıdan input bekler. Böylece main thread'in çıkması önlenmiş olur.

AsysCmd objesi console ekranında "exit" yazılıp enterlanmasıyla döngüden çıkar. Bundan sonra StopSys fonksiyonu çağrılır. Bu fonksiyon da oluşturulmuş olan thread'lere durmaları için sinyal gönderir ve bitmelerini bekler. Ardından main thread'den ve programdan çıkılmış olur.

3. Programın servis olarak çalıştırılması. Program windows servis olarak çalıştırılır. Servis olarak çalıştırmadan önce servisin install edilmesi gereklidir (madde 1). Programı servis olarak açıyorsak, configrasyon aşamasında da servis olduğu belirtilmelidir (SYS_PRM_ONLINE tablosundaki run_as_svc alanı 1 olmalıdır). Aksi halde program açılmayacaktır.

   Servis olarak açılışta kod akışı:

```
oSysSvc.Config(aSysCfg.sSvcName, C_StartSys, C_StopSys
	, aSysCfg.sDependencies, aSysCfg.sStartupLogDir, aSysCfg.sDescription);

...
else
{
	oSysSvc.Dispatch();
}
```

   oSysSvc.Config fonksiyonu ile Olbase servis objesine start ve stop fonksiyonlarını veriyoruz. Buradan sonrası artık OsysSvc ve windows servis dispatcher'ın işidir. Servisini start edilirken StartSys, Stop edilirken de StopSys çağrılır.

Görüldüğü gibi console ve servis modlarında başlatım farklı olsa bile ortak bir altyapı kullanılmaktadır. Başlangıöta StartSys fonksiyonu, Bitişte ise StopSys fonksiyonu çağrılmaktadır.

StartSys ve StopSys fonksiyonları birbirinin tersi olarak düşünülmelidir. StartSys içinde başlatılan herşeyin StopSys içinde bitirilmesi gerekir. Aksi halde resource leak'ler oluşacaktır.

Program açılıp kapanırken aşağıdaki şekilde log atılan satırlar görülür:

```
aSysMgr.StartUpLog("StartUp Log for %s   [" __DATE__ "][" __TIME__ "]",
aSysCfg.sSvcName);
```

Bu satırlar loglama sisteminde bağımsız olarak programın düzgün açılıp kapandığını görmek amaçlı ekstra loglardır. Startup logları config aşamasında alınmış olan Starup Log dizinine yazılır (SYS_PRM_ONLINE tablosunda STARTUP_LOG_DIR kolonu). Bu dizinde aynı zamanda service objesinin internal logları da yer alır. Her program için 2 log dosyası oluşur:

<<ProgramName>>_start.log : aSysMgr objesinin logu
<<ProgramName>>_svc.log : oSysSvc objesinin logu

Program açılmıyorsa yada kapanma aşamasında takılıp kaldıysa bu logları incelemek sorunu anlamayı sağlayabilir.

Programda TCP/IP kullanılacaksa aşağıdaki fonksiyon çağrılır. Winsock library'sinin initialize rutinleri çağrılmış olur. Bu çağrılmadan winsock fonksiyonları hata alır.

```
iRet = ONwmTcp::TcpEnvInit();
```

Program cache'lerinin başlatılması ve ilk kez load edilmeleri için aşağıdaki rutin çağrılır. Cachhe'lerin nasıl yenilendiği ayrıca cache bölümünde anlatılmıuştır.

```
iRet = StartCache();
```

AsysWrkDsp objesi diğer programların ayakta olup olmadığını izleyen, aynı zamanda mevcut uygulamanın ayakta olduğunu bildirmekle görevli thread'dir. İleride içeriği ayrıntılı anlatılacaktır. Burada çağrılmasının sebebi, log ve mon hostları ayakta mı kontrolü içindir. Örneğin log server ayakta değilse açılışta merkezi sisteme log gönderilmez.

```
iRet = aSysWrkDsp.Check4Startup();
```

Bu fonksiyon Log ve Monitiring serverlar ile msmq bağlantısının kurulmasını sağlar. Server ip adresleri ve log & mon queue adları parametre olarak verilir. Bir programa 2 tane log ve 2 tane monitoring uygulamasının bilgileri verilebilir. İlk kapalı olduğunda ikincisine bilgi gönderiminde bulunur. Log sistemi ayrıca anlatılacaktır.

```
iRet = StartLogBox();
```

Aşağıdaki fonksiyon ile programın doğru makinada çalıştırıldığı kontrol edilir. OC_SYS.SYS_GATE tablosunda programın IP adresi bulunmaktadır. Buradan alınmış olan IP

adresi ile çalışılan makinanın IP adresi karşılaştırılarak farklı ise hata verilir. Makinanın birden fazla IP adresi olabilir, bunlardan biri ile match etmesi yeterlidir.

```
iRet = ONwmTcp::CheckIPAddr(aCchPrm1.rSYS_GATE.gate_host);
```

Açılışta işlem kuyruğu boşaltılabilir. Config aşamasında set edilen bir parametredir SYS_PRM_ONLINE tablosundaki PURGE_QUEUE = 1 ise kuyruk temizlenir. Uygulamanın tipine göre bu parametre set edilir. Uygulamanın gelen mesajı sonradan işlemesi kabul edilebilir ise queue temizlenmez (örneğin log server için bu şekilde çalışma mantıklıdır).

```
iRet = StartPurgeQueue();
```

HSM server programı ile bağlantı ayağa kaldırılır. Hsm Server ile msmq yada TCP/IP olarak konuşulabilir. Hsm istekleri senkron olarak gönderen thread tarafından beklenir. Program içindeki tüm Hsm isteklerini yöneten global “oHsmCnn” objesidir.

```
iRet = StartSec();
```

Program Hsm serverdan başka pek çok yardımcı uygulamayla daha senkron olarak konuşabilir. StartHosts fonksiyonunda bu uygulamalar da başlatılır. Core bank ile konuşuluyorsa globa “oNwmHst” objesi kullanılır, Fraud ile konuşulurken global “oSsfCnn” objesi kullanılır, Emv server ile haberleşilirken global oNwmEmv kullanılır vb. İleride değişik hostlar için benzer objeler eklenebilir.

```
iRet = StartHosts();
```

```
NOT: her host için ayrı obje kullanmak yerine tek bir senkron haberleşme objesiyle
bunlar halledilebilirdi. İleride böyle bir yapıya geçilmesinde fayda var.
```

Config adımları neredeyse tamamlanmıştır. Artık thread pool’lar ayağa kaldırılabilir.

```
iRet = StartWorkers();
```

Her bir thread pool için globalde AsysWrk ile başlayan bir obje bulunur. Worker threadlerin çalışmaları ayrıca incelenecektir.

1. aSysWrkMon : monitoring thread. Belli aralıklar uygulama statüsünü monitoring’ gönderir.
2. aSysWrkAdm : admin thread. Dışarıdan gelen administrative requestleri işler.
3. aSysWrkTxn : Transaction thread POOL. Business işini yapan threadler.
4. aSysWrkDsp : Dispatcher thread. Diğer uygulamaların ayakta oldup olmadığjnı kontrol eder.
5. aSysWrkCch : Cache thread. Cache refresh ve switch etme işlemlerini yapar.

Program kapanırken, StopSys fonksiyonunda yukarıda yapılmış olan işlerin tersleri yapılır.

Cache kullanılması ve refresh edilmesi

Programlar bazı tabloları hafızada tutar ve periyodik olarak refresh eder. Uygulama içinde bu tablolar için db erişimi yerine hafızadaki değerler alınabilir. Programda cache ile ilgili objeler şunlardır.

1. Tablo objeleri : Bir tablodan select edip hafızada tutan objeler.
2. ACchPrm : Tablo objelerini içeren global cache objeleri. İçindeki tablo objelerinin doldurulmasını da sağlar.
3. AsysWrkCch : Periyodik olarak cache refresh akışını başlatan thread objesi.

Tablo classları ORMapper uygulaması ile kodu üretilen T_ ile başlayan class'lardır. Tablo kodlarının nasıl üretildiği ilgili bölümde anlatılmıştır. Cache'lenecek tablolar için T_CCH_TABLE_BASE class'i extend edilir. Cache Tablo classlarında 4 metod implemet edilmelidir. Config ve Clear zaten tüm tablo objelerinde bulunan metodlardır. Config fonksiyonunda cache query'si ve varsa parametreleri tanımlanır. CacheLoad fonksiyonu select query'sini çalıştırıp gelen sonuçları hafızaya dolduran fonksiyondur. CacheFree ise hafızadaki bilgileri siler. CacheLoad'dan önce CacheFree çağrılır.

Cache'lenen bilgiler tablo objesinin içinde saklanır. Otomatik üretilen kodlarda std::map yapısı bulunur ancak daha sonra bunu değiştirebilmekteyiz. Özellikle az sayıda data olan tablolarda, yada cache içinde karışık metodlarla arama yapılacaksa std::vector yapısı kullanılabilir. Kayıt sayısı aşağı yukarı belliyse, array kullanımı da mümkündür.

Örnek cache doldurma şekilleri.

Vector :

```
typedef std::vector <R_SYS_GATE> V_SYS_GATE;
V_SYS_GATE v_SYS_GATE;
...
while ( ((iRet = oCrsCch.ResFetch()) == ORC_OKI) && (bIsUp == B1) )
{
	...
	v_SYS_GATE.push_back(r_SYS_GATE);
}
```

Map

```
typedef std::map <std::string, R_SYS_PATH> M_SYS_PATH;
M_SYS_PATH m_SYS_PATH;
...
while ( ((iRet = oCrsCch.ResFetch()) == ORC_OKI) && (bIsUp == B1) )
{
	...
	sprintf(sCchKey, "%03d%03d", r_SYS_PATH.app_code_src
		, r_SYS_PATH.app_code_dst);
	m_SYS_PATH[sCchKey] = r_SYS_PATH;
}
```

Map şeklinde doldurulmuş bir cache objesinde Key ile aratmak mümkündür. Çok sayıda item içeren ve unique bir key kullanabilecek datalar için uygundur. Vector yapısında for döngüsü

içinde tüm itemlar taranır ve kritere (yada kriterlere) uyan item'lar bulunabilir. Bunu avantajı aramada bazı kolonların opsiyonel olabilmesidir. Map içinde tüm itemleri de dolaşmak mümkündür ancak böyle kullanım olacaksa vector tavsiye edilir.

Map cache'de item aranması

```
OSI4 T_SYS_PATH::CacheFind(R_SYS_PATH *fioStr)
{
	...
	sprintf(sCchKey, "%03d%03d", fioStr->app_code_src, fioStr->app_code_dst);
	iFindCol = m_SYS_PATH.find(sCchKey);
```

vector cache'de item aranması

```
OSI4 T_SYS_GATE::CacheFind(R_SYS_GATE *fioStr)
{
	...
	for (iGateIdx = 0; iGateIdx < v_SYS_GATE.size(); iGateIdx++)
	{
		pSYS_GATE = &v_SYS_GATE[iGateIdx];

		if (fioStr->gate_code == pSYS_GATE->gate_code &&
			fioStr->gate_ntw == pSYS_GATE->gate_ntw)
		{
```

Tüm Cache objeleri ACchPrm içinde member objeler olarak bulunurlar. ACchPrm class'inin 2 tane global instance'I oluşturulur. aCchPrm1 ve aCchPrm2. Bu objeler istenilen bir anda içlerindeki tüm cache objelerinin doldurulmasını sağlayabilirler ve cache'in bir kopyası lıkarılmış olur. Aşağıdaki örnekte aCchPrm1 içindeki tüm tablo objelerinin cache dataları dolmuş olur.

```
ACchPrm aCchPrm1;

iRet = aCchPrm1.Config();
iRet = aCchPrm1.Update(CCH_ALL, 1);
iRet = aCchPrm1.Fill();
```

2 tane global cache kullanılmasının sebebi, cache update sırasında diğer objenin kullanılabilmesini sağlamaktır. Cache update edilme zamanında pasif olan global cache objesi doldurulur, bu sırada programlar aktif olanı kullanmaya devam ederler. Cache refresh tamamlandığında, pasif olan global cache objesine switch edilir.

Program içinde aCchPrm1 yada aCchPrm2'den hangisinin güncel ve aktif olduğunu anlamak için global bir pointer tanımlanmıştır ve o anda aktif olan global cache objesini gösterir. Business objeleri bu pointer yardımıyla aktif cache objesini kullanırlar.

```
ACchPrm aCchPrm1;
ACchPrm aCchPrm2;
ACchPrm *pCchPrm;


pCchPrm = &aCchPrm1;
```

```
(yada) pCchPrm = &aCchPrm2;
```

ACchPrm içinde cache tabloları şu şekilde kullanılır.

- ORMapper ile üretilmiş ve gerekirse editlenmiş olan tablo classları projeye eklenir.
- Tablo class header'I include edilir. Genllikle proje header dosyasında

```
#include "T_SYS_MBR_PRM.h"
```

- ACchPrm class'inda ilgili tablo objesi tanımlanır.

```
class ACchPrm
{
	...
	private:
		...
		T_SYS_MBR_PRM		SYS_MBR_PRM;
```

- ACchPrm classinin config metodunda, ilgili obje listeye eklenir. CCH_STT seçilirse, bu tablo sadece ilk program açılışında cache'e doldurulur, CCH_DYN seçilirse, periyodik update'lerde de bu objenin içeriği doldurulur. Static cache objelerine pointer yardımıyla değil, aCchPrm1 objesi üzerinden erişilmelidir.

```
OSI4 ACchPrm::Config()
{
	...
	sCchTblDef.pName = "SYS_MBR_PRM";
	sCchTblDef.iLoadType = CCH_DYN;
	sCchTblDef.pCacheTable = &SYS_MBR_PRM;
	v_CCH_TABLE_LIST.push_back(sCchTblDef);
```

  Bu işlemler yapıldığında cache objesinin refresh edilmesi garanti edilmiş olur. BElli periyodlarla aCchPrm1 ve aCchPrm1 global cache objeleri içinde sırasıyla bu tablo objesi cache'lenir.

- Cache içindeki dataya ulaşmak için ACchPrm içine bir metod eklenir. Tablonun içeriğini anlatan makul bir fonksiyon ismi verilir.

```
OSI4 ACchPrm::FindSysMbrPrm(R_SYS_MBR_PRM *fiPrm)
{
	...
	iRet = SYS_MBR_PRM.CacheFind(fiPrm);
```

- Bundan sonra uygulama içinden cache datasına fonksiyon yardımıyla ulaşılabilir:

```
iRet = pCchPrm->FindSysMbrPrm(&rSYS_MBR_PRM_ISS);
```

Cache refresh edilmesi ve 2 global cache arasında switch edilmesini yöneten ASysWrkCch class'idir. Bu classın da global bir objesi vardır. Program başlangıcında bir thread ayağa kaldırılır ve periyodik olarak cache refresh eder. Cache Refresh aralığı SYS_GATE yada SYS_HOST tablosundan alınır.

gate_cch_upd_type : cache update aralık tipi

I (Interval) – belli aralıklarla

E (Exact) – Günün belli bir saatinde.

gate_cch_upd_time

I (Interval) tipinde : dakika cinsinden cache update aralığı

E (Exact) tipinde : HHMM şeklinde saat ve dakika olarak

ASysWrhCch classinin CacheSwap fonksiyonunda cache refresh ve switch edilmesi görülebilir. Aktif cache aCchPrm1 ise aCchPrm2 doldurulur ve pointer ona set edilir. Aktif cache aCchPrm2 ise tam tersi bir akış uygulanır.

```
if (pCchPrm == &aCchPrm1)
{
	iRet = aCchPrm2.Update(CCH_DYN, 0);
	if (iRet == ORC_OKI)
		pCchPrm = &aCchPrm2;
}
else
{
	iRet = aCchPrm1.Update(CCH_DYN, 0);
	if (iRet == ORC_OKI)
		pCchPrm = &aCchPrm1;
}
```

Periyodik cache objesi dışında dışarıdan administrative komut gönderilerek de cache refresh edilebilir. Programın admin queue'suna "C3" şeklinde bir komut gönderildiğinde cache refresh başlatılmış olur.

Admin queue'suna cach refresh komutu göndermek için "Sistem izleme" ekranlarından gate'e tıklanıp cache refresh butonu seçilebilir. Alternatif olarak bir utility exesi yapılarak queue'ya ilgili komut basılabilir.

Uygulama numaraları ve servis adları

Her uygulamanın programda tekil bir numarası vardır. Şu anda 99'a kadar program numarası verilebiliyor (Bunu 3 hane yapsak iyi olacak - TODO). Program numarası her türlü program tanımında, yönlendirmelerde, log ve monitoringde kullanılmaktadır ve değişmez.

Program numaraları verilirke genellikle işlevlerine göre gruplama yapılır.

- 10-19 : Acquiring gate'ler
- 20-29 : switch tarzı uygulamalar
- 30-39 : authorisation uygulamaları
- 50-69 : Host uygulamaları (yardımcı servisler)
- 90-99 : takas ve batch uygulamalar.

İçeride bir kontrol olmamakla birlikte bu şekilde bir dağılım tavsiye edilir. Özellikle development ortamlarında, her geliştirici kendi program setini açacağı için tüm numaralar karışık olarak kullanılabilmektedir. Prod ortamında dikkat edilebilir.

Aynı uygulama tipinden çift tanım yapıldığı durumda, genellikle uygulama numaraları ardışık olarak verilir. 10,11 : posgate, 30,31 : card authorisation gibi.

Aşağıdaki tabloda örnek Uygulama tipleri ve örnek servis adları verilmiştir.

TİP	AÇIKLAMA	EXE ADI	ÖRNEK SERVİS ADI
C	Authorization Gate	OCAuth.exe	OCGate30C
P	Pos Gate	OCGatePOS.exe	OCGate10P
B	BKM Gate	OCGateNSW.exe	OCGate20B
O	OKM Gate	OCGateNSW.exe	OCGate24O
F	EST Gate	OCGatePOS.exe	OCGate12F
I	Şube Gate	OCGateBNK.exe	OCGate14I
M	Management Gate	OCGateMNG.exe	OCGate16M
V	Vısa Gate	OCGateVIS.exe	OCGate26V
H	HSM Host	OCHsmd.exe	OCHost50H
L	Log Host	OCLogger.exe	OCHost50L
N	Monitoring Host	OCMonCore.exe	OCHost53M
E	BKM Reversal Host	OCEvent.exe	OCHost54E
K	Task Host	OCTask.exe	OCHost55K

Bankaya göre değişik uygulama tipleri yada tip farkları olabilir. Monitoring uygulamasının tipi bir bankada "M" iken, bir başkasında "N" olmuştur. Şube gate bazı bankalarda Ivr gate olarak da geçebilir. Bazı bankalarda VPos'lar ile poslar aynı gate'e gelirken, bazılarında ayrıca bir EST gate vardır. (*** Yeni projelerde VposGate yapalım. Ayrıca Posgate fiziksel poslar hariç hiç bir kod içermesin ***).

Listede aynı exe isminin değişik uygulama tiplerinde kullanılabildiği görülmektedir. Bu tip uygulamalar config dosyasında tip bilgisini de okurlar. Alınan tip bilgisine göre çalışmalarında farklılıklar olur.

Servis adları ise şu şekilde oluşturulur:

Örnek - OCGate10P

- OC → Ocean (Standart)
- Gate → "Gate" yada "Host" kullanılır
- 10 → Program ID (Unique)
- P → Pos

Gate, işlem akışı boyunca doğal olarak geçilen programları ifade eder. Host ise daha çok gate uygulamalarının aldığı servisleri ifade eder. Host uygulamalarına tüm işlem gönderilmez, sınırlı data gönderilir ve senkron olarak cevap beklenir. Gate'ler kendi aralarında ise asenkron konuşur. Yani gönderilen mesajın cevabı beklenmez. Cevap mesajı ayrı bir akışa girer.

Programların ayakta ve çalışır durumda olduğu kontrolü

Programlar birbirlerine MSMQ ile mesaj gönderdiği ve msmq asenkron çalıştığı için karşı programın ayakta olup olmadığını bağlantı yoluyla anlama imkanı yoktur. Bunun yerine database kullanan bir mekanizma geliştirilmiştir. Programlar periyodik olarak database'de kendi bilgilerini update etmekte, diğer programlar bu bilgilerden ayakta olma kontrolünü yapmaktadır. Bu iş için OC_SYS.SYS_GATE ve OC_SYS_SYS_HOST tablolarında 2 kolon kullanılır.

ONL_STAT -> 0- program kapatıldı, 1-program ayaktadır.

LAST_ONL_TIME -> programın son update zamanı YYAAGGHHMMSS

Program düzgün olarak kapatıldığında, ONL_STAT alanı 0 yapılacağı için LAST_ONL_TIME'ın bir önemi kalmaz. ONL_STAT = 0 olmasından program kapalı olduğu anında anlaşılır.

Ancak program istem dışı olarak kapanırsa (task manager'dan kill edilmesi, göçme vb), ONL_STAT 1 olarak kalacaktır. Bu durumda LAST_ONL_TIME değişkeninin belli bir süre update görmemesinden programın kapandığı anlaşılır. LAST_ONL_TIME alanın database saati ile güncellenir ve select edilir. Böylece serverlar arasında zaman farklarından dolayı sorun yaşanmamış olur.

Programlar 4 saniyede bir tabloya update ve select atmaktadır. Bir program göçtüğünde en kötü ihtimalle 8 saniye içinde diğer programlar bunu anlayacak ve mesaj göndermeyi sonlandıracaktır. Biraz marj bırakılmış ve statüsü açık olup, 10 saniye içinde update görmüş programlar aşağıdaki query ile açık kabul edilmektedir. Zaten plansız program kapanmaları prod ortamında görülmemektedir.

```
"SELECT HOST_CODE, 0 TBL_IDX FROM OC_SYS.SYS_HOST "
"WHERE LAST_ONL_TIME > TO_NUMBER(TO_CHAR((SYSDATE - INTERVAL '10'
SECOND), 'YYMMDDHH24MISS')) "
"AND ONL_STAT = 1 AND STATUS = 1 AND QUE_LOCK = 0 "
```

Program düzgün kapandığı halde diğer programların hemen bunu anlayamacağı gibi bir düşünce akla gelebilir. Buna önlem olarak ONL_STAT = 0 yapıldıktan sonra program 5 saniye boyunca çalışmaya devam eder. Bu sürede diğer programlar yeni mesaj göndermeyi bıracaktır. Aynı zamanda programın queue'sunda bulunan mesajlar da bu süre içinde tamamlanmış olacak ve mesaj kaybı yaşanmayacaktır.

*** Log ve monitoring uygulamaları için ek bir önlem daha alınmıştır. Bu uygulamalara çok fazla mesaj gelmektedir ve mesajların birikmesi durumunda sistemde tıkanmalara yol açabilmektedir. Bu uygulamalarda queue mesaj sayısı kontrolü de yapılmaktadır. SYS_HOST tablosuna aşağıdaki 2 alan daha eklenmiştir

TXN_QUE_COUNT → txn queu'da birikmiş mesaj sayısı

QUE_LOCK → 1 ise queue mesaj sayısı kritik seviyeyi aşmıştır.

Log ve monitoring hostları, statülerini update ederken aynı zamanda queue'da bekleyen mesaj sayılarını da kontrol edip tabloya yazmaktadır. Bu mesaj sayısı kritik seviyelere ulaştığında diğer uygulamalar 2 şekilde aksiyon alabilir.

1. Alternatif host varsa ve bir problem yoksa oraya gönder
2. Alternatif host yoksa, local dosyalara logla

Log ve monitoring hostlarının queue'larının boş olması da yetmiyor. Gönderen uygulamanın outgoing queue'sunda da mesajlar birikiyor olabilir. Bu durum da zamanla msmq'nun işlemez hale gelmesine sebebp olabilir. Bu yüzden uygulamalar periyodik olarak outgoing queue mesaj sayılarını kontrol eder ve belli bir seviyey ulaştığında, log ve monitoring datası gönderimini keser. Loglar local dosyalara atılır. Aşağıda outgoing queue kontrolünü yapan kod parçaları görülüyor. Outgoing queue'daki mesaj sayısı 100'ü aştığında ORC_NON döner.

```
// Check Primary Log Server
iRet = CheckHostEx(&aCchPrm1.rSYS_HOST_LOG1);
...

OSI4 ASysMgr::CheckHostEx(R_SYS_HOST *pSYS_HOST)
{
	iRet = oNwmMmq.OutgoingCount(pSYS_HOST->host_host
		, pSYS_HOST->host_txn_box, &iMsgCount, &iByteCount);
	if (iRet == ORC_OKI)
	{
		if (iMsgCount > 100)
		{
			return ORC_NON;
		}
	}
```

Yukarıdaki kontrollerden dolayı zaman zaman merkezi loglama yerine local log dizininde loglar oluştuğu görülebilir.

Uygulamalar arası PATH tanımları

Bir uygulamanın hangi uygulamalarla konuşacağı path tanımları ile yapılır. Path tanımları, önceki bölümde anlatılan ayakta olup olmadığı kontrolü ile birbirini tamamlar. Path tanımları OC_SYS.SYS_PATH tablosunda yapılır. Aşağıdaki önek path tanımlarında, 10 numaralı posgate'in 24,26,30,50 nolu programlara gidebileceği belirtilmiştir. Tablodaki APP_TYPE alanlarının program açısından bir anlamı yoktur. Sadece APP_CODE alanları kullanılır.

GUID	STATUS	LASTUPDATED	MBR_ID_SRC	APP_TYPE_SRC	APP_CODE...	MBR_ID_DST	APP_TYPE_DST	APP_CODE_DST	PRIORITY
4	1	1	1	GP	10	1	GZ	26	1
3	1	1	1	GP	10	1	GO	24	1
8	1	1	1	GP	10	1	SH	50	1
6	1	1	1	GP	10	1	GC	30	1
21	1	1	1	GF	12	1	GO	24	1

Path tanımlanması mutlaka diğer programa gidileceği anlamına gelmez. Örneğin bir Pos gate'den başka bir posgate'e path tanımı yapılsa bile bunun bir anlamı yoktur ve boşuna tanım yapılmış olur. Ancak Authorisation gate'e eklenen bir path, onus kredi kartı geçitiğinde kullanılabilir.

Aynı tipte 2 adet programa path tanımlıysa, bu programlar arasında yük dağılımı ve yedekleme yapılabilir. PRIORITY kolonu bu iş için kullanılır. Priortiy düşük ise path daha önceliklidir. Aynı priority'li programlar arasında yük dağılımı yapılır. Priority değeri yüksek olan path'ler ancak diğer path'ler kullanılamadığında kullanılır (yedek path). Uygulamaların açık/kapalı olduğunun anlaşılması daha önceki bölümlerde anlatılmıştır.

ÖRNEK1: 10 nolu posgate'den 30 ve 31 nolu auth gate'lere path tanımlı ve priority'leri 1 olsun. Bu durumda gelen işlemler sırasıyla bu uygulamalara gönderilir. Uygulamalardan biri kapalı olduğunda tüm işlemler diğerine gider.

1	1	GP	10	1	GC	30	1
1	1	GP	10	1	GC	31	1

ÖRNEK2 : Benzer bir örnekte pathlerden birinin priority değeri 2 yapılmıştır. Bu durumda tüm işlemler 30 nolu auth gate'e gönderilir. 30 nolu gate kapalıysa 31 numaralıya gönderilir. 30 nolu gate tekrar açıldığında yönlendirme geri döner.

1	GP	10	1	GC	30	1
1	GP	10	1	GC	31	2

Bir uygulama tipine mesaj gönderilmek istendiğinde, path'ler priority'e göre taranır. Eğer tüm uygulamalar kapalı ise hata dönülür.

Genellikle path tanımları çift olarak yapılır. Her programdan en az iki tane çalıştırılmaktadır. Programın tipine göre aktif-aktif yada aktif-yedek çalışma yapılabilir. Aktif –aktif çalışan uygulamalarda (authorisation, hsm gibi uygulamalar bu şekildedir) hiç bir mesaj kaybı olmadan programların güncellenmesi mümkün olur.

Yukarıdaki örneklerde program değişikliği şu şekilde kayıpsız yapılabilir.

a. 30 nolu auth gate'e stop komutu gönderilir.
b. 30 nolu auth gate SYS_GATE tablosundaki ONL_STAT alanını 0 yapar. Diğer gate'lerin bu kapanışı anlaması için 5 saniye beklenir.
c. 10 nolu posgate 4 saniyede bir SYS_GATE tablosunu check etmektedir. 30 nolu gate'in kapanacağını anladığı anda mesaj göndermeyi bırakır. Mesajları sadece 31 nolu gate'e göndermeye başlar
d. Kapanma aşamasında 30 nolu auth gate gelen mesajları işlemeye devam eder. Diğer gate'lerin algılamasına ve queue'da birikmiş mesaj sayısına göre bir süre işlem gelebilir. 10 nolu posgate, 31 nolu aıth gate'e yönlense bile 30'dan gelen cevap mesajlarını işlemeye de devam eder.
e. 30 nolu gate kapandıktan sonra program yada config değişikliği yapılır ve açılır.
f. Açıldığında ONL_STAT = 1 set edilerek diğer programların bunu algılaması beklenir.
g. 10 nolu posgate 4 saniye içinde 30 nolu gate'in açıldığını anlayıp mesaj göndermeye başlar.
h. Sonrasında 31 nolu auth gate aynı şekilde değiştirebilir.

Aynı anda hem 30 hem de 31 nolu auth gate'ler kapanmadığı sürece sistem cevap verebilir.

Ancak TCP client şeklinde çalışan programlarda aynı tipten 2 programın açılmasına izin verilmeyebilir. Genellikle bu tip programlar switch uygulamalarıdır ve karşıdaki host tek bir bağlantıya izin verir. İki program açılsa bile sadece bir tanesi bağlantı kurabilecektir. Diğer programların da bağlantısı olmayan programa mesaj göndermesi soruna yol açar. Bu tip uygulamalarda hangi uygulamanın açık hangisi kapalı olacağına manual karar verilir.

Log ve monitoring path tanımları

Path tablosunaLog ve monitoring uygulamarına doğru path girilmez. Çünkü zaten doğal olarak path tanımı olmalıdır. Path tablosuna gereksiz satır olacaktır. Bunun yerine SYS_GATE ve SYS_HOST tablolarında aktif ve yedek olarak 2'şer tane log ve mon host tanımı yapılır.

)E	LOG1	LOG2	MON1	MON2
	60	-1	62	-1
	60	-1	62	-1
	60	-1	62	-1

Örnekte görülen programlar 60 nolu log hostuna log göndermektedir. Yedek log hostu ise tanımlanmamıştır.

Mesaj Gönderilecek Program tipine karar verilmesi

Path tanımları, mesaj gönderilecek program tipi belli olunca kullanılmaktadır. Gönderilecek program tipi ise şu şekilde bulunur.

1. Bazı programlarda tip hardcoded olarak set edilir. Mesajın gönderileceği program tipi belli ve kesindir. Örneğin
    a. Hsm : hsm komutu çalıştırılması için her zaman hsm server program tipi
    b. H2h : Core banka çağrısı yapılacakse h2h program tipi
    c. Evt : timer event set edilecekse Evt program tipi
    d. Log : log gönderilekse her zaman log server program tipi
    e. …
2. Işlem bilgilerine göre yönlendirilecek program tipinin bulunması. Bir gate'e dışarıdan bir işlem geldiğinde, kartın ve işlemin tipine göre değişik uygulamalara bu mesaj yönlendirilir. Bazen bu yönlendirme de hardcoded yapılabilmektedir (örneğin ilk pakette bu şekilde set edilmekteydi). Ancak SYS_TXN_DST tablosu parametrik olarak hedef gate tipini belirtir. Aşağıdaki bu tablonun bir örneği görülüyor.

PRI...	BIN_SRC	BIN_DCI	BIN_BRD	BONUS_TXN	BONUS_BIN	TXN_TRM	BIN	NTW_FWD	NTW_DRC
1							462276	R	R
100	O	D						Z	Z
101	O	C						C	C
102	C							C	C
103	S							C	C
104	K							O	O
105	B			Y				W	W
106	B			N				B	B
107	D							B	B
108	J							B	B

Online uygulamalarda bu tablo priority kolonuna göre taranıp ilk match eden kayda göre işlem yönlendirilir. Boş olan kolonların bir etkisi yoktur, sadece dolu olanlar kontrol edilir. Ilk satırda sadece BIN = 462276 olması kontrol edilir. Doğruysa R tipindeki uygulamalaya işlem gönderilir. Bu işlem başka satırlara da uysa bile priority en düşük olan satır seçilir. BIN 462276 değilse, ikinci satıra geçilir. Burada kart onus-debit mi kontrolü vardır. Kritere uyarsa, Z tipinden bir uygulamaya yönlendirilir. Uymadığı durumda üçüncü satıra geçilir ve bu şekilde aranır.

Daha dar kriterler her zaman düşük priority ile verilmelidir. Yoksa hiç bir zaman ilgili yönlendirme çalışmaz.

Administrative mesajlar

Tüm programlarda işlem queue'sundan sonra Admin queue tanımlanmaktadır. Programlara bu queue vasıtasıyla özel mesajlar gönderilerek belli akışlara girmesi sağlanbilir. Uygulamalarda admin queue'dan mesajları alan bir thread vardır. Her program için ortak admin mesajlarının yanında, belli programlara özel mesaj tipleri tanımlanabilir (ve vardır).

Admin mesajları değişik şekillerde gönderileblir. Bazı backoffice ekranlarından belli mesajlar gönderilebilmektedir. Ayrıca basit bir utility uygulaması ve batch dosyaları ile console'dan mesaj göndermek mümkündür (Backoffice dışında gelişmiş bir admin tool olmasında fayda var).

ASysWrkAdm.cpp ve ASysWrkAdm.h dosyalarında admin mesajlarını işleyen class ve admin thread başlangıç fonksiyonu bulunur. Bu class'in tek bir global instance'i vardır ve program açılışında manager thread tarafından ayağa kaldırılıp, bitişinde sonlandırılır.

Admin queue'ya mesaj şu şekilde gönderilir. Mesajın ilk karakter mesaj tipini belirler. Programa aktif olarak şunlar gelmektedir:

- C : Komut mesajı
- I : Internal mesaj (Olbase library'den gelir)

Komut mesajları custom olarak her şekilde dizasyn edilebilir. Mevcut kullanılan komutlar şunlardır.

- C1 : signon gönder. Switch tarzı uygulamalar için anlamlıdır.
- C2 : signoff gönder. Switch tarzı uygulamalar için anlamlıdır.
- C3 : Parametre cache refresh et.
- C4 : Bin cache refresh et
- C5 : config bilgilerini tekrar oku (cfg dosyası yada tablodan)
- C6 : config bilgilerini monitoring'e gönder
- C7 : Transaction queue'sunu boşalt
- C8 : Admin queue'sunu boşalt
- C9 : Hsm queue'sunu boşalt

Bunların dışında sadece belli programlar için anlamlı kmutlar vardır

- CC : Visa advice start mesajı gönder
- CD : Visa advice start mesajı gönder
- CT : Mastercard Session Activate mesajı gönder.

…

Komut mesajlarının herhangi bir formatı yada uzunluk kısıtı yoktur ve programa göre custom istenen mesaj gönderilebilir. Komuttan sonra parametre geçilmesi gibi incelikler düşünülebilir. Örneğin signon – signoff gibi network komutlarından sonra session id bilgisinin geçilmesi mantıklı olabilir.

En çok kullanılan komut Parametre cache refresh komutudur.

Admin queue'su bazı switch uygulamalarında, library'den programa uyarı – bilgi mesajları geçmek için kullanılır. Switch bağlantısının gelip gitmesinde bu queue'ya bir mesaj atılır. Admin thread daha sonra bu mesajı loglara basar yada monitoring'e izleme amaçlı gönderebilir.

Monitoring data gönderimi

Online sistem merkezi bir monitoring uygulaması (OCMonCore) vardır ve tüm programlardan gelen bilgilerin monitoring tablolarına yazılmasından sorumludur. Monitoring database ocean'dan bağımsız olabilir ancak genellikle aynı database kullanılıyor. Programlar monitoring server'a msmq üzerinden mesaj gönderirler. Her program için bir tane aktif bir tane de yedek monitoring server tanımlanabilir.

> OC_SYS.SYS_GATE ve OC_SYS.SYS_HOST tablolarındaki MON1 ve MON2 kolonları aktif/pasif monitoring uygulama kodlarını gösteriri

Monitoring tarafına 2 farklı kategoride mesaj gönderilir.

1. Periyodik olarak gönderilen durum mesajları : Hangi periyodda mesaj gönderileceği parametriktir. SYS_PRM_ONLINE tablosundaki LIVE_INTERVAL değeri, milisaniye cinsinden monitoring data gönderim aralığını gösterir. GEnellikle 8000 (8 saniye) dir. Periyodik mesajları göndermek için bir thread çalışır. ASysWrkMon.h ve ASysWrkMon.cpp dosyalarında, bu thread fonksiyonu ve ilgili class bulunur. Monitoring thread class'I da tek bir global obje olarak tanımlanır. Periyodik gönderilen datalar şunlardır:
    a. Queue'larda bekleyen item sayıları
    b. Çalışan ve boşta thread sayısı.
    c. Thread çalışma süresi
    d. Son parametre update, config update zamanları
    e. Programın başlangıç zamanı
    f. Açık olan connection bilgileri
    g. …

Çalışan thread sayısı şu şekilde hesaplanır. Thread'ler bir işleme başladıklarında global obje üzerinden statülerini update ederler. Burada thread id bazında takip yapılır. Ikinci parametrede verilen başlangıç zamanı ise, threadin ne kadar zamandır meşgul olduğunu gösterir. Belli bir süreden fazladır bekleyen threadlerin izlenmesi faydalı olabilir.

```
aSysWrkMon.MonThrBusy(iThrId, sMon.iMscProcStart);
```

Thread işini bitirip boşa çıkacağı zaman şu fonksiyonu çağırır. Böylece thread istetistikleri update edilmiş olur. Thread hemen başka bir işe başlayacak olsa bile arada bu fonksiyonu çağırmalıdır.

```
aSysWrkMon.MonThrFree(iThrId, sMon.iMscProcEnd);
```

Connection bilgilerinin toplanması ise global oNwmMon objesi yardımıyla yapılır. Bu obje Olbase library ile de içiçe geçmiştir ve alt seviyedeki tcp server/client objelerine parametre olarak geçilip buralardan connection bilgileriyle update edilir. Gönderim aşamasında objedeki connection bilgileri array'a alınarak monitoring tarafına gönderilir. Aşağıdaki komut ile obje içindeki tüm connection bilgileri oMonConn isimli ön tanımlı struct içine yazılır.

```
oNwmMon.FillMsg(&oMonConn);
```

oNwmMon objesinin connection bilgilerini toplaması ise şu basit fonksiyonlar ile yapılır. Add fonksiyonu ile bir connection bilgisi tanımlanır. Client / Server / Hsm / db gibi tanımlardan biri kullanılır. Connection tipine göre hesaplamalar ve inputlar farklılık gösterir. Add fonksiyonundan dönen handle diğer fonksiyonlarda parametre olarak kullanılır.

```
pNwmHandle = oNwmMon.Add(NWM_TYPE_DB, aSysCfg.sDbmHost, 1521);
```

Alınan handle yardımıyla Connected, Disconnected gibi fonksiyonlar çağrılarak connection durumu güncellenir.

```
if (pNwmHandle != 0) oNwmMon.Connected(pNwmHandle  ....);

if (pNwmHandle != 0) oNwmMon.Disconnected(pNwmHandle);
```

oNwmMon objesi ayrıca içeride ne kadar bağlantıda kalındığı, kaç kez bağlanıldığı gibi istatistikleri de tutar.

2. Işlem yada olay bazlı gönderilen mesajlar.
    a. Gate üzerinden her işlem geçitiğinde mesaj gönderilmesi.
    Monitoring ekranlarından geçen işlemlerin, tps değerlerinin, ortalama işlem sürelerinin, onay/red oranlarının izlenmesini sağlar. Işlemde geçilen her gate monitoringe bir mesaj atar. Ancak performans açısından artık sadece en son atılan mesaj tabloya kaydedilmektedir. Daha önceleri her işlemde 3-4 mesaj

kaydedilmesi yer ve zaman sorunlarına yol açmaktaydı. Business threadde işlem bittiğinde aşağıdaki fonksiyon içinde mesaj gönderimi yapılır.

```
iRet = MonPushTxn();
```

b. Önemli eventlerin gönderilmesi. Monitoring ekranlarında akan uyarı yada hata kategorisindeki eventler programlardan gönderilebilir. Buraya çok nadiren mesaj gönderilmelidir. MOND makrosu ile gönderilir. Aşağıdaki ilk örnekte servisin stop/sart olması önemli olduğundan monitoring event gönderilmiştir. Switch tarzı programlarda ikinci örnekte olduğu gibi bağlantı gittiğinde monitoring event gönderilir.

```
sprintf(sBff1, "Service %s started. " __DATE__ " " __TIME__ " .",
aSysCfg.sSvcName);
MOND(ML_TYP_INF, ML_CAT_SYS, ML_SEV_INFORMATION, sBff1);

sprintf(sBff1, "Connection Closed %s", sData);
MOND(ML_TYP_WRN, ML_CAT_NTW, ML_SEV_NORMAL, sBff1);
```

c. Expire olan eventlerin bilgilerinin gönderilmesi. OCEvent uygulaması diğer uygulamalar için timer gibi çalışmaktadır (ayrıca anlatılacak). Herhangi bir timer süresi dolup expire olduğunda monitoringe bilgi gönderilir.

d. Hsm Mesajlarının gönderilmesi : Bu loglama çok fazla resource tükettiği ve gereksiz olduğu için kapatıldır.

DB classlarının üretilmesi ve kullanılması

Proje yapısını incelediğimizde “Table” filtresi altında tablo ve sp’lere erişen classlar görülecektir. Erişilen her tablo ve sp için ayrı bir class eklenir. Nadiren bir class içinde birden fazla tablo yada sp için kod olabiliyor. Table folder’i dışında herhangi bir db query’si yada db objesi görülmez.

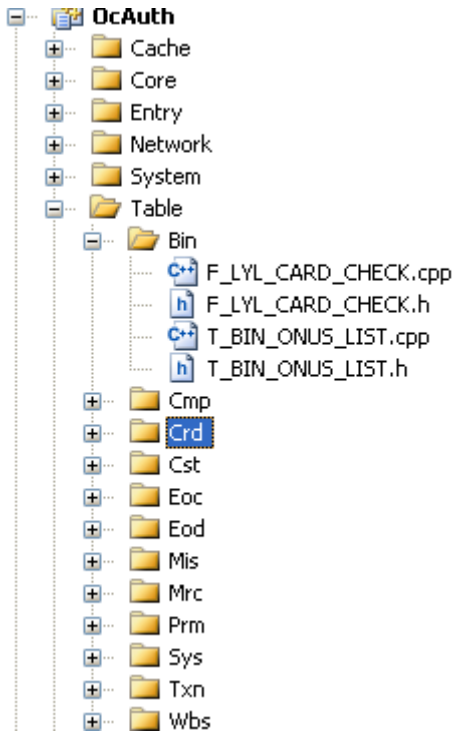


Table filtresi altında ayrıca schema isimlerine göre bir filtreleme daha yapılmıştır. Çok fazla tablo olduğu için aramada kolaylık sağlar. Yukarıdaki örnekte Bin klasöründe 2 tane class görünüyor.

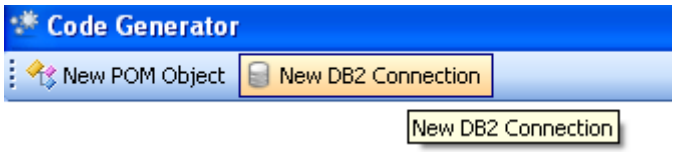
- T_BIN_ONUS_LIST -> OC_BIN. BIN_ONUS_LIST tablosuna erişen class.
- F_LYL_CARD_CHECK -> OC_BIN.SP_LYL_CARD_CHECK sp’sini execute eden class.

Tablo class isimlerinin başına T_ öneki getirilerek, Function, sp objeleri de isimlerinin başlarına F_ öneki getirilerek oluşturulmakta.

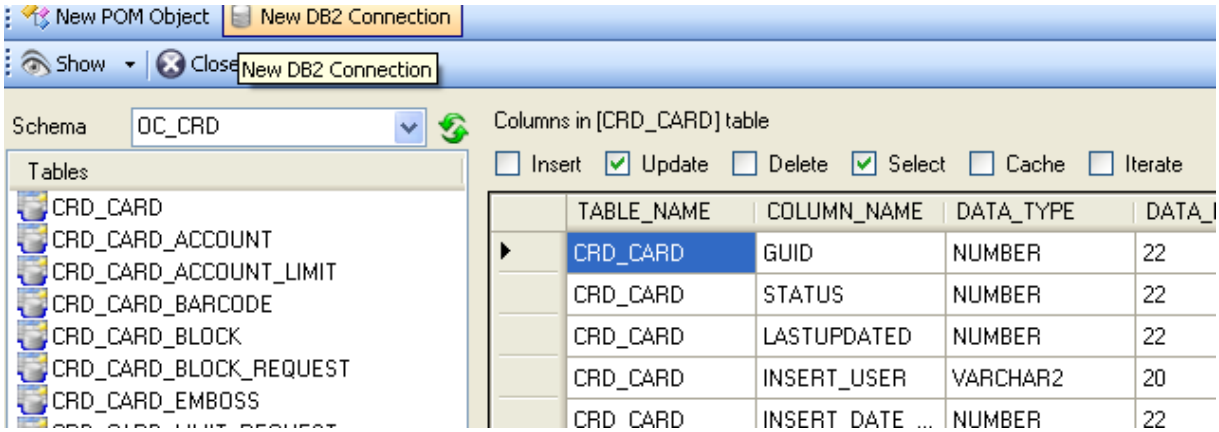
Tablo objelerini ilk olarak ORMapper isimli uygulama ile otomatik olarak üretiyoruz ve gerektiğinde üzerinde oynamalar yapıyoruz. Pek çok tablo için bu objelerde değişiklik yapılmaktadır. Değişiklik yapıldıktan sonra tekrar ORMapper ile kod üretilmesi durumunda yapılan değişiklikler ezilebileceği için yeni eklenen kolonları elle objenin içine ekliyoruz. Çok fazla kolon değişiğikliği olması halinde, ORMapper’dan üretilen bulk kodlar sadece gerekli yerlere paste edilerek kodlar merge edilebilir.

ORMapper programıyla C++ DB objesi üretilmesi:

a. ORMapper.exe çalıştıılıp, New DB2 Connection seçilir.

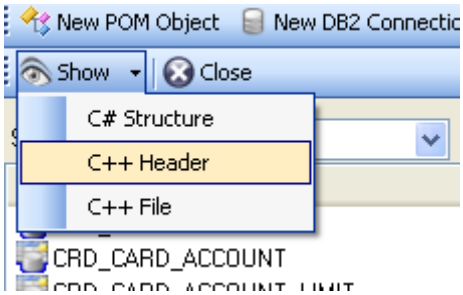


b. Schema ve tablo getirilerek alanlar listelenir.



c. Burada üretilmesi istenen query'ler seçilir (Select, Inser, Update, Delete, Cache ..). Genellikle 1-2 tanesini seçilmesiyeterlidir.

d. Ilgili .cpp ve .h dosyaları boş olarak projeye eklenir ve üretilen kodlar bu dosyalara kopyalanır. Ormapper'dan Save as ile kaydetmeyelim çünkü bu durumda unicode olarak kaydediyor ve projede bazı uyumsuzluklar oluyor.



e. Bundan sonra projede değişiklikler gerekir. Db class headerlarını genellikle proje ile aynı addaki header dosyasına ekliyoruz (Örneğin OCGatePos.h):

```
#include "T_CRD_CARD.h"
```

f. İlgili DB classinin bir objesi eklenir. Tablo cachelenecekse, ACchPrm.h içine eklenir. Başka bir objeden query çalıştırılacaksa o obje içine eklenir. Genel olarak BMsgPrc içine eklenmekte

```
T_CRD_CARD CRD_CARD;
```

g. Cache objesi içine eklendiyse, nasıl kullanılacağı cache bölümünde anlatılmıştır. Başka bir obje içinde DB table objesi eklendiyse, Config ve Clear metodları Initialize ve Destroy akışlarında çağrılmalıdır.

```
iRet = CRD_CARD.Config(&oCnnTxn);
if (iRet != ORC_OKI)
	__ERRD("D968222", "CRD_CARD.Config error", ORC_ERR);

...

CRD_CARD.Clear();
```

Config fonksiyonu çağrılmadan objeye erişilmeye çalışıldığında program göçebilir. Clear fonksiyonu çağrılmazsa resource leak oluşabilir. Ancak query ikinci kez config yapıldığında Clear edilmese bile leak oluşmaz.

Otomatik olarak sp üreten bir program maalesef yok. sp eklemek için mevcut bir sp class'ının kodlarını alıp rename ederek içeriğini de editliyoruz.

Üretilen Tablo objesinin kodlarını inceleyecek olursak, Header dosyasında ilk görülen bir structure'dur. Bu structure, tablodaki bir satırı turabilecek şekilde üretilmiştir. Table objesinin fonksiyonlarına bu structure tipinde data geçilir yada alınır.

```
typedef struct R_BIN_ONUS_LIST
{
	OSI8 guid;		// NUMBER 22 18 0
	OSI2 status;		// NUMBER 22 1 0
	OSI8 lastupdated;		// NUMBER 22 18 0
	OSC bin[6+1];		// VARCHAR2

	...


}R_BIN_ONUS_LIST;
```

ODbmCrs db library'sine ait bir class'tır. Her bir database query'si için bir adet bu objeden üretilir. ORMapper'ın ürettiklerinde başka, yeni query'ler için bu objeden eklemek gerekir. Örneğin aşağıdaki oCrsCch objesi bir select query'si çalıştırıp sonuçları dönecektir.

```
ODbmCrs  oCrsCch;
```

Tablo objelerinin Config fonksiyonu çağrılarak cursor objeleri initialize edilir. Burada parametre olarak connection objesinin geçildiğine dikkat edilmelidir. Connection stop/start

edildiğinde, Config fonksiyonu tekrar çağrılır. Uygulama kodlarında zaten bu şekilde bir yapı vardır.

```
OSI4 Config(ODbmCnn *fiCnn);
```

Config fonksiyonunda aşağıdaki gibi tüm cursor objelerinin Initialize edildiği görülecektir. ORMapper burada basit bir query oluşturduğu için genellikle query'leri sonradan edit etmek gerekebilmektedir. Özellikle where kısımları değişir.

```
oCrsCch.CmdInit(fiCnn, OSA_SQL,
      "SELECT "
      "GUID, STATUS, LASTUPDATED, BIN, DCI, "
      "BRAND, BIN_CLASS, BANK_NAME, CVKA_VALUE, CVKA_KCV, "
      "CVKB_VALUE, CVKB_KCV, CVK2A_VALUE, CVK2A_KCV,"
      "CVK2B_KCV, PVKI, PVKA_VALUE, PVKA_KCV, PVKB_VALUE,"
      "PVKB_KCV, PVK2A_VALUE, PVK2A_KCV, PVK2B_VALUE,"
      "CAVVKA_VALUE, CAVVKA_KCV, CAVVKB_VALUE, CAVVKB_KCV "
      "FROM OC_BIN.BIN_ONUS_LIST "
      "WHERE STATUS=1 "
      );
```

Yukarıdaki query tabloyu cache'leyen bir query olduğu için herhangi bir where kriteri yok. Örnek olması açısından where kriteri olan başka bir cursor objesine bakalım. Burada parametrik bir query örneği görülüyor. Server tarafında bu query sadece bir kez parse edilir. Sonraki kullanımlarda sadece card_no parametresi geçilerek hızlı çalışması sağlanır.

```
// select statement
oCrsSel.CmdInit(fiCnn, OSA_SQL,
      "SELECT "
      "GUID, STATUS, LASTUPDATED, CARD_LEVEL, CARD_NO, "
      "MAIN_CARD_NO, CUSTOMER_NO, MAIN_CUSTOMER_NO,
      "PHOTO_REQUEST_FLAG, AGREEMENT_FLAG "
      "FROM OC_CRD.CRD_CARD "
      "WHERE "
      " CARD_NO = :card_no "
      "AND STATUS=1 ");

oCrsSel.PrmNew("card_no", OVT_TXT, OPD_INP);
```

Query içindeki parametrelerin başında ":" karakteri yer alır. CmdInit fonksiyonundan sonra PrmNew ile query'de geçen tüm parametreler ayrıca tanımlanmalıdır. Yukarıdaki örnekte card_no parametresi eklenmektedir. OVT_TXT parametrenin karakter array tipinde olduğunu göstermekte. OVT_NUM olursa number tipinde bir parametre olacaktır. OPD_INP bu parametrenin input parametresi oldığunu gösteriri. Output parametreleri sadece sp'ler için vardır.

NOT : Ocean'dan tüm kolonlar NUMBER yada VARCHAR2 tipinde tutulmaktadır.

Select fonksiyonunu inceleyecek olursak, parametre olarak R_CRD_CARD tipinden bir struct aldığını görürüz.

```
OSI4 T_CRD_CARD::Select(R_CRD_CARD *fiStr)
{
```

Çağıran yerde where kriterinde kullanılacak olan member'lar doldurulmuş olmalıdır. Bu örnekte card_no member'inda gelen değerle query parametresi set ediliyor.

```
oCrsSel.PrmGet("card_no") = fiStr->card_no;
```

Ardından query execute edilmekte. CmdExec fonksiyonu hiç bir data getirmez. Insert ve update tipinden query'lerde CmdExec başarılı oluysa başka bir işlem yapılmadan çıkılır. Select tipinden query'ler ise CmdEXec'den sonra select edilen satırların alınması işlemi vardır. Sp tipinden query'lerde ise sp'nin varsa output parametreleri alınabilir.

```
iRet = oCrsSel.CmdExec();
if (iRet != ORC_OKI)
      __ERRD("D968301", pDbmCnn->CnnError(), ORC_ERR);
```

Select query'leri için ResFetch ile ilk kayıt getirilmiş olur. ResFetch fonksiyonunun her çağrılışında sonraki kayıt alınır. Tek kayıt beklendiği durumda ResFetch tek sefer çağrılır, data gelmişse alınır ve çıkılır. Ancak bir selectte Resfetch fonksiyonunu birden fazla çağrıldığı pek çok örnek bulunabilir. ResFetch fonksiyonundan ORC_NON değeri döülmüşse, herhangi bir row gelmemiştir.

```
iRet = oCrsSel.ResFetch();
if (iRet == ORC_ERR)
      __ERRD("D968302", pDbmCnn->CnnError(), ORC_ERR);

if (iRet == ORC_NON)
      return ORC_NON;
```

Resfetch ile Row gelirse, internal db objelerinde gelen row'a ait değerler bulunur. Bu değerler alınarak structure içindeki member'lar set edilir. Aşağıdaki örnekte 3 kolon için select edilmiş değerlerin atılması görülmektedir.

```
fiStr->guid = oCrsSel.ColGet("GUID").AsOSI8();
fiStr->status = oCrsSel.ColGet("STATUS").AsOSI2();
fiStr->lastupdated = oCrsSel.ColGet("LASTUPDATED").AsOSI8();
```

Cache'lenen tabloların class'larında sadece select fonksiyonu çalışmakta ve dönen değerler hafızaya alınmaktadır. T_BIN_ONUS_LIST örneğinden cache fonksiyonlarını inceleyelim.

CacheLoad fonksiyonu, tüm tabloyu select eder. Aiağıdaki görüldüğü gibi while döngüsünün başında selectten dönen bir sonraki satırı getirmekte ve tüm satırlar için döngüye girmektedir.

```
OSI4 T_BIN_ONUS_LIST::CacheLoad()
{
    ...

    iRet = oCrsCch.CmdExec();
    if (iRet != ORC_OKI)
          __ERRD("D968282", pDbmCnn->CnnError(), ORC_ERR);

    while ( ((iRet = oCrsCch.ResFetch()) == ORC_OKI) && (bIsUp == B1) )
```

satırda gelen değerler bir struct içine doldurulur.

```
    r_BIN_ONUS_LIST.guid = oCrsCch.ColGet("GUID").AsOSI8();
    r_BIN_ONUS_LIST.status = oCrsCch.ColGet("STATUS").AsOSI2();
    r_BIN_ONUS_LIST.lastupdated = oCrsCch.ColGet("LASTUPDATED").AsOSI8();
```

Doldurulan bu structure, std::map tipinden bir yapıya atılır. Bu örnekte bin kolonuna göre map'den alınacağı için map kullanımı uygun olmuş. Ancak her zaman map kullanılmaz. Tablonun yapısına göre gelen satırların nasıl saklanacağı programcıya kalmıştır. Yeri geldiğinde vector yapısı yada array kullanılabilir. Hatta bazen tek satırlık bir tablo cach'leniyorsa, cache aynı zamanda bir structure olabilir.

```
                sprintf(sCchKey, "%6s",
                      r_BIN_ONUS_LIST.bin);
                m_BIN_ONUS_LIST[sCchKey] = r_BIN_ONUS_LIST;
```

CacheFind fonksiyonu, cache'e atılmış datalardan bir satırı getirmek için kullanılır. Aşağıdaki örnekte gelen struct içindeki bin member'ı set edilmiştir ve bu member'a göre map'de arama yapılır. Kayıt bulunursa struct'a kopyalanır. Yoksa ORC_NON dönülür.

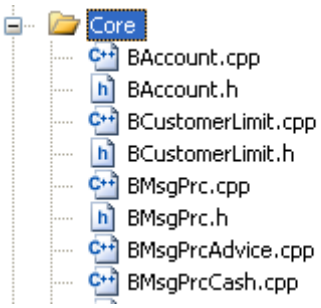
```
OSI4 T_BIN_ONUS_LIST::CacheFind(R_BIN_ONUS_LIST *fioStr)
{
    OSC sCchKey[128] = {0};
    M_BIN_ONUS_LIST::iterator iFindCol;

    try
    {
        sprintf(sCchKey, "%6s",  fioStr->bin);

        iFindCol = m_BIN_ONUS_LIST.find(sCchKey);
        if (iFindCol == m_BIN_ONUS_LIST.end())
            return ORC_NON;

        *fioStr = iFindCol->second;
        return ORC_OKI;
    }
```

Genel Business thread yapısı

Utility thread'ler (admin, cache update, monitoring …), yaptıkları iş bakımından tüm programlarda hemen hemen ortaktır. Bazı farklar olabilir. Esas fark business thread'lerde ortaya çıkmaktadır. Business thread'leri aSysWrkTxn isimli global obje yönetir. Bu obje istenen sayıda business thread'I ayağa kaldırır ve program bitişinde sonlandırır.

Her bir business thread, BMsgPrc ismini verdiğimiz global business class'in bir instance'ına sahiptir. Thread fonksiyonunda bu class'in az sayıdaki public metodu çağrılarak kontrol, class içindeki kodlara devredilir. Tüm business kodları ve dataları bu class içindedir.

BMsgPrc class'I çok büyük olduğundan implementasyonu pek çok dosyaya dağıtılmıştır. Bu dağıtım fonksiyonların işlevine göre yapılır. Projede Core dizini altında business classlar bulunur.



Business thread'den sadece business class'in aşağıdaki public fonksiyonarı çağrılır.

```
** business thread start fonksiyonu1
static void* __WorkerProc(void *fiPrm)
{
	BMsgPrc bMsgPrc; ** thread'e ait business objesi

	iRet = bMsgPrc.Config(); ** business objesi initialize edilir.

	** program sonuna kadar thread çalışır
	while(pCaller->bIsUp == B1)
	{
		** business threadden mesaj pop etmesi istenir
		iRet = bMsgPrc.MsgPopTxn();

		** mesaj pop edildiyse işlenir
		if (iRet == ORC_OKI)
			iRet = bMsgPrc.MsgCore();
```

Program ayakta olduğu sürece Pop-Process akışı devam eder. Bu akış mantığı tüm programlarda ortaktır.

Yukarıdaki 3-4 public fonksiyon dışında business objelerinin içlerinde de ortak kısımlar vardır. Aşağıdaki mesaj işleme mantığı tüm programlarda görülebilir. Burada anlaşılması açısından sadece başlıklar verilmiştir.

```
OSI4 BMsgPrc::MsgCore()
{

	MonStartTxn();

	iRet = MsgPre();

	iRet = MsgBsn();

	iRet = MsgPost();

	iRet = MsgSend();

	MonEndTxn();
}
```

MsgPre, mesajla ilgili ön işlemleri yapar. Dışarıdan gelen mesaj ise parse edilmesi, field kontrolleri, data initialize işlemleri, database bağlantısının kontrolü, Mesaj ID (GUID) üretilmesi gibi işleti yapar.

MsgBsn, bundan sonraki business mantığını içerir.

MsgPost, işlem bittikten sonraki akışı içerir. Bu aşamada işlem onaylanmış yada reddedilmiştir zaten. Dışarıya mesaj gönderilecekse bu mesajın hazırlanması, gidecek fieldların kontrolü gibi işler yapılır.

MsgSend ise diğer gate'lere yada external bağlantılara mesajı gönderir. Gönderilen mesaj requet yada reply mesajı olabilir. Request mesajı gönderiliyorsa, daha sonra reply mesajı da bu akışa girecektir büyük ihtimalle. Reply mesajını başka bir thread de işleyebilir.

Batch uygulamalardan Olbase kullanımı

Olbase genel olarak online uygulamalar düşünülerek yazıldığı halde bazı C++ ile yazılmış batch uygulamalarda da kullanılabilmektedir. Kodlama olarak online uygulamalara benzemekle birlikte şu farklar ortaya çıkmıştır

- Servis olarak çalışmazlar. Başka bir uygulama tarafından ayağa kaldırılırlar ve çağıran uygulamanın user bilgileri ile çalışırlar
- Cache refresh alkışı ve threadi yoktur. Tek cache tutulup açılışta doldurulur.
- Admin thread yoktur
- Diğer programlar ayaqktamı kontrolü ve threadi yoktur.
- Console input bekleme yoktur. Business threadler ayağa kaldırılıp işlerinin bitirmeleri beklenir.
- Business thread'ler genellikle batch dosya okum / yazma işlemlerini yaparlar

Batch uygulamaları çoğunlukla OCTask isimli uygulama ayağa kaldırır. Batch uygulamaların bazı dizinlere yada kaynaklara erişimi gerekiyorsa, OCTask uygulamasının çalıştırıldığı user account'ın değiştirilmesi gerekebilir.

64 bit konuları

Artık 64 bit makinaların oranları arttığından bu konuda da bazı hususlara dikkat etmek gerekiyor.

1. Programları 32 bit yada 64 bit olarak derleyebilirsiniz. 64 bit sistemde her ikisi de çalışır. ancak 64 bit derlenmiş uygulamalar 32 bit windowsta çalışmaz.
2. Uygulamalarımız 32 bit ise oracle client 32 bit kurulmalıdır. Uygulama 64 bit ise oracle client 64 bit olmalıdır.
3. 64 bit makinada 32 bit uygulama kullanırken registry patherinde değişiklik olur. Örneğin programınız

   "HKEY_LOCAL_MACHINE\SOFTWARE\OCEAN\Online"

   şeklinde erişmeye çalıştığı halde şu key'e erişmektedir.
   "HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\OCEAN\Online"

4. Programı 64 bit derliyorsanız, OCI library 64 bit client ile gelen OCI library olmalıdır. Makinanıza 64 bit client kurmadan bu işi şöyle halledebilirsiniz. 64 bit client kurulmuş bir makinadan OCI dizinini alıp kendi makinanızda bir yere kopyalayın. Sonra project directories'den 64 bit config için bu yeni dizini set edip kullanın.
5. Çok kullanılmamakla birlikte IBM mq kullanılıyorsa, 32 – 64 bit library'ler farklıdır. Sadece belli projeler için gerekli bir durum.
6. Sql server kullanılıyorsa, Native Client 64 bit kurulmalıdır. Uygulamalar 32 bit olsa bile.
7. …

ONLINE System Tabloları ve Config Dosyası

Bu bölümde online programların yönetimini sağlayan tablolar ve alanlarına topluca bakılacaktır. Bazı field'lar önceki bölümlerde yeri geldikçe ayrıntılı anlatılmış olabilir. Burada daha kısa olarak listelenecektir.

OC_SYS.SYS_PRM_ONLINE

Son yapılan projelerde, Cfg dosyası yerine SYS_PRM_ONLINE tablosu kullanılmaya başlanmıştır. Bölüm sonunda, tablodaki hangi alanın cfg dosyasındaki hangi key'e denk geldiği de listelenmiştir.

SYS_PRM_ONLINE tablosunda programların genel configurasyon bilgileri tutulur. HEr program için bu tabloda bir satır vardır. Program kaydı açılışta okunur, istenen bir anda tekrar okunması sağlanabilir. Ancak tekrar okumalarda çoğu parametre yenilenmez. Bu parametrelerin değişebilmesi için program restart edilmelidir. Değişmeyen parametrelerin yanına static ibaresi konmuştur.

- GATE_IP (static) : Programın çalıştığı (çalışacağı) Ip adresi. Programlar tabloda kendi satırlarını IP adresi ve Program tipi alanlarını key vererek bulurlar. Yani program kodu açılışta bilinmez. Bu durumda aynı IP adresinde aynı program tipinden 1 tane açılabilir. Bazı bankalarda programlar bu şekilde çalışıyor. Bu tip kullanımın getirdiği sorunlardan dolayı, opsiyonel olarak cfg dosyasından program kodunun alınması ve bu tabloya program kodu ile gelinmesi tercih edilebilir.
- GATE_NTW (static) : Program tipi. Gate ip ile birlikte programın kendi satırını bulmasını sağlar. Program içinde bir anlamı yoktur. Gate ve host tipleri eskiden aynı olabiliyordu. Program kodunun cfg dosyasından alınmadığı durumlarda gate ve host tiplerinin aynı olamaması gibi bir durum doğdu. Monitoring uygulamasının tipinin M'den N'ye çekilmesi de bundandır.
- GATE_CODE (static) : Program kodu. Program kodu cfg dosyasında alınırsa, bu kod ile satır select edilir. Ip ve Program tipinden select edilirse, program kodu buradan set edilir.
- RUN_AS_SVC (static) : 1-Servis olarak çalış. 0-Console ekranında çalış. Bu parametre 1 iken exe tıklanarak çalıştırılaamaz. Parametre 0 iken servis listesinden start edilemez.
- LIVE_INTERVAL : monitoring'e hangi aralıkta program statu bilgilerinin gönderileceği. Milisaniye cinsinden.
- WORKER_CNT (static) : Business thread sayısı.
- RVRS_WORKER_CNT (static) : Reversal thread sayısı. Çoğu program için bir anlamı yoktur. Switch tarzı programlarda arka planda reversal üretilmesi içindir. 1 verilir.
- CNN_TRY_CNT : kullanılmıyor. Işlemde db bağlantısının kaç kez deneneceğini gösteriyordu. Ancak bir kez bağlanamayınca tekrar denemek işlem için mantıksız.

- SVC_DEPENDENCIES (static) : servis olarak çalışıyorsa, bağlı olduğu diğer servisler. Genellikle sadece “MSMQ” verilir. Bu servis başlamadan önce bağlı olduğu servisler de otomatik çalışır.
- PURGE_QUEUE (static) : açılışta işlem queue temizlenir.
- STARTUP_LOG_DIR (static) : açılış ve kapanış loglarının atıldığı dizin. Servis manager ve program manager objelerinden log atılır.
- LOCAL_LOG_DIR : local dizine log atılacaksa bu dizinin path’i
- MASK_TRACK2 : Loglarda ve tablolarda track2, pin block, cvv2 gibi hassas bilgiler maskelenerek yazılır.
- FUP_INTERVAL : (Sadece bir bankadaki OCBatch uygulamasında var) file update mesajlarının tekrarlanma sıklığı. Kullanılmıyor.
- FUP_TRY_CNT : (Sadece bir bankadaki OCBatch uygulamasında var) file update mesajlarının tekrarlanma sayısı. Kullanılmıyor.
- FLUSH_PATH : (Sadece LOG Server için) Merkezi Log dizini.
- SAVE_PATH : (Sadece LOG Server için) Merkezi logların sıkıştırılıp saklandığı arşiv dizini. Rar.exe kullanılarak shell komutu ile sıkıştırılır.
- ARC_DAYS : (Sadece LOG Server için) Logların arşivlenme gün sayısı.
- LOG_SVR_INFO : “Information“ tipindeki logları yaz. Programlarda dikkat edilmediği için ve bugüne kadar hep açık olduğundan 1 olarak set edilmeli.
- LOG_SVR_ERROR : “Error” tipindeki logları yaz. Programlarda dikkat edilmediği için ve bugüne kadar hep açık olduğundan 1 olarak set edilmeli.
- LOG_SVR_WARNING : “Warning” tipindeki logları yaz. Programlarda dikkat edilmediği için ve bugüne kadar hep açık olduğundan 1 olarak set edilmeli.
- LOG_DST_SCREEN : Ekrana log at
- LOG_DST_FILESYS : Local dosyalara log at. Msmq log açık olduğu halde log servera ulaşılamazsa her zaman local dosyaya log atar
- LOG_DST_MSGBOX : Msmq üzerinden merkezi log at.
- LOG_CAT_NETWORK : Network kategorisindeki logları yaz.
- LOG_CAT_BOX : (Anlamsız) Box kategorisindeki logları yaz.
- LOG_CAT_ISOMSG : ISO field dump loglarını yaz.
- LOG_CAT_MESSAGE : Full mesajı loga yaz.
- LOG_CAT_MONITORING : Monitoring eventlerini gönder.
- LOG_CAT_FUNC : (Kullanılmıyor) Fonksiyon başlangıç ve bitişinde log at.
- LOG_CAT_FUNC_PRM : (Kullanılmıyor) Fonksiyon parametrelerini logla.
- LOG_CAT_BUSINESS : Business kategorisindeki logları yaz.
- LOG_CAT_DATABASE : Database kategorisindeki logları yaz.
- LOG_CAT_COLLECTION : (Kullanılmıyor) Collection kategorisindeki logları yaz.
- LOG_CAT_FORMATTER : Formatlamayla ilgili logları yaz.
- SVC_DESCRIPTION : Servis açıklaması. Servis listesine çıkan açıklama.

Cfg dosyası kullanan bankalar yada programlar için bu alanların cfg dosyasındaki karşılıkları şu şekildedir.

DB KOLON ADI	CFG DOSYASI SECTION + KEY
registry'den alınır	SCT_DBM.DBM_HOST
registry'den alınır	SCT_DBM.DBM_USER
registry'den alınır	SCT_DBM.DBM_PSWD
GATE_IP	---
GATE_NTW	SYSTEM.SYS_GATE_NTW
GATE_CODE	SYSTEM.SYS_GATE_CODE
RUN_AS_SVC	SYSTEM.SYS_MODE
LIVE_INTERVAL	SYSTEM.MON_LIVE_INT
WORKER_CNT	SYSTEM.SYS_WORKERS
RVRS_WORKER_CNT	SYSTEM.SYS_WORKERS_REV
CNN_TRY_CNT	SCT_DBM. CNN_TRY_CNT
SVC_DEPENDENCIES	SYSTEM.SVC_DEPENDENCIES
PURGE_QUEUE	SYSTEM.SYS_PURGE_QUEUE
STARTUP_LOG_DIR	SYSTEM.STARTUP_LOG_DIR
LOCAL_LOG_DIR	SCT_LOG.LOCAL_LOG_DIR
MASK_TRACK2	SYSTEM.SYS_MASK_TRACK2
FUP_INTERVAL	FILE_UPDATE.TRY_INTERVAL
FUP_TRY_CNT	FILE_UPDATE.TRY_COUNT
FLUSH_PATH	SYSTEM.FLUSH_PATH
SAVE_PATH	SYSTEM.SAVE_PATH
ARC_DAYS	SYSTEM.ARC_DAYS
LOG_SVR_INFO	SCT_LSI.KEY_LSI_INF
LOG_SVR_ERROR	SCT_LSI.KEY_LSI_ERR
LOG_SVR_WARNING	SCT_LSI.KEY_LSI_WRN
LOG_DST_SCREEN	SCT_LDS.KEY_LDS_SCR
LOG_DST_FILESYS	SCT_LDS.KEY_LDS_FSY
LOG_DST_MSGBOX	SCT_LDS.KEY_LDS_BOX
LOG_CAT_NETWORK	SCT_LOG.KEY_LOG_NTW
LOG_CAT_BOX	SCT_LOG.KEY_LOG_BOX
LOG_CAT_ISOMSG	SCT_LOG.KEY_LOG_ISO
LOG_CAT_MESSAGE	SCT_LOG.KEY_LOG_MSG
LOG_CAT_MONITORING	SCT_LOG.KEY_LOG_MON
LOG_CAT_FUNC	SCT_LOG.KEY_LOG_FNC
LOG_CAT_FUNC_PRM	SCT_LOG.KEY_LOG_FPR
LOG_CAT_BUSINESS	SCT_LOG.KEY_LOG_BSN
LOG_CAT_DATABASE	SCT_LOG.KEY_LOG_DBM
LOG_CAT_COLLECTION	SCT_LOG.KEY_LOG_CLL
LOG_CAT_FORMATTER	SCT_LOG.KEY_LOG_FMT
SVC_DESCRIPTION	---

OC_SYS.SYS_HOST

Ocean sisteminde online programlar gate ve host olmak üzere 2 gruba ayrılmıştır. Gate, bir işlemin asenkron olarak geçtiği programları ifade etmekteydi. Host ise işlem akışında genellikle senkron olarak alınan servisleri veren programları. Host tarzı programlara tüm işlem alanları geçilmez. Bu programlar asıl olarak işlem akış path'inde yer almazlar.

Ancak zamanla H2h gibi programların gate'mi yoksa host mu olduğu tam belli olmadığından bu ayrım önemini yitirdi. Zaten gate ve hostların ayrı tablolarda olmasının pratikte pek bir önemi yok. Heps tek tablo da olabilirdi.

- MBR_ID : Kullanılmamakla birlikte farklı member'lara ait programların açılabileceği düşünülmüştür. Ancak multi member yapıda programlar değil datalar ayrışabilir. Bu kolon aktif olarak kullanılmıyor.
- HOST_CODE : Programın unique numarası. Her program bu değerden kendi satırına ve ilişkide olduğu programların satırlarına ulaşır.
- HOST_TYPE : Program tipi.
- HOST_NAME : Program açıklaması. Online için anlamı yok.
- HOST_HOST : Program IP adresi. Programlar birbirlerine bu Ip yardımıyla mesaj gönderir. Msmq yada tcp mesaj gönderilebilir. Açılışta makina IP'si ile bu IP kontrol edilir. Farklıysa program açılmas
- HOST_TXN_BOX : Transaction queue. Programın worker thread'leri bu queue'dan mesajları alıp işlerler. Bir program diğerine msmq ile mesaj gönderiyorsa, ilgili programın txn queue'suna mesaj gönderir. Host programlarında alternatif olarak tcp mesaj da gönderilebilir.
- HOST_ADM_BOX : Admin queue. Programa komutlar bu queue üzerinden gönderilir. Admin thread komutları işler.
- HOST_UP : Program kapalı olarak duracaksa N yapılır. Böylece Monitoring ekranlarında uyarı verilmemiş olur.
- HOST_PORT : Programa MSMQ yerine TCP ile mesaj gönderilmesi için kullanlabilir. Her program tcp mesaj kabul etmez. Örneğin Hsm server kabul eder, Ama log server etmez. Tcp dinlemiyorsa bu alan boştur.
- HOST_TOV_MSEC : timeout süresi. Bu tablo için anlamsız.
- HOST_CCH_UPD_TYPE : cache update tipi (I-Interval, E-Exact)
- HOST_CCH_UPD_TIME : cache update parametresi. Interval ise kaç dakikada bir cache refresh edileceği. Exact ise, HHMM cinsinden saat verilir.
- HOST_TXN_BOX_TTL : işlem queue bekleme süresi. Bu kadar milisaniye süreyle queue'da bekleyen mesajlar işlenmeden çöpe atılır (değeri 0'dan büyük ise).
- HOST_ADM_BOX_TTL : Admin queue bekleme süresi. Bu kadar milisaniye süreyle queue'da bekleyen mesajlar işlenmeden çöpe atılır (değeri 0'dan büyük ise).
- MON_USER : anlamsız???

- MON_PWD : anlamsız ???
- LAST_ONL_TIME : Programlar bu alanı 4 saniyede bir update eder. Bir programın göçtüğünü bu alanın değişmemesinden anlayabiliyoruz.
- OUTER_UNIT : Ocean sisteminde olmayan programlar da bu tabloya girilebilir. Bu durumda bu alan Y yapılır. Fraud, safenet hsm server gibi programlar bu şekilde girilir. Bu programlara sadece TCP bağlantı kurulabilir ve port no dolu olmalıdır.
- HOST_HOST2 : Dış programlar için yedek host tanımı yapılabilir. Örneğin fraud için yedek host IP buraya girilir.
- HOST_PORT2 : Dış programlar için yedek host tanımı yapılabilir. Örneğin fraud için yedek host port numarası buraya girilir.
- INST_IDX : kullanılmıyor. Amacını hatırlamıyorum
- ONL_STAT : program kapanırken 0 yapılır. Böylece diğer programlar kapandığını anlayıp mesaj göndermeyi bırakır. Göçmelerden farklı olarak time update olmasına bakılmaz. Program açıksa bu alanı 1 olarak update eder.
- TXN_QUE_COUNT : işlem queue'da bekleyen item sayısı. Bu sayı belli bir seviyenin üstüne çıkarsa diğer programlar göndermeyi kesebilir. Bu alan aktif olarak monitoring ve Log server programlarında kullanılıyor. Tüm programlara yaygınlaştırılsa iyi olabilirdi.
- QUE_LOCK : Program açık ama işlem queue'da çok fazla mesaj birikmişse 1 olarak set edilir. Log ve monitoring server uygulamaları set ediyor sadece. Bu flag 1 olunca diğer programlar göndermeyi keser. Queue sayısı belli bir seviyenin altına inince tekrar 0 yapılır.
- LAST_WRK_TIME : önemsiz. Kullanmıyor.
- LOG1 : primary log server uygulamasının kodu.
- LOG2 : secondary log server uygulamasının kodu (opsiyonel).
- MON1 : primary monitoring server uygulamasının kodu.
- MON2 : secondary monitoring server uygulamasının kodu (opsiyonel).

OC_SYS.SYS_GATE

Ocean online gate programlarının tanımları buraya girilir. Gate'ler hem kendilerinin hem de konuşacakları diğer gate'lerin bilgilerini buradan alır. Ayrıca gate'lerin ayakta olduğu kontrolü buradan yapılır. (SYS_HOST tablosunun açıklamasına da bakılmalı)

- MBR_ID : Kullanılmamakla birlikte farklı member'lara ait programların açılabileceği düşünülmüştür. Ancak multi member yapıda programlar değil datalar ayrışabilir. Bu kolon aktif olarak kullanılmıyor.
- GATE_CODE : Programın unique numarası. Her program bu değerden kendi satırına ve ilişkide olduğu programların satırlarına ulaşır.
- GATE_NTW : Program tipi.
- GATE_NAME : Program açıklaması. Online için anlamı yok.

- GATE_HOST : Program çalışacağı IP adresi. Programlar birbirlerine bu Ip yardımıyla mesaj gönderir. Sadece Msmq mesaj gönderilebilir. Açılışta makina IP'si ile bu IP kontrol edilir. Farklıysa program açılmaz.
- GATE_TXN_BOX : Transaction queue. Programın worker thread'leri bu queue'dan mesajları alıp işlerler. Bir program diğerine bu queue yardımıyla işlem mesajını geçirir. Gate'ler arasında belli structure'lar ile işlemler taşınır.
- GATE_ADM_BOX : Admin queue. Programa komutlar bu queue üzerinden gönderilir. Admin thread komutları işler.
- GATE_HSM_BOX : Hsm Server cevap mesajları bu queue'ya düşer. Program Hsm server'in txn queue'suna hsm isteğini gönderir. Hsm server ise bu queue'ya hsm komutunun sonucunu döner. Hsm serverla mesajlaşma senkron yapıldığı için (business thread sonucu bekler), txn queu kullanılmamıştır. Hsm queue'dan alan thread global oHsmCnn objesi içindedir.
- GATE_EVT_BOX : Event server expire olan eventleri bu queue'ya atar. Switch tarzı programlar için anlamlıdır (bkm, bonus, visa, mc ..). Acquiring için mesaj gönderiminden önce event ayarlanır ve belli bir sürede cevap gelmezse event server bu queue'ya mesaj gönderir. Evt queue'dan alan reversal thread vardır.
- GATE_H2H_BOX : H2h Gate programına atılan core bank mesajlarının cevapları buraya düşer. Senkron haberleşme olduğundan ayrı queue kullanılmıştır. Global oNwmHst objesi içindeki bir thread bu queue'dan mesajları alır.
- GATE_EMV_BOX : Emv & Script gate'in cevap mesajları buraya gelir. Emv gate ile senkron konuşulduğundan ayrı bir queue tanımlanmıştır. Global oNwmEmv objesi içindeki receiver thread bu queue'dan mesajları alır.
- GATE_CRD_BOX : Prepaid gate'den kredi kartı otorizasyona senkron gidilmesi durumunda cevap mesajları buraya düşer. Distributed transaction mantığı kurmak yerine prepaid tarafında tek bir transaction içinde halledilmiştir. Senkron mesajlaşma olduğudan ayrı bir queue açılmıştır.
- GATE_CNN_TYPE : Gate'in dış bağlantılarının tipi (C – Client, S – Server). Bazı gate'ler içeride hardcoded bağlantı yönünü set ediyor olabilir.
- GATE_CNN_FRM : Dış bağlantı mesaj formatı. GEnel olarak kullanılmıyor. Gate'ler bağlantı formatlarını kendileri hardcoded set ediyor.
- GATE_CNN_TOV : dışarıya mesaj gönderen switch tarzı gate'lerde cevap bekleme süresi (milisaniye cinsinden). Bkm, visa gibi gate'lerde acquiring işlemler için geçerlidir.
- SND_RVR_CNT : Reversal repeat sayısı.
- SND_RPT_CNT : Kullanılmıyor. Advise repeat sayısı.
- SND_RVR_TYPE : hatalı kodlanmış bir parametre. R verilmesinde fayda var.
- SND_ECH_MSG : kullanılmıyor.
- SND_ECH_INT : Echo mesajı gönderme sıklığı. Saniye cinsinden.
- SND_SGN_ON_MSG : Açılışta yada connection gidip geldiğinde otomatik signon mesajı gönderilir.

- GATE_CCH_UPD_TYPE : cache update tipi (I-Interval, E-Exact)
- GATE_CCH_UPD_TIME : cache update parametresi. Interval ise kaç dakikada bir cache refresh edileceği. Exact ise, HHMM cinsinden saat verilir.
- GATE_TXN_BOX_TTL, GATE_ADM_BOX_TTL, GATE_HSM_BOX_TTL, GATE_EVT_BOX_TTL, GATE_H2H_BOX_TTL, GATE_CRD_BOX_TTL : her bir queue tanımı için bekleme süresi set edilmiştir. Değeri 0'dan büyük ise ve mesaj queue'da bu kadar süre beklemişse çöpe atılması gerekir. Bu değer transaction queue haricinde kullanılmuıyor.
- GATE_UP : Program kapalı olarak duracaksa N yapılır. Böylece Monitoring ekranlarında uyarı verilmemiş olur.
- ONL_STAT : program kapanırken 0 yapılır. Böylece diğer programlar kapandığını anlayıp mesaj göndermeyi bırakır. Göçmelerden farklı olarak time update olmasına bakılmaz. Program açıksa bu alanı 1 olarak update eder.
- LAST_ONL_TIME : Programlar bu alanı 4 saniyede bir update eder. Bir programın göçtüğünü bu alanın değişmemesinden anlayabiliyoruz.
- LAST_WRK_TIME : önemsiz. Kullanmıyoruz.
- MSG_FLD_CODE : ISO8583 mesajlarının field detayları ve mti-pcode değerlerinden işlem kodu bulunması bu değere göre yapılır. SYS_GATE_FLD_PRF ve SYS_GATE_TXN_DET tablolarındaki MSG_CODE alanına bir referanstır. ISO mesaj alıp göndermeyen bir gate için anlamı yoktur.
- LOG1 : primary log server uygulamasının kodu.
- LOG2 : secondary log server uygulamasının kodu (opsiyonel).
- MON1 : primary monitoring server uygulamasının kodu.
- MON2 : secondary monitoring server uygulamasının kodu (opsiyonel).

OC_SYS.SYS_GATE_CNN

Dış dünya ile konuşan gate'lerin TCP connection ayarları buradan yapılır. Burada yapılan ayarlar sadece işlem mesajlarının akışında kullanılan bağlantılar içindir. Örneğin fraud, loyalty gibi ek bağlantılar başka tablolardan alınır. Tabloya Client yada Server bağlantı tanımı yapılabilir. Bağlantı tipine göre ONwmTcm yada ONwmTcc isimli client yada server objelerinden biri kullanılır. Ip adresi yoksa server tanımı kabul edilir.

Server tipinde tek port tanım yapılabilir. Client olduğumuz durumda ise istenen sayıda Ip-port çifti tanımlanıp hepsi ayrı session olabilir. Ancak genellikle her program tek session ile çalıştığından çoklu session'da sorun olma ihtimali az da olsa olabilir.

- MBR_ID : Programların member ID alanları kullanılmıyor. Gate code unique zaten.
- GATE_CODE : program kodu. Gate tablosundan.
- GATE_SSN_IDX : Session ID değeri. Sessiomn ve key tablolarına erişimde kullanılır.
- PRM_HOST_ADDR : Primary Host IP (port dinlenecekse boş)
- PRM_HOST_PORT : Primary Host Port No.
- SEC_HOST_ADDR : Secondary Host IP
- SEC_HOST_PORT : Secondary Host Port No.

Server olunan durumda ikinci port tanımı gereksizdir. Client olduğumuz durumda her session için 2 adres girilebilir. 3 kez ilk adrese bağlantı kurulmazsa ikinci adres denenir.

OC_SYS.SYS_GATE_SSN

SYS_GATE_CNN tablosuyla yakından ilişkilidir. Aslında 2 tablo birleşse olabilirmiş. Genelde Bkm, Visa, Mc gibi switch tarzı programlarda session manrığı vardır ve buraya otomatik kayıt atılır. Buradan sessin key tablosuna da referans vardır.

- GATE_CODE : Program kodu.
- SSN_IDX : Session Id. SYS_GATE_CNN tablosundaki ile aynıdır. Program bazında unique olmalıdır. 1'den büyük molmalıdır.
- KEY_IDX : Session üzerindeki aktif key numarası. Key exchange yapıldıkça değişebilir. Session üzerinde bir çok key olsa bile biri aktiftir.
- CNN_STAT : Aktif update olmuyor. Bağlantının ayakta olduğunu gösterecekti.
- SGN_STAT : SignOn olunduğunda Y yapılır. Signoff olunduğunda N yapılır.
- ADV_STAT : Visa advice On mesajı gönderildiğinde Y yapılır.
- CNN_DATE : Connection statüsünün son değişme tarih saati
- SGN_DATE : Son signon mesajı tarih/saati
- ADV_DATE : Son visa advise on mesajı tarih/saati
- ECH_DATE : Son echo mesajı tarih/saati
- KEY_DATE : Son key exchange mesajı tarih/saati

OC_SYS.SYS_GATE_SEC

Session üzerindeki ZMK ve ZPK keylerinin değerleri tutulur. Key exchange yapılıyorsa ZMK olmalıdır (bkm). Sabit ZPK da olabilir. Bu durumda ZMK girilmesine gerek yoktur (VISA). Key exchange sırasında key index de set edilebilir. Bu durumda aynı session üzerinde bir çok ZPK key olabilir. Ancak bir tanesi aktiftir ve aktif olan SYS_GATE_SSN tablosunda belirtilir. Aynı indexli yeni bir key geldiğinde eskisi ezilir.

- GATE_CODE : uygulama unique numarası
- SSN_IDX : session Id değeri. SYS_GATE_SSN ve SYS_GATE_CNN tablolarındaki session id değerleri ile aynı.
- SSN_KEY_TYPE : Key tipi.
  M – ZMK
  A – ZPK (sadece acquiring için)
  I – ZPK (sadece issuing için)
  B – ZPK (hem acquiring hem de issuing için)
- SSN_KEY_IDX : Key index değeri. Mesajlarda key index gelip gidiyorsa set edilir. Yoksa 1 olarak set edilir ve tek key olur.
- SSN_KEY : Key değeri HSM LMK'sı altında.
- SSN_KCV : Key KCV değeri.
- LAST_DATE : Key'in son değişme tarih/saati (sadece ZPK keyleri için key exchange zamanında set edilir).

OC_SYS.SYS_TXN_DST

Programlarda işlemlerin hangi gate tipine ve oradan hangi dış bağlantıya gönderileceğini set etmekte kullanılır. Bir işlem tablodaki pek çok satıra uyabilir ancak priority değeri en düşük olana gönderilir. Satırlarda boş olan kriterler kontrol edilmez.

- SRC_NTW : kullanılmıyor (source network type).
  * - tüm networkler
  P – Pos
  Z – Atm vb.
- PRIORITY : ilgili yönlendirme tanımının önceliği. Küçük değerler daha öncelikli.
- BIN_SRC : Kart kaynağı.
  O – Onus
  D – domestic
  …
- BIN_DCI : kart de bit credit indikatör (C – Credit, D – Debit)
- BIN_BRD : Kart brand (V – Visa , M – Mastercard …)
- BONUS_TXN : İŞlem bonus mu (Y / N) . Kaldırılabilir.
- BONUS_BIN : Bonus binimi (Y / N) . Kaldırılabilir.
- TXN_TRM : Terminal tipi (PO – Pos, AT – Atm, VP – Vpos …)

- BIN : Kartınilk 6 hanesine göre de yönlendirme yapılabilir. BIN bazlı yönlendirmeler en öncelikli olmalıdır.
- NTW_FWD : İşlemin hedef networkü (B – Bkm, V – visa, C – Credit …)
- NTW_DRC : işlemin gönderileceği ara network. Örneğin visaya gidecek bir mesaj, direk visa bağlantısı olmadığında bkm üzerinden visaya gönderilir. Bu durumda bu alan bkm olur.

OC_SYS.SYS_PATH

Programlar arası patj tanımları yapılarak hangi program hangisine mesaj gönderebilir belirtilir. Yük dağılımı ve yedekleme için priority kolonu da kullanılır.

- MBR_ID_SRC : Kullanılmıyor. kaynak programın member ID'si.
- APP_TYPE_SRC : Kaynak programın tipi. Online için anlamsız. Kozmetik olarak doğru olmasında fayda var.
- APP_CODE_SRC : Kaynak programın unique kodu (online için öneml molan alan)
- MBR_ID_DST : Kullanılmıyor. Hedef programın member ID'si.
- APP_TYPE_DST : Hedef programın tipi. Online için anlamsız. Kozmetik olarak doğru olmasında fayda var.
- APP_CODE_DST : Hedef programın unique kodu (online için öneml molan alan)
- PRIORITY : Bağlantının önceliği. Aynı program tipinden programlara birden fazla path varsa en düşük öncelikli olanlar kullanılır. Düşük öncelikli programa erişilemezse bir sonraki öncelikli olan denenir Aynı öncelikli pathler arasında yük dağılımı yapılır.

OC_SYS.SYS_GATE_FLD_PRF

ISO8583 mesajlarını gönderirken filtrelemeyi sağlar. Kod içinde gerekli olup olmamasına bakılmaksızın ISO field'ları parsera eklenmektedir. Bu yüzden MTI bazında gereksiz alanların çıkarılması sağlanır. Ayrıca mandatory alanlar boş olsalar bile mesaja eklenirler. Gerçi bu şekilde eklemek sorunlu olabilir.

Field detaylarına bakıldığında şu şekilde bir değer görülür : "-MMM-MM--M;MM".

Her karakter 1 field'I temsil eder. Ilk karakter ISO field 1 (bitmap) alanıdır ve bu şekilde devam eder. Aradaki ";" karakterleri atlanır. Sadece gösterimde 10'lu bloklar anlaşılsın diye konmuştur. Field özelliklerinin anlamı şu şekildedir.

  - "M" – Mandatory
  - "C" – Conditional
  - "-" – Field yok

- GROUP_CODE : Field gruo tipi. Gate tablosundaki GROUP_CODE alanından buraya bir referans vardır. Aynı tipteki gate'lerin aynı parametreleri kullanmasını sağlar.

- MTI : gönderilen mesajdaki mti değeri.
- FLD_PRF : Mti değerine göre gönderilecek alanların filtrelenmesi

OC_SYS.SYS_GATE_TXN_DET

Gelen ISO8583 mesajındaki mti ve processing code alanlarından internal işlem tipini bulmaya yarar. Burada bulunan mesaj tipi (OTC-OTS) kod içinde bazı kontrollerde değişebilmektedir.

- GROUP_CODE : Field gruo tipi. Gate tablosundaki GROUP_CODE alanından buraya bir referans vardır. Aynı tipteki gate'lerin aynı parametreleri kullanmasını sağlar.
- MTI : Gelen mesajdaki MTI değeri (key)
- PCODE : Gelen mesajdaki Processing Code (F3) alanının değeri (key)
- OTC : Bulunan internal tracsaction Code
- OTS : Bulunan internal tracsaction Sub Code
- OTE : Bulunan internal tracsaction Effect

OC_SYS.SYS_HSM_CONN

Hangi hsm server'in hangi Hsm yada hsm'lere bağlanacağını gösteren bir tablodur. SYS_HOST ve SYS_HSM_DEVICE tabloları ile birlikte düşünülmelidir.

- HOST_CODE : Hsm server programının unique numarası
- HSM_CODE : Hsm'in unique ID'si SYS_HSM_DEVICE tablosuna referanstır.
- CONN_COUNT : Belirtilen Hsm server'in belirtilen hsm'e kaç tane TCP connection açacağını gösterir.

OC_SYS.SYS_HSM_DEVICE

Kullanılan Thales Hsm'lerin tanımlandığı tablodur. Seri çalışan HSM'imiz yok ve desteklemiyoruz ancak seri port parametreleri de bulunmaktadır.

- HSM_CODE : Hsm'e verilen unique bir kod değeri. HSM_CONN tablosunda kullanılır. Anlamlı değerler verilir (ONL1, ONL2, PIN …)
- HSM_NAME : Açıklama. Online için anlamsız.
- HSM_NTW_TYPE : Hsm bağlantı tipit. (E – Ethernet). Sadece bu desteklenir.
- HSM_HOST : Hsm IP Adresi
- HSM_PORT : Hsm port numarası
- HSM_SLOT_CNT : Kullanılmıyor (Eskiden HSM_CONN tablosu yoktu ve tüm hsm serverlar bu hsm'e aynı sayıda connection kuruyordu).
- ASYN_DATABITS : Kullanılmıyor . Seri Port Parametresi
- ASYN_STOPBITS : Kullanılmıyor . Seri Port Parametresi
- ASYN_PARITY : Kullanılmıyor . Seri Port Parametresi
- HSM_ECHO_TOV : Hsm'e echo mesajı gönderme süresi. Periyoidk olarak NC mesajı gönderilmektedir. Saniye cinsinden.

- HSM_ACTIVE : Kullanılmıyor.
- PRINTER_FLAG : “Y” ise Hsm’e printer bağlı olduğunu gösterir. Pin print fonksiyonları sadece bu flag’li hsm’leri kullanır.
- HSM_TYPE : Kullanılmıyor. R – Racall. Diğer hsm’leri kullanmadık henüz.

OC_SYS.SYS_BANK_CNN

Core Bank bağlantılarının tanımlandığı tablodur. H2h Gate programından kullanılır. Bu tabloda fonksiyon koduna göre farklı hostlara gidilmesi ve timeout değeri set edilmesi mümkündür. Genelde tek tanım yapılır (FUNC_CODE = 0000 ise default).

- FUNC_CODE : Bankacılık fonksiyon kodu (0000 -default)
- DST_HOST1 : Primary Host IP.
- DST_PORT1 : Primary Host Port No.
- DST_HOST2 : Secondary Host IP.
- DST_PORT2 : Secondary Host Port No.
- MSG_TOV : Mesaj cevap bekleme süresi
- ADD_TOV : ???

OC_SYS.SYS_BANK_RPT

Bazı bankacılık mesajlarında cevap önemli değildir. Mesaj queue’ya atılır ve belli sayıda tekrar edilir. Reversal mesajları her zaman böyledir. Sadece fiş kesilmesi gereken bazı mesajlarda da cevap beklenmez. Cevap beklenmeyen mesajlar bu tabloya girilir. Ayrıca kaç kez gönderim deneneceği ve gönderim aralığı da set edilir.

- FUNC_CODE : direk onay dönülecek bankacılık fonksiyon kodu
- RETRY_INTERVAL : tekrar deneme süresi
- RETRY_COUNT : tekrar deneme sayısı
- RETRY_TOV : cevap bekleme süresi

OC_SYS.SYS_CUSTOM_KEY

Online işlemler sırasında kullanılan bazı custom keyler bu tabloya girilebilir. Key id ile ilgili keye ulaşılır.

- MBR_ID : member id kullanılmıyor.
- KEY_CODE : key tipi
- KEY_VAL : Key değeri. Genellikle Hsm LMK altındadır.
- KEY_DSCR : Key açıklaması. Online için anlamı yoktur.

OC_SYS.SYS_SEC

Online programlarda set edilmiş olan IRC değerleri bu tabloya girilir. IRC değerine göre alınacak aksiyonlar, dönüş kodları, gönderilecek hata mesajları gibi bilgiler burada tanımlanabilir.

- MBR_ID : kullanılmıyor.

- APP_CODE : kullanılmıyor. Tüm programlar ortak havuzu kullanıyor.
- APP_IRC : Irc kodu. Programlarda set edilen unique değer.
- APP_RC : işlemde dönülecek ISO response code değeri
- SRC_TYPE : IRC değerini üreten programın tipi (online için anlamsız)
- NUM_RC : irc değerinin son 4 hanesi (online için anlamsız)
- SND_MAIL : Kullanılmıyor. Mail gönder.
- SND_SMS : Kullanılmıyor. Sms gönder.
- SND_MON : Kullanılmıyor. Monitoring'e event gönder.
- DSCR : Hata açıklaması
- DSCR_ENG : Hata açıklaması ingilizce.
- POS_RC : Onus Pos terminallerine dönülecek hata kodu. Pos terminali için ayrı bir kod girilmemişse, APP_RC değeri dönülür. 3 hane olabilir.
- POS_SCREEN_MSG : Onus pos terminalleri için ekrana basılacak hata mesajı (opsiyonel)
- POS_PRINT_MSG : Onus pos terminalleri için slibe basılacak hata mesajı (opsiyonel)
- BANK_RC : Onus Şubelere dönülecek hata kodu. Set edilmemişse APP_RC değeri dönülür.
- BANK_MSG : Onus Şubelere dönülecek hata mesajı.