기술-2006-019
2006. 12

사례제시를 통한 RFID 적용 본사, 지사간
자산출입관리 시스템 구축 가이드

2006. 12

NIDA
한국인터넷진흥원

사례제시를 통한 RFID 적용 본사, 지사간
자산출입관리 시스템 구축 가이드

◎ 연구책임자 단 장 김 원 (한국인터넷진흥원)

◎ 내부연구원 팀 장 나정정 (한국인터넷진흥원)
대 리 이승재 (한국인터넷진흥원)
대 리 임현덕 (한국인터넷진흥원)
연구원 고현봉 (한국인터넷진흥원)
연구원 김인혜 (한국인터넷진흥원)

◎ 외부연구원 부 장 한재종 (한도하이테크)
대 리 이후석 (한도하이테크)
주 임 최병진 (한도하이테크)

서 문

우리나라는 그동안 지식정보사회 일등국가 건설을 목표로 21세기 정보화 선진국으로 발돋움하였으며, 세계 최고 수준의 정보통신 인프라를 구축하여 IT 선도 국가로 급부상 하였습니다.

이를 바탕으로, 지속적인 IT산업 발전을 위한 정부의 u-IT839 전략은 첨단 정보통신 서비스 도입을 가속화하고, 유비쿼터스 IT시대를 주도할 신성장 발판을 마련하는 등 IT 산업의 기본적인 발전방향을 제시하였습니다.

특히, u-IT839 전략 중 RFID/USN 분야는 인간과 사물, 사물과 사물의 네트워킹을 통해 언제, 어디서나 원하는 정보를 주고받을 수 있는 유비쿼터스 사회의 핵심 인프라로서, 가까운 미래에 우리의 삶의 질을 더욱더 풍요롭게 해줄 것으로 기대되고 있습니다.

한국인터넷진흥원은 RFID 시장 활성화를 위해 2004년부터 RFID 검색시스템 구축, 표준화 및 정책 마련을 추진하고, 국내 RFID 서비스 네트워크 연동을 위하여 국제표준인 ISO/IEC 15459 기반의 코드체계를 정립하는 등의 최선의 노력을 다하고 있습니다.

이번에 출간되는 '사례제시를 통한 RFID 적용 본사, 지사간 자산출입관리 시스템 구축 가이드'는 우리원의 RFID 서비스 시장 활성화를 위한 또 다른 노력의 결실로써, 올해 NIDA에서 실제 구축 완료한 RFID 자산출입관리 시스템을 중심으로 RFID 시스템 구축 기술에 대한 상세한 설명을 담아 책자로 발간하게 되었습니다.

끝으로, 본 책자가 RFID 시장 확산을 위한 기술적 토대를 마련하고 향후 유비쿼터스 환경의 국민 삶의 질 향상에도 기여할 수 있는 유용한 자료로 활용될 수 있기를 기대합니다.

2006년 12월
한국인터넷진흥원
원장 송관호

목 차

서문 ····· **ii**

제1장 개요 ····· **1**
1.1 발간배경 ····· 1
1.2 목적 및 범위 ····· 2
1.2.1. 목적 ····· 2
1.2.2. 내용 및 범위 ····· 3
1.3 사업 소개 ····· 5

제2장 RFID 검색 서비스 ····· **10**
2.1 RFID 개념 ····· 10
2.2 RFID 네트워크 ····· 12
2.2.1. RFID 네트워크의 개요 ····· 12
2.2.2 RFID 네트워크의 구성요소 ····· 13
2.2.3. RFID 네트워크 시나리오 ····· 17

제3장 코드체계 설계 및 적용 ····· **23**
3.1 kCode 개요 ····· 23
3.2 kCode 체계 ····· 24
3.2.1. 코드체계 구현 ····· 24
3.3 kCode 적용 예제 ····· 34
3.3.1. 개요 ····· 34
3.3.2. 코드 인코딩 ····· 34
3.3.3. 코드 디코딩 ····· 43

제4장 자산출입관리 시스템 구축 사례 ····· **48**
4.1 도입 배경 ····· 48
4.2 시스템 구성 ····· 48
4.2.1. 시스템 구성내용 ····· 48
4.2.2. RFID 장비 및 미들웨어 소개 ····· 51
4.2.3. 검색서비스(ODS) 연동 방안 ····· 56
4.2.4. 객체정보서비스(OIS) 구현 방안 ····· 65
4.2.5. 객체이력관리서비스(OTS) 구현 방안 ····· 72
4.2.6. 로컬 ODS 설치 ····· 76

제5장 기대효과 및 의의 ····· **78**

참고문헌 ····· **80**

용어정리 ····· **81**

표 목 차

[표 1-1] 본 지침서의 각 장별 내용 및 범위 요약 ····· 4
[표 1-2] 시스템 주요구성 요소별 요약 ····· 6
[표 1-3] 본 시스템에서 구현된 클래스/패키지 목록 ····· 6
[표 2-1] 시스템 주요구성 요소별 요약 ····· 16
[표 3-1] 태그 데이터 코드 체계 적용안 ····· 24
[표 3-2] NIDA 본사의 자산번호-RFID코드 매핑테이블의 일부분 ····· 25
[표 3-3] ISO/IEC 15459-4의 kCode 예제 분석 ····· 25
[표 3-4] 분류코드 구분 ····· 27
[표 3-5] kCode를 위한 ISO/IEC 18000-6 Type C 태그 설정 ····· 31
[표 3-6] Code Assignments for ApplicationFamilyId ····· 32
[표 3-7] DSFID 세부구조 ····· 35
[표 3-8] Precusor의 세부구조 ····· 36
[표 3-9] Object의 세부구조 ····· 37
[표 4-1] 본 구현에 사용된 고정형 리더 사양 ····· 51
[표 4-2] 본 구현에 사용된 안테나 사양 ····· 52
[표 4-3] 본 구현에 사용된 휴대형 리더 사양 ····· 54
[표 4-4] 본 미들웨어 시스템에 탑재된 기능 리스트 ····· 55
[표 4-5] 본 OIS 구현에 사용된 DB 구성 ····· 66
[표 4-6] 데이터 유형에 따른 OIS 인터페이스의 역할 ····· 66
[표 4-7] 본 OTS 구현에 사용된 DB 구성 ····· 72
[표 5-1] 본 구현의 기대효과 및 의의 ····· 78

그 림 목 차

[그림 1.1] NIDA RFID 자산출입관리 시스템 구성도 ····· 5
[그림 2.1] Web 서비스와 RFID 서비스의 비교 ····· 13
[그림 2.2] RFID 네트워크 구성도 ····· 15
[그림 2.3] 객체 정보 검색 과정 ····· 17
[그림 2.4] 객체 이력정보 등록 과정 ····· 19
[그림 2.5] 객체 이력정보 검색 과정 ····· 21
[그림 3.1] KS X ISO/IEC 15459를 준용한 코드 예시 ····· 26
[그림 3.2] kCode 디코딩 솔루션 구성 ····· 43
[그림 3.3] kCodeClass.cs 디코딩 프로그램 실행화면 결과 ····· 46
[그림 4.1] 출입 게이트 도식도 ····· 48
[그림 4.2] 자산 조회 및 실사 서비스 구성도 ····· 49
[그림 4.3] 시스템 구성도 ····· 49
[그림 4.4] 새 프로젝트 생성 ····· 57
[그림 4.5] 참조 추가 1 ····· 58
[그림 4.6] 참조 추가 2 ····· 58
[그림 4.7] 참조 추가 3 ····· 59
[그림 4.8] 참조 확인 ····· 60
[그림 4.9] BIND 실행 ····· 63
[그림 4.10] .NET Resolver 실행 예제 ····· 64
[그림 4.11] OIS 서버 접근 절차 ····· 68
[그림 4.12] 자산 통과시 모니터 실행 화면 화면 ····· 71
[그림 4.13] OTS 서버 접근 절차 ····· 73
[그림 4.14] 관리자 페이지의 자산 이력 조회 화면 ····· 75

제1장 개요

1.1 발간배경

정부의 u-IT839 전략은 우리나라 IT 산업 정책 전반을 포괄하는 최상위 전략으로 유비쿼터스 IT시대를 주도할 신성장 발판을 마련하고, IT 산업의 기본적인 발전 방향을 제시하였다.

특히 u-IT839 전략 중 RFID/USN 분야는 8대 서비스, 3대 인프라, 9대 신성장 동력에 모두 포함될 정도로 산업 활성화를 위한 미래 유망산업으로 손꼽히고 있으며, 정부는 관련 신기술 및 원천기술 개발을 위한 정책을 추진하고 있다.

관련하여 정부는 RFID/USN 활성화를 위해 공공분야의 시범사업을 확대하고 국산제품의 현장시험과 애로사항을 지원하는 종합시험센터를 구축하는 등 RFID/USN의 본격적인 확산을 위한 정책을 추진 중이다.

이와 더불어, 한국인터넷진흥원(이하 'NIDA'로 지칭한다.)은 2004년부터 RFID 서비스간 상호 운용성 제고를 위해 RFID 검색시스템 구축, 표준화 및 정책 마련을 추진하고, 국내 RFID 서비스 네트워크 연동을 위하여 국제표준인 ISO/IEC 15459 기반의 코드체계를 정립하였다. 이를 통해 RFID 서비스 연동기반을 제공함으로써 RFID 보급 확산을 위한 서비스 활성화에 기여하고자 한다.

이에 따라, NIDA는 그간 "RFID 검색시스템 구축 및 운영 지침서"를 발간하여 RFID 사업 기관에게 검색서비스 구축을 위한 가이드라인을 제공하였으며, 또한 코드체계를 태그에 기록하고 해석하는 방법에 관한 "RFID 코드 인코딩 지침서"를 발간하여 관계기관에게 제공하는 등 최선의 노력을 기울여왔다. 이와 더불어 NIDA에서는 RFID 검색서비스에 대한 이용자의 이해도를 높이고, 서비스 활성화를 위해 구축·운영에 필요한 노하우를 정리하여 본 가이드를 발간하였다.

본 가이드를 통해 올해 NIDA에서 구축한 "RFID 적용 자산출입관리 시스템" 사례를 공개함으로써, RFID 기술 이해도를 높이고, 검색서비스(ODS), 객체정보서비스(OIS), 객체이력관리(OTS) 및 ISO/IEC 15459 기반의 코드체계 등 관련 기술에 대한 실증 사례를 제시하고자 한다.

1.2 목적 및 범위

1.2.1. 목적

본 사업에서는 향후 RFID 네트워크의 안정적인 연동과 확장을 도모할 수 있도록 글로벌 RFID 네트워크상에서 ISO/IEC 15459 국제 표준 코드체계를 준용하고 기존에 개발된 RFID 검색서비스를 이용, NIDA 본사-지사 간을 연동시켜 자산 출입관리 시스템을 구축하여 제정된 표준 코드 체계의 검증 및 검색서비스의 활용사례 제시를 최종 목표로 갖는다.

최종목표		
RFID 검색서비스를 연동한 NIDA 자산 출입관리 시스템을 구축하여 kCode 표준안 및 RFID 검색서비스(ODS)의 실제 적용 사례를 제시		
번호	**구축내용**	**세부구현내용**
1	RFID 검색서비스(ODS) 실증 사례 제시	● OIS,OTS 구현 ● ODS와 미들웨어간의 메시지 통신 구현 ● 글로벌 RFID 네트워크 확장을 위한 National-ODS 활용 ● RFID 네트워크를 활용한 토탈 자산관리시스템 구축
2	kCode 표준(안)에 의한 코드체계 적용 사례 제시	● 자산용 RFID 태그에 KS 표준(안) 코드 인코딩 ● kCode 디코딩 모듈 구현
3	RFID 자산출입관리 자동화 시스템 구축	● NIDA 자산 보유 현황 파악 가능 ● PDA를 이용한 자산실사 기능 ● 자산조회/관리 효율성 향상 ● NIDA 자산 RFID Tag 부착, 조회 및 이력관리

1.2.2. 내용 및 범위

본 문서는 NIDA에서 구축한 RFID 검색시스템을 바탕으로 kCode 표준안에 기반한 RFID 검색시스템의 각 요소를 구축, 운영하고 미들웨어 혹은 응용 프로그램과 연동하기 위한 지침을 제공한다. 각 장별 대한 내용 및 범위는 다음과 같으며 [표 1-1]에 내용을 요약/정리 되어있으므로 참고하기 바란다.

2장. RFID 검색서비스 개요

RFID에 관한 기본 지식을 간단하게 소개한다. RFID 네트워크의 기본적인 구성 요소와 H/W에 대해 간략히 설명하며 이러한 RFID 네트워크에 기반하여 운영되는 RFID 검색서비스에 대한 내용으로 구성된다.

3장. 코드체계 적용

차후 글로벌 RFID 네트워크 확장을 위해서는 국제 표준에 맞는 코드체계가 필수적이며 이러한 코드체계의 개요에 관해 언급한다. 본 사업에 적용된 kCode는 ISO/IEC 15459 국제 표준에 준하여 NIDA에서 제정된 코드체계 표준안으로서 향후 RFID 관련 사업을 위한 국내 표준 RFID 코드체계 관련 지침을 제공한다. 또한 코드체계를 적용하기 위한 구체적인 예제와 함께 구현방법을 설명한다.[1)]

4장. 자산출입관리 시스템 구축 사례

본 사업에서 구축된 자산출입관리 시스템에 관하여 소개하는 부분이며, 구축 개요 및 내용에 대해 구체적으로 언급한다. 시스템 구성도 및 실제 쓰인 장비의 사양과 더불어 미들웨어에 쓰인 기능정의를 설명한다. 또한 NIDA에서 서비스 중인 RFID 검색서비스와 미들웨어를 연동시키는 부분에 대하여 소스코드와 함께 구체적으로 설명하는 중점적인 부분이다. 더불어 이번에 제작된 OIS, OTS의 간단한 기능을 소개한다.

5장. 기대효과 및 의의

본 구현에서는 검색 서비스의 효용성을 이용한 자산출입관리 시스템을 구축하였다. 구축된 시스템은 자산관리 부문에서 대부분 수동으로 진행되던 프로세스를 상당 부분 자동화하였으며 국제 표준을 준용한 코드체계인 kCode와 RFID 검색서비스를 성공적으로 적용하였다. 이에 예상되는 기대효과 및 의의에 대해 간략히 기술한다.

---

1) ISO/IEC 15459 KKR 코드체계는 구현 단계에서, KS 입안 예고된 상태이며, 향후 표준화 진행 방향에 따라 변경될 수 있다. 또한 본 책자에서 적용된 ISO/IEC 15459 KKR 코드체계는 'kCode'로 지칭한다.

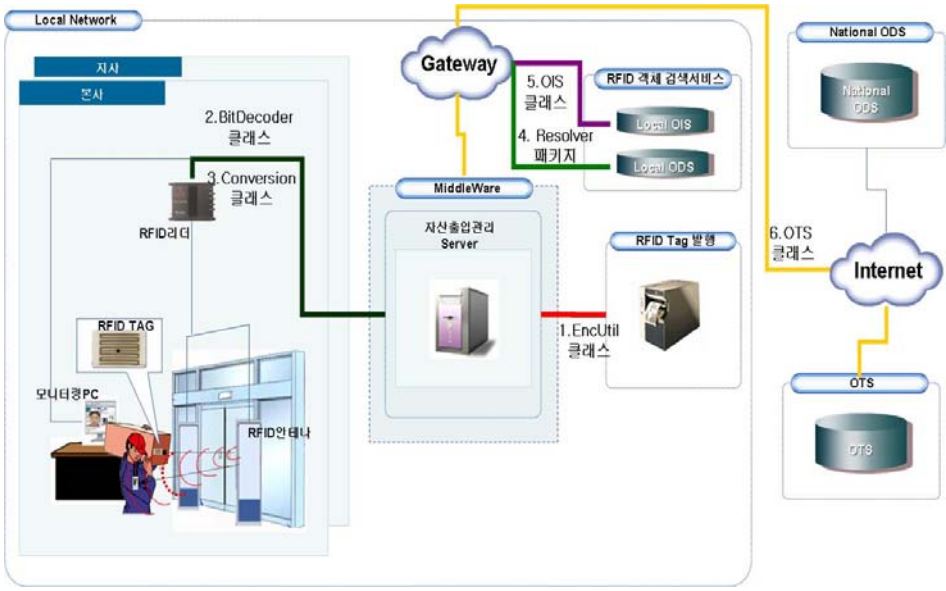
각 장별 내용 및 범위 요약		
순서	**제목**	**내용 및 범위**
2장	**RFID 검색서비스 개요**	● RFID에 관한 기본 지식 ● RFID 네트워크의 기본적인 구성요소 ● RFID 검색서비스 소개
3장	**코드체계 적용**	● 표준 코드체계의 개요 ● ISO/IEC 15459 와 kCode ● 향후 RFID 관련 사업을 위한 국내 표준 RFID 코드체계 관련 지침 제공 ● 코드체계를 적용하기 위한 구체적인 예제와 함께 구현방법 설명
4장	**자산출입관리 시스템 구축 사례**	● 자산출입관리 시스템 구축 개요 ● 자산출입관리 시스템의 기본 구성 ● 시스템 구성도 및 실제 쓰인 장비의 사양 ● 미들웨어에 쓰인 기능정의 설명 ● RFID 검색서비스와 미들웨어의 연동 (소스코드 및 예제화면 첨부) ● 시스템에 들어간 OIS, OTS의 기능 소개
5장	**기대효과 및 의의**	● 검색 서비스의 효용성을 이용한 자산출입관리 시스템 구축 ● 자산관리 부문에서 수동으로 진행되던 프로세스 상당부분을 자동화 ● 국제 표준을 준용한 코드체계인 kCode와 RFID 검색서비스를 성공적으로 적용 ● 위 사항에 의해 예상되는 기대효과 및 의의에 대해 간략히 기술

[표 1-1] 본 지침서의 각 장별 내용 및 범위 요약

1.3 사업 소개

본 사업은 RFID 객체 검색 서비스를 활용하여 사내 자산의 위치 정보와 현황, 자산 출입 관리와 이력을 실시간 확인할 수 있는 시스템을 구축하는 데 그 목적이 있다. 즉 기존의 RFID 객체 검색 서비스와 연계하여 우수한 품질을 가진 자산의 등록/삭제, 자산의 반입반출 관리 서비스를 구현하는 것을 목표로 한다. 또한 본 사업에 적용된 kCode는 ISO/IEC 15459 국제 표준에 준하여 NIDA에서 제정된 코드체계 표준안으로서 2006년 7월 KS 입안예고 된 상태이다. 본 구현은 이러한 kCode 체계를 적용하여 구현되었다.

위와 같은 최신 RFID 관련 기술 및 kCode를 적용한 본 시스템은 NIDA의 본사 및 지사의 자산출입을 통합관리 할 수 있게 설계 되었으며 그에 따른 개략적 시스템 구성도를 표시하면 다음과 같다.



[그림 1.1] NIDA RFID 자산출입관리 시스템 구성도

NIDA의 본사와 지사의 각 출입문에는 현재 RFID 안테나와 RFID 고정형 리더가 설치되어 있어 태그가 부착된 자산 출입시 태그로부터 RFID 코드를 읽어서 미들웨어에 전달한다. 미들웨어는 자산 출입이 일어날 때마다 RFID 코드값을 받아 각

본사/지사별로 구축되어있는 Local ODS로부터 해당 RFID 코드에 해당하는 객체정보를 검색하여 OIS에서 정보를 받아와 모니터링 PC 에 표시하고 Local OIS에 해당 객체의 출입이력을 기록한다. 기록된 이력 정보는 관리자가 원할 때 열람이 가능하다. 또한 미들웨어로부터 Local OIS를 조회하여 현재 발행된 태그의 이력을 조회하고 새로운 태그를 발급할 수 있으며 이를 이용하여 자산을 손쉽게 등록/삭제/조회하는 작업이 가능해졌다.

본 문서를 읽는 사람의 이해를 돕기 위해 위 시스템 구성도의 각 구성요소별로 세부사항을 [표 1-2] 에 간략히 요약/정리되어 있으며 실제 시스템 구축시 개발자를 위한 참고자료로 이번 구현에 사용된 클래스들의 기능 및 목록이 [표 1-3] 에 정리되어있으니 참고하기 바란다.[2)]

구성요소	설명
미들웨어 및 서버	RFID 미들웨어 자산출입관리 서버 (H사) Windows 2003 Svr/Ent 및 .Net Framework 1.1 IBM X-Series 226 (Intel Xeon 3.0/UW-SCSI) 채용
RFID Tag	ISO 18000-6C Air Interface 표준규약을 준수한 A 사의 96Bit 태그 사용
Local OIS 및 Local OTS	Solaris Svr.9 Apache 1.3.28, Apache Tomcat 4.2, Java2 SDK 1.4.2_01 에 기반한 AXIS 1.1 로 구현.
Local ODS	Solaris Svr.9 환경에서 Bind 9 에 기반하여 NAPTR 레코드를 이용하여 구현[3)]
모니터링PC	Intel® Pentium® 4 (EIDE 80GB HDD, DDR2 RAM 2GB) Windows XP HomeEdition 및 .Net Framework 1.1
RFID 고정형 리더	A사 Gen 2 Surpported 고정형 리더
RFID 고정형 안테나	A사 x 4EA (Circular 형 안테나)[4)]

[표 1-2] 시스템 주요 구성 요소별 요약

2) 본 시스템 개발 환경은 아래와 같다
- MS사 Visual Studio.net 2003(이하 VS.net) 및 Solaris 9 - JDK 1.4.2
따라서, 배포중인 소스실행을 위해서는 아래 프로그램 환경을 권장한다.
- .net 실행 : VS.net , .java 실행 및 뷰어 : JDK 1.4.2 , 개발용 텍스트 뷰어
※ .java 소스의 경우 Unix 파일로 되어있기 때문에 울트라에디터 등 범용 텍스트 뷰어가 있어야 윈도우즈 환경에서도 열람 가능하다

3) Local ODS : DNS와 동일한 구조를 지니며 BIND 9에서 NAPTR 레코드를 이용하여 구현되었다. 자세한 것은 참고자료 목록의 'RFID 검색 시스템 구축 및 운영지침서 V1.2'을 참고하라.

4) 안테나는 RF 전파 파형에 따라 Linear 형과 Circular 형으로 크게 구분되며 본 구현에서는 태그의 인식 거리보다는 정확도에 중점을 둔 만큼 Circular 형을 채용하였다.

Class/Package Name	Function	설명
1.EncUtil Class (.Net)	● 각종 ISO/IEC 646 인코딩 관련 함수 모음 클래스	● kCode 값을 Tag UID로 변환시켜준다. 태깅 작업시 필요하다.
EncUtil	String BitToHex(string)	● 96비트의 비트 문자열(2진수)을 받아 Tag UID로 변환 ● String 형태의 Tag UID 값을 반환
EncUtil	string StrToBit(string)	● 문자열을 입력받아 5비트 인코딩한 2진수 문자열로 변환 ● 아스키문자에 해당하는 5Bit 인코딩 값을 2진수 스트링으로 반환
2.BitDecoder Class (.Net)	● ISO/IEC 646 표준에 의거하여 비트스트링을 5비트 혹은 6비트 디코딩한 결과를 반환하는 클래스	● RFID 리더기에서 올라온 Tag UID[5] 값을 표준 kCode 로 변환한다.
BitDecoder	String Bit5Char(string)	● 들어온 비트문자열을 5비트 디코딩 결과값으로 반환. ● 디코딩된 문자 1개를 반환하며 에러시 '!'문자 혹은 예외를 반환.
BitDecoder	string Bit6Char(string)	● 들어온 비트문자열을 5비트 디코딩 결과값으로 반환. ● 디코딩된 문자 1개를 반환하며 에러시 '!'문자 혹은 예외를 반환.
3.Conversion Class (.Net)	● 2진수 <-> 10진수 <-> 16진수 상호변환함수를 제공하는 클래스	● Tag UID <-> kCode 변환시 진수 변환에 사용되는 클래스
Conversion	string HexToBin(string)	● Tag UID 문자열을 변환하여 2진수 문자열로 바꿔 반환 ● 에러시 예외를 반환
Conversion	int BinToInt(string)	● 2진수 문자열을 변환하여 Int 형태로 반환 ● 에러시 예외를 반환

5) Tag UID : 16진수 코드의 형태로 된 UID값의 기록형태, 자세한 사항은 용어정리를 참고한다.

4.Resolver Package (.Net)	● kCode 유효성 검사 및 Local ODS 통신을 담당하는 클래스들의 모음	● RFID 리더기에서 올라온 Tag UID 값을 표준 kCode 로 변환한다.
Formatter Class	● 입력 받은 코드에 대한 유효성 검사 수행하는 클래스	● 리졸버클래스가 지원하는 코드체계에 필요한 형식으로 코드변환
Code Analyzer Class	● 넘어온 코드를 해석하여 코드체계를 분류하는 클래스	● 분석된 코드체계에 맞는 URN 형식 작성
Converter Class	● urn -> FQDN 으로 변환하는 클래스	● 넘어온 urn 값의 유효성을 검사후 FQDN 으로 변환하여 DNS Resolver 클래스에 넘겨줌.
DNS Resolver Class	● Local ODS 에 질의하는 클래스	● FQDN 값으로 Local ODS에 질의하여 해당 객체의 정보가 있는 서버의 URL을 받아옴
5.OIS Class (Java)	● M/W 혹은 어플리케이션과 통신하고 데이터 처리를 하기 위한 클래스	● OIS 서버에서 웹서비스 형태로 실행되고 있는 프로세스.
OIS	DataSet GetStaticData(String[] DataParameter)	● DataParameter 로 넘어오는 Tag UID 정보에 해당하는 OIS Static 정보 질의결과를 DataSet 형태로 리턴 ● DataParameter 배열을 분석하여 사용자의 요구에 맞는 정보로 DataSet 을 구성하여 넘겨줌
OIS	DataSet GetHistoryData(String[] DataParameter)	● DataParameter 로 넘어오는 Tag UID 정보에 해당하는 OIS 이력정보 질의결과를 DataSet 형태로 리턴 ● DataParameter 배열을 분석하여 사용자의 요구에 맞는 정보로 DataSet 을 구성하여 넘겨줌
OIS	Bool SetStaticData(DataSet DataParameter)	● DataSet 형태로 넘어오는 데이터를 검사하여 OIS DB 형식에 적합 판단 ● OIS Static DB 혹은 Instance DB에 등록 ● 등록이 성공할 경우는 True, 실패할 경우는 False 값을 반환

OIS	int DelStaticData(DataSet DataParameter)	● DataSet 형태로 넘어오는 Tag UID 정보에 해당하는 OIS 정보를 DB 상에서 삭제 ● 성공적으로 삭제 시 0 이상의 자연수를 리턴, 삭제 실패 시 -1 리턴
OIS	Bool ModifyStaticData(DataSet DataParameter)	● DataSet 형태로 넘어오는 Tag UID 정보에 해당하는 OIS Static & Instance 정보를 찾아 수정 ● 성공적으로 정보를 수정했을시 True, 실패 시 False 를 리턴
OIS	int DelHistoryData(DataSet DataParameter)	● DataSet 형태로 넘어오는 Tag UID 정보에 해당하는 OIS 이력 정보를 Historical DB 상에서 삭제 ● 성공적으로 삭제 시 0 이상의 자연수를 리턴, 삭제 실패 시 -1 리턴
6.OTS Class (Java)	● M/W 혹은 어플리케이션과 통신하고 데이터 처리 하기 위한 클래스	● OTS 서버에서 웹서비스 형태로 실행되고 있는 프로세스.
OTS	DataSet GetTraceData(String TagUID, Date sDate, Date eDate)	● 인자값으로 넘어오는 Tag UID 정보에 해당하는 OIS URIs 를 DataSet 형태로 리턴 ● 검색은 날짜별로 지정해 줄 수 있으며 날짜가 생략된 경우에는 가지고 있는 모든 레코드를 DataSet 형태로 반환
OTS	Bool SetTraceData(String[] DataParameter)	● DataParameter 로 넘어오는 Tag UID 정보를 분석하여 OTS DB에 기록
OTS	int DelTraceData(DataSet DataParameter)	● 삭제할 레코드의 OIS URI 리스트를 검색하여 각 OIS 들에게 해당 레코드의 이력을 삭제해줄 것을 요청

[표 1-3] 본 시스템에서 구현된 클래스/패키지 목록

<소스 다운로드 안내>

본 책자에서 나오는 실제 구현 소스 목록은 [표 1-3]과 같으며, 소스 전체는 **RFID ODS 홈페이지(http://www.ods.or.kr) 자료실**에서 다운로드 가능함

제2장 RFID 검색 서비스

2.1 RFID 개념

RFID(Radio Frequency Identification)는 무선 주파수 인식기술은 20세기 중반에 개발되어 1990년대 말에 재고 관리 및 공급 체인 관리 등에서 사용됨으로써 두각을 드러낸 기술이다. RFID는 앞에서 언급 한데로 무선 주파수 인식 기술을 말하는 것이다. RFID는 주파수를 이용하여 개별 상품을 식별하는 방식을 일컫는다. RFID와 바코드를 비교하여 보면 바코드의 경우 레이저 판독기를 바코드에 직접 접촉시켜야 하지만 RFID는 안테나와 테그만 있으면 판독기를 직접 접촉하지 않아도 쉽게 상품의 정보를 식별할 수 있으며 필요한 정보를 삽입할 수도 있다.

RFID 구성 요소는 데이터를 저장할 수 있는 RFID 태그와 RFID에 있는 데이터를 읽을 수 있는 리더, 그리고 중간에서 데이터를 전송하는 안테나 등으로 구성되어 있다.

유비쿼터스(Ubiquitous)는 라틴어로 '언제 어디서나 있는'을 뜻하는 말이다. 혹자는 지난 50년간 컴퓨팅의 역사를 보면 크게 3가지의 뚜렷한 패러다임이 존재했는데 하나는 메프레임, PC, 그리고 마지막으로 유비쿼터스로 요약할 수 있다고 말한다. 이것을 인간과 컴퓨팅의 관계로 볼 때 메인프레임 시대에는 1개의 컴퓨터에 많은 단말(사람)이 붙어 있었고, PC 시대에는 1개의 컴퓨터에 1사람이, 그리고 유비쿼터스 시대에는 1사람 주변에 수 많은 컴퓨터들이 존재하는 모습이 된다. 인터넷이 등장하면서 온라인, 즉 버추얼 공간 개념이 등장했고 실체를 온라인 공간에 옮기는 것이 지금까지의 인터넷 발달과정인 것에 비해 유비쿼터스는 반대로 모든 실체에 컴퓨팅 공간 개념을 심는 것이다.

이 유비쿼터스를 실현시키는 핵심이 바로 RFID이다. 과거에는 시스템이 개별적인 실체를 인식할 수 없었지만 이제 RFID를 통해 모든 실체들이 무선 네트워크 상에서 인식될 수 있는 존재가 된 것이다. 공급 체인에 있는 모든 품목의 개별적 식별이 가능해짐으로써 공급체인의 효율은 증가하며, 완벽한 유통이 실현되는 것이다.

시스템은 상품을 자동으로 인식하여 그 처리를 하게 된다. 시스템은 사람의 개입 없이 상품을 모니터링하고 판단하고 필요한 조치를 취할 수 있도록 상품 정보를 신속히 검색한다. 그렇게 됨으로써 시스템의 재고와 실제 재고가 일치되고, 거래업체와의 상품 인수인계가 자동화된다. 수많은 점포에 대해 재고관리, 판매, 보충, 도난방지 등 모든 업무가 일괄 관리 및 제어되며, 상품 보충 및 생산 주문이 자동화될 것이다. 판매 계산대로부터 원자재 구매에 이르는 공급체인 업무 전체가 통합된 수요 주도의 네트워크 체제로 형성될 것이다.

2.2 RFID 네트워크

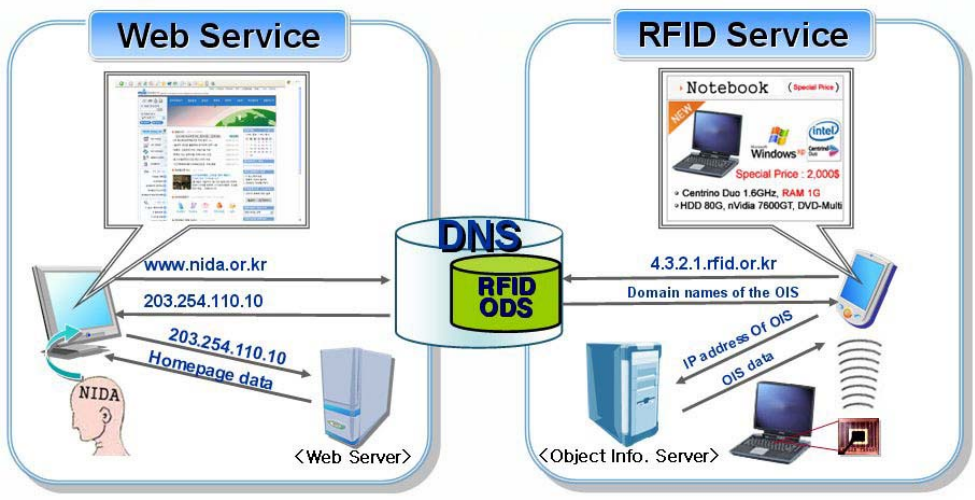
2.2.1. RFID 네트워크의 개요

RFID 네트워크는 전자태그간의 통신(센서 네트워크)으로부터 시작하여 태그에 할당된 RFID 코드로부터 실제 태그에 맵핑되는 정보를 검색하거나 태그가 부착된 객체의 변화를 등록하는 일련의 과정을 위한유무선 통신과 통신하는 주체 모두를 말한다.

RFID 네트워크는 RFID 검색시스템(National ODS, Local ODS)과 RFID코드와 관련된 정보를 저장하고 있는 객체정보서버, 그리고 이력정보가 저장된 위치정보를 가진 객체이력서버, 다수의 리더로부터 들어오는 정보를 수집 및 여과하는 미들웨어, RFID 태그로부터 RFID 코드 및 관련정보를 무선주파수로 수집하는 리더, 무선주파수를 이용하여 상품을 식별하기 위한 초소형 IC칩과 안테나가 내장된 태그 등으로 구성된다.

RFID의 전반적인 기술을 알아보기 위해 RFID 검색서비스를 중심으로 설명하겠다. [그림 2.1] “Web 서비스와 RFID 서비스의 비교”에서 보여주는 것과 같이 RFID 검색서비스는 ‘Web 서비스에서의 DNS’와 ‘RFID 서비스에서의 ODS’를 비교하면 이해가 쉬울 것이다. 두 서비스 모두 DNS 시스템을 기반으로 하며, 웹 서비스의 경우에는 웹서버의 도메인네임을 가지고 IP주소를 질의하고, RFID 서비스의 경우 RFID코드를 가지고 도메인네임을 질의하는 유사한 형태이다. 또한 RFID 서비스의 경우 도메인네임 획득 후 다시 DNS로의 질의를 통해 IP 주소를 얻어서 OIS로 실제 객체정보를 요청하게 된다.

Web 서비스에서 사용자가 원하는 홈페이지(NIDA 홈페이지)에 접근하기 위해서는 IP 주소대신 외우기 쉬운 도메인이름을 웹 브라우저의 주소창에 입력한다. 이때 도메인이름은 DNS를 거치면서 IP 주소로 바뀌게 되어 마침내 사용자는 원하는 홈페이지에 접속할 수 있게 된다. 이와 비슷하게 RFID 서비스에서는 사용자가 어떤 제품의 정보를 알고 싶을 때, RFID 전용 PDA 등을 이용하여 제품에 부착된 RFID 태그에서 RFID 코드를 읽어 온다. 읽어온 코드가 NIDA의 ODS를 거치면서 물체의 정보가 기록되어 있는 OIS의 위치를 얻게 된다. 그러면 사용자는 DNS를 거쳐 얻은 OIS의 IP 주소를 이용하여 OIS에서 물체의 정보를 얻을 수 있다.



[그림 2.1] Web 서비스와 RFID 서비스의 비교

2.2.2. RFID 네트워크의 구성요소

[그림 2.2]에서와 같이 RFID 네트워크는 크게 RFID Tag, RFID 리더, 미들웨어(Middleware), 객체검색서비스(ODS : Object Directory Service), 객체이력서비스(OTS : Object Traceability Service), 객체정보서비스(OIS : Object Information Service) 등으로 구성된다.[6)]

RFID Tag은 크게 데이터를 송수신하는 안테나와 데이터가 저장되어있는 칩이라는 두 부분으로 구성된다. 안테나는 태그에 전력을 공급하며 태그는 그 응답으로 데이터를 되돌려 주며, 자기장을 이용하는 방식과 전파를 이용하는 방식이 주로 이용된다.

자기장을 이용하는 방식은 일반적으로 30Mhz 이하의 주파수를 사용하고 (125Khz, 134Khz, 13.56Mhz) 인식거리가 짧은 (10cm 내외) 접촉식에 주로 이용되며 안테나에서 강한 고주파를 발생시켜 생성된 자기장이 TAG의 안테나 코일을 통과함으로써 생기는 전류에 의해 작동되는 원리를 이용한다.

6) RFID 네트워크 구성에 대한 자세한 사항은 http://www.ods.or.kr/ 의 자료실에 있는 'RFID 검색시스템 구축 및 운영지침서 V1.2' 를 참고하라

그에 비해 전파를 이용하는 방식은 레이더 기술과 비슷하게, 리더기에서 안테나에서 전파를 보내면, 태그에서 받아 송수신 파워로 사용하는 원리를 이용하여 주로 100MHZ이상의 주파수(900MHZ, 2.45GHZ)대역에 사용하며 전파 특성상 금속과 물에는 잘 흡수되는 성질이 있다.

또 다른 분류 기준으로는 전원 공급의 유무에 따라 전원을 필요로 하는 Active 형과 내부나 외부로부터 직접적인 전원의 공급 없이 리더기의 전자기장에 의해 작동되는 Passive 형으로 구분된다. Active 타입은 Reader기의 필요전력을 줄이고 리더와의 인식거리를 멀리 할 수 있는 장점이 있으나, 전원 공급 장치를 필요로 하기 때문에 작동시간의 제한을 받으며 Passive 형에 비해 고가인 단점이 있다. 반면, Passive 형은 Active형에 비해 매우 가볍고, 가격도 저렴하면서 반영구적으로 사용이 가능하지만, 인식거리가 짧고 리더에서 더 많은 전력을 소모한다는 단점이 있다.

참고로 본 문서에서 언급하는 시스템은 Passive 형 900Mhz 태그를 채용하였다.

RFID 리더는 태그의 정보를 읽어내기 위해 태그와 송수신하는 기기이며, 태그에서 수집된 정보를 미들웨어로 전송하는 기능을 한다. 태그에서 정보를 읽기위해서는 해당 태그에 맞는 주파수와 통신규격을 가지고 있는 리더기를 써야한다. 최근 나오는 리더들은 자체적으로 태그 필터링 기능이 내장되어 있기도 하다. 본 문서에서 언급하는 시스템에서는 ISO18000-6C 표준 Air-interface 규격을 가진 900Mhz 주파수를 사용하는 리더를 채용하였다.

미들웨어(Middleware)는 리더에서 계속적으로 발생하는 식별코드 데이터를 수집, 제어, 관리하는 기능을 하며, 모든 구성요소와 연결되어 계층적으로 조직화되고 분산된 구조의 미들웨어 네트워크를 구성하여 서로 통신한다. 미들웨어는 다양한 형태의 리더 인터페이스, 다양한 코드 및 망 연동, 여러가지 응용플랫폼에 대해서도 상호 운용성을 보장할 수 있어야 한다

ODS는 RFID 코드에 해당하는 OIS 및 OTS의 위치정보를 검색하는 시스템 으로 국가객체검색서비스(National ODS)와 로컬객체검색서비스(Local ODS)로 나뉘어진다. National ODS는 RFID 네트워크에서 DNS의 root DNS와 유사한 역할을 수행하며, 국가적 차원에서 관리되는 ODS이다. National ODS는 각 기관의 Local ODS의 위치 정보(IP Address)를 존(zone) 파일 내에 저장하고, RFID 코드 질의에 대해 Local ODS의 위치정보를 서비스 한다.

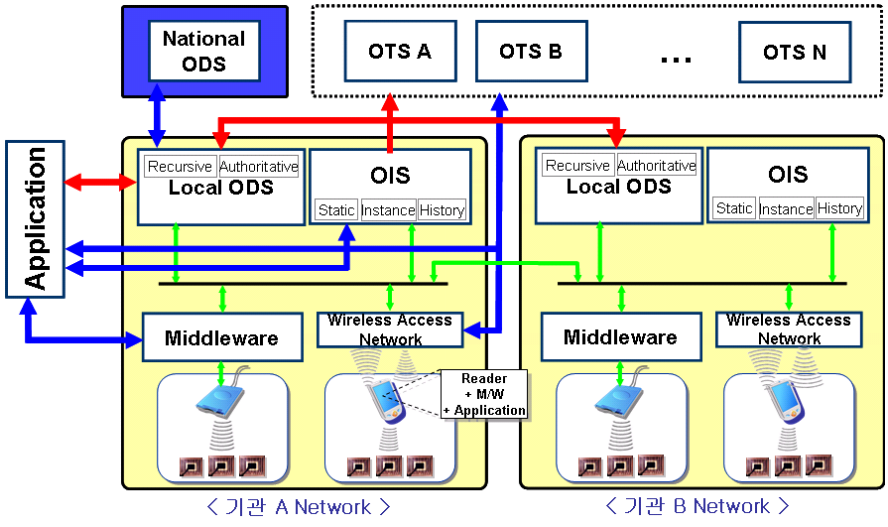
Local ODS는 DNS의 Local DNS 역할을 수행하며, 존 파일 내에 기관의 OTS 및 OIS의 위치정보(Service URI)를 저장하고, RFID 코드 질의에 대해 RFID 코드에 해당하는 OTS 및 OIS의 위치정보를 서비스한다.

National ODS는 국가 객체 검색 서비스로서, 각 기관의 Local ODS의 위치에 대한 정보 파일을 관리하고 이에 대한 서비스를 제공하는 역할을 한다. National ODS는 Local ODS의 위치 정보를 DNS 형태로 서비스한다. 이것은 BIND를 통해서 구현되는데, BIND의 검색을 위한 정보들이 저장되는 파일로는 마스터존(Master Zone) 파일과 슬래이브존(Slave Zone) 파일이 있다.

National ODS는 사용자별 접근 레벨을 두고 서비스를 제공하며, 동적 업데이트를 주기적으로 수행한다. 또한 등록/수정/삭제 관리 모듈을 두고 각 코드 정보에 해당하는 URL를 알맞게 처리할 수 있다.

객체정보서버(OIS)는 객체의 정적정보(Static) 및 동적정보(Instance, Historical)를 저장하고 서비스 한다. OIS는 각 기관 내부의 데이터를 저장하는 데이터베이스로써 다양한 형태로 저장된다. 이러한 다양한 형태의 저장된 데이터에 접근하기 위해서는 OIS 접근을 위한 인터페이스가 필요하다.

객체이력서비스(OTS)는 객체가 이동해간 객체정보서버(OIS)의 위치추적 정보를 서비스한다. 이러한 OTS는 각 기관별 혹은 저장되는 데이터의 용도에 따라 국가, 공공기관(물류, 유통, 국방, 등) 혹은 각 기관별로 관리가 가능하다.



[그림 2.2] RFID 네트워크 구성도

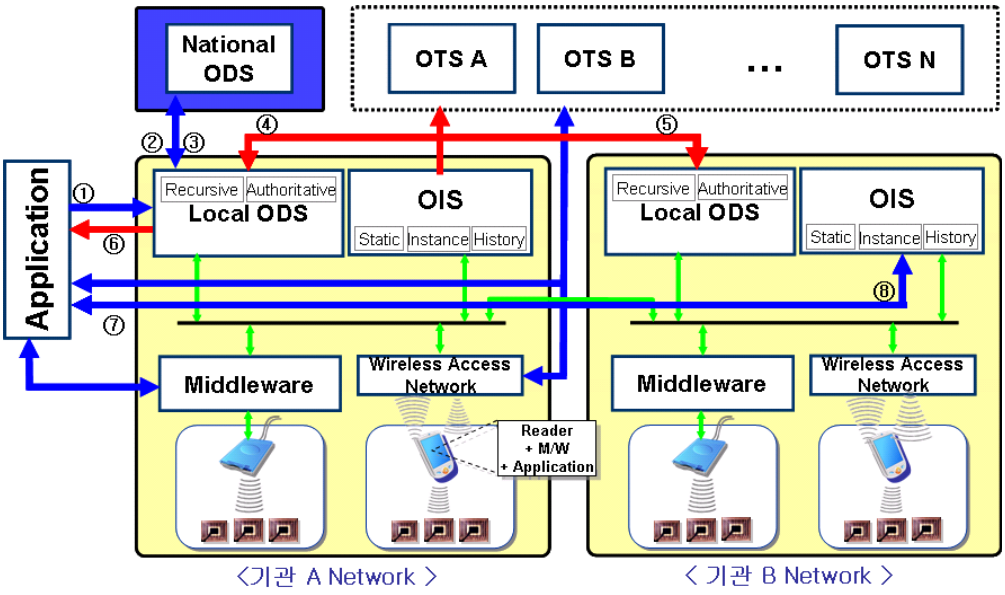
[표 2-1] 에 RFID 네트워크의 구성요소들에 대한 설명을 간략히 정리하였으니 참고하기 바란다.

구성요소	설명
RFID Tag	● 데이터를 송수신하는 안테나와 데이터가 저장되어있는 칩으로 구성 ● 안테나는 태그에 전력을 공급하며 태그는 그 응답으로 데이터를 되돌려 줌
RFID Reader	● 태그의 정보를 읽어내기 위해 태그와 송수신하는 기기 ● 태그에서 수집된 정보를 미들웨어로 전송하는 기능
Middleware	● 다양한 리더와 바코드 시스템이 ODS에 접속하기 위한 가교의 역할을 수행 ● 필요에 따라 데이터 필터링, 이벤트 처리와 메모리 관리 등의 역할담당
Local ODS	● 기관의 네트워크에 위치함 ● 각 기관별 자체적 관리 ● 해당기관에 속해있는 OIS 및 OTS의 위치 정보 존(Zone) 파일을 관리하고 이에 대한 서비스를 제공
National ODS	● 국가 객체 검색 서비스로서, 각 기관의 Local ODS의 위치에 대한 정보 파일을 관리하고 이에 대한 서비스를 제공
OTS	● 객체의 이력정보에 대한 검색 서비스를 제공 ● 이력정보를 가지고 있는 OIS 위치 목록을 저장 ● ODS가 이를 요청하는 경우에 해당 이력정보에 대한 OIS의 URL을 제공

[표 2-1] 시스템 주요구성 요소별 요약

2.2.3. RFID 네트워크 시나리오

2.2.3.1 객체정보검색 시나리오



[그림 2.3] 객체 정보 검색 과정

RFID 네트워크의 객체정보 검색 시나리오는 [그림 2.3]과 같다.

RFID 네트워크 상에서 태그의 객체정보를 알고 싶은 사용자로부터 리더기나 혹은 어플리케이션 상에서 코드값을 입력받아 객체정보를 알아내는 과정은 아래와 같다.

① 어플리케이션은 주어진 RFID 코드를 FQDN형식으로 질의 패킷을 생성하여 Local ODS로 질의한다. 이때 응용프로그램은 FQDN 형식을 필드별로 분리하여 fb.zzb.kkr.isoiec-15459.id.ods.or.kr 로 질의한다.

② Local ODS는 입력받은 FQDN 형식인 zzb.kkr.isoiec-15459.id.ods.or.kr 로 자신의 존(Zone) 파일 또는 캐쉬를 검색한 후 해당하는 정보가 있으면 어플리케이션으로 해당 OIS의 서비스 URIs 값을 NAPTRs RR 메시지로 돌려주고 없을 경우 기관코드 부분까지를 남긴 zzb.kkr.isoiec-15459.id.ods.or.kr 를 National ODS로 질의한다.

③ 기관 A 네트워크에서 National ODS로 들어온 질의는 국가 존 파일을 검색하여 객체가 처음 등록되어 객체정보가 있는 기관 B 네트워크의 Local ODS의 서비스 URIs를 응답한다.

④ 기관 A 네트워크의 Local ODS는 응답받은 기관 B 네트워크의 Local ODS의 URI로 OIS의 위치 정보를 얻기 위해 질의(DNS query)한다.

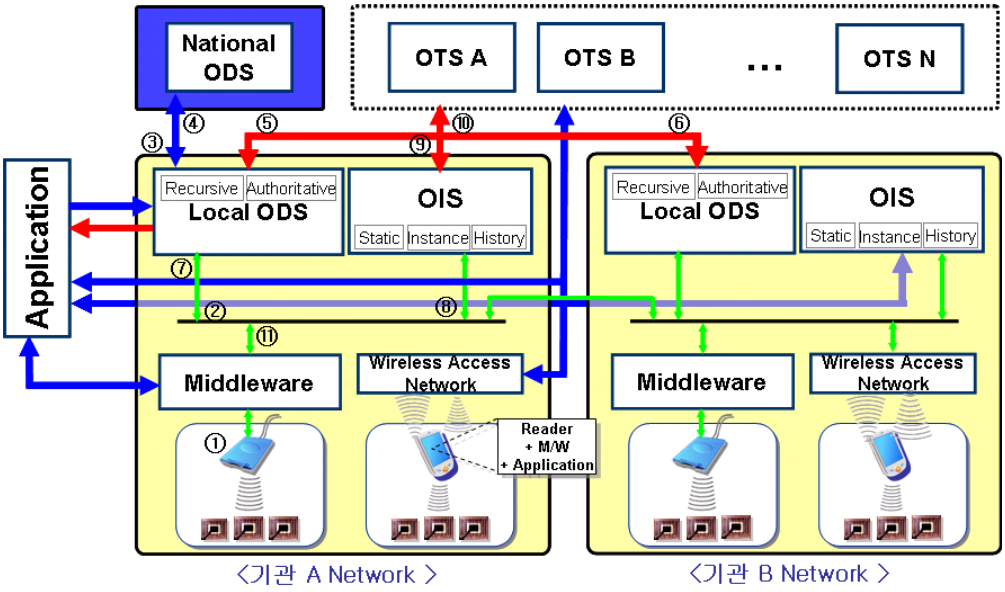
⑤ 기관 B 네트워크의 Local ODS는 자신의 존 파일을 검색하여 질의한 코드에 대한 OIS의 NAPTRs RR (서비스 URI)를 기관 A Local ODS에 응답한다.

⑥ 기관 A 네트워크의 Local ODS는 기관 B 네트워크의 Local ODS로부터 받은 NAPTRs RR (서비스 URI)을 질의한 응용프로그램에 넘겨준다.

⑦ 응용프로그램은 NAPTRs RR로부터 서비스 URI를 파싱하여 기관 B 네트워크의 OIS에 객체정보를 질의한다.

⑧ 기관 B 네트워크의 OIS는 질의한 응용프로그램으로 객체정보를 제공한다.

2.2.3.2 객체이력정보등록 시나리오



[그림 2.4] 객체 이력정보 등록 과정

기관 B 네트워크에 있던 태그가 새로운 기관 A 네트워크로 들어와서 이력정보를 쓰고 OTS에 등록하는 과정은 [그림 2.4]와 같다.

① 새로운 기관 A 네트워크로 이동한 태그의 코드정보를 리더가 읽어 들이고, 리더는 미들웨어로 태그로부터 읽어 들인 코드 값과 이력정보(리더ID, 타임스탬프) 등을 보낸다.

② 미들웨어는 RFID 코드에 대한 OTS의 위치정보(IP Address)를 Local ODS로 질의(DNS query)하면, Local ODS는 자신의 존 파일 내에 질의한 RFID 코드가 있는지를 검색하고 코드에 대한 OTS의 위치정보가 없음을 확인한다.

③ National ODS로 최초 태그가 생산되어서 OTS가 할당된 기관 B 네트워크의 Local ODS로 OTS의 위치정보를 질의한다. 이때 질의는 DNS의 root DNS 질의 과정과 동일하다.

④ National ODS는 기관 A 네트워크의 Local ODS로 기관 B 네트워크의 Local ODS의 URI(IP Address)를 응답한다.

⑤ 기관 A 네트워크의 Local ODS는 기관 B 네트워크의 Local ODS로 RFID 코드에 해당하는 서비스 URIs(NAPTR)를 질의(DNS query)한다.

⑥ 기관 B 네트워크의 Local ODS는 자신의 존 파일내의 코드에 해당하는 NAPTR RR을 기관 A 네트워크의 Local ODS로 응답(DNS response)한다.

⑦ 기관 A네트워크의 Local ODS는 미들웨어로 OTS의 위치정보가 들어있는 NAPTR RR을 응답(DNS response)한다.

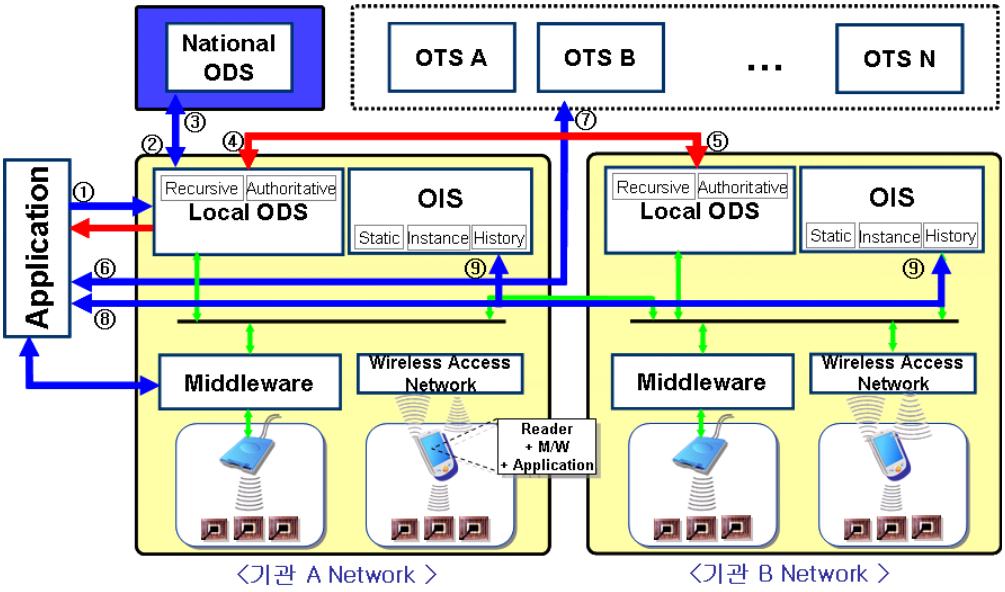
⑧ 미들웨어는 입력받은 코드 값과 이력정보를 필터링하여 실제 저장될 정보들을 OIS에 기록 요청한다. 객체검색서버로부터 받은 NAPTR RR을 파싱하여 OTS의 URI(IP Address) 정보도 OIS에 제공한다.

⑨ 기관 A 네트워크의 OIS는 이동한 객체의 이력정보(리더ID, 타임스탬프)를 기록 후 기록된 이력정보의 서비스 URI의 등록을 OTS에 요청한다.

⑩ OTS는 태그 생성시 할당되어진 필드에 기관 A 네트워크의 OIS의 이력정보 서비스 URI를 추가 후 기관 A 네트워크의 OIS로 등록이 완료되었음을 알린다.

⑪ 기관 A 네트워크의 OIS는 미들웨어로 OTS와 OIS의 등록이 완료되었음을 알린다.

2.2.4.3 객체이력정보검색 시나리오



[그림 2.5] 객체 이력정보 검색 과정

태그의 이력정보를 알고 싶은 사용자로부터 리더나 혹은 어플리케이션 상에서 코드 값을 입력받아 이력정보를 알아내는 과정은 [그림 2.5]와 같다. (기관 네트워크 B로부터 기관 네트워크 A로 이동한 객체에 대한 이력정보 검색과정)

① 기관 네트워크 A의 Local ODS로 객체의 OTS의 URI를 질의한다.

② Local ODS는 National ODS로 최초 등록되었던 기관 네트워크 B의 Local ODS의 URI를 질의한다.

③ National ODS는 기관 네트워크 B의 Local ODS의 URI를 제공한다.

④ 기관 네트워크 A의 Local ODS는 기관 네트워크 B 의 Local ODS로 OTS의 URI를 질의한다.

⑤ RFID 코드에 해당하는 OTS의 URI를 응답한다.

⑥ 코드를 OTS에 질의한다.

⑦ OTS는 코드에 맵핑되는 이력정보가 있는 OIS의 URIs를 응용프로그램으로 보내준다.

⑧ 응용프로그램은 이력정보 OIS의 URIs로 이력정보 질의한다.

⑨ 각 이력정보 OIS는 코드에 해당하는 이력정보를 응용프로그램에 제공한다.

제3장 코드체계 적용

3.1 kCode 개요

RFID 서비스 네트워크는 RFID 코드체계, 태그, 리더, 미들웨어, 검색시스템, 객체정보시스템, 객체이력시스템, 관련 어플리케이션 등으로 구성된다. 이러한 구성 요소 중에서 RFID 코드는 태그에 기록되어 객체를 구별하는 식별자로서 인터넷상에서 인터넷주소체계로 변환되어 객체정보 접근 매개체로 이용된다. 특히 RFID 코드는 RFID 서비스 네트워크에서 최초로 취급되는 정보라는 점에서 매우 중요한 요소라고 할 수 있다.

현재 EPC, ISO/IEC, uID 센터 등 다양한 기관에서 RFID 코드체계를 규정하여 활용하고 있으며, 국방/의료/모바일용 등 특수목적과 시험용을 위한 코드체계의 도입 논의가 국내·외적으로 진행되고 있다. 또한, 이러한 RFID 코드체계를 전달하기 위한 RFID의 Air Interface 도 표준화가 되어가고 있다.

이처럼 RFID 코드체계의 다양성은 코드체계 선택에 따라 다양한 RFID 네트워크가 설계되며 서비스 또한 이에 맞춰 진행될 것으로 판단된다. 이러한 RFID 서비스 네트워크의 상호 운용성을 위하여 NIDA는 RFID 검색시스템을 구축·운영하고 국내 RFID 서비스 네트워크의 연동을 위하여 관련 표준화·시스템·정책 측면에서 많은 노력을 기울여 왔다.

그러나 RFID 코드체계를 RFID 태그 메모리에 기록하여 활용하기 위한 인코딩에 대한 정보의 부재로 지난 2004년 및 '05년도 RFID 관련 사업 추진이 코드 인코딩에 대한 구체적인 가이드라인의 부재로 표준 RFID 코드 인코딩 규칙을 따르지 않아 향후 RFID 서비스 네트워크 간의 데이터 교환에 문제를 가져올 것으로 예상되는 등 RFID 사업자에게 큰 혼란을 가져왔다.

이에 NIDA에서는 ISO/IEC 15459 국제표준을 준수하는 kCode 체계를 마련하였으며, 2006년 7월 KS 입안예고 된 상태이다. 본 구현은 이러한 kCode 체계를 적용하여 구현되었다. kCode 체계 관련 세부내용은 "RFID 코드 인코딩 지침서 V1.0" 참고하면 되고, 검색서비스 홈페이지 www.ods.or.kr 에서 책자를 다운로드 가능하다.

3.2 kCode 체계

3.2.1. 코드체계 구현

3.2.1.1. 태그 코드 체계

이번 구현에 있어서 코드 체계는 96bit ISO/IEC 18000-6C (EPCglobal Class1 Generation2) Tag를 인코딩 하는데 사용된다. NIDA에서 지정한 kCode 체계를 준용하여 이번 사업에 적용된 태그 데이터 코드 체계는 [표 3-1]와 같다.[7)]

구분	Encoding Header	IAC	CC	Prefix	IC	SC	총계
비트수 (5bit인코딩)	24	15	15	5	10	25	94
문자수	-	3	3	1	2	5	15
표현가능 문자 수	-	-	17,576	26	676	11,881,376	-

[표 3-1] 태그 데이터 코드 체계 적용안

※ 26의 지수 정보 : 26^2=676 / 26^3=17,576 / 26^4=456,976 / 26^5=11,881,376

쓰여질 정보는 총 94비트이며 남는 2비트는 "00" 으로 덧댐한다.

NIDA에서 관리되고 있는 자산 번호는 숫자로 되어있고 총 10자로 구성되어 있다. 그러나 [표3-1]에서 볼 수 있는 것처럼 IC 부분과 SC 부분을 다 합쳐도 7문자밖에 사용할 수 없으므로 자산 번호를 축약 또는 매핑하여 크기를 줄일 필요가 있다. 다음은 이번 시스템에서 사용된 자산번호와 RFID코드간의 매핑테이블의 일부이다.

7) 이번에 구축된 시스템에 적용된 코드체계는 자산을 대상으로 하고 있으므로 ISO/IEC 15459-4 kCode 체계를 적용하였다.

대분류			소분류		일련번호
명칭-1	명칭-2	코드	명칭-3	코드	ooooo
토지	-	10 - A	-	000 - A	ooooo
건물	-	11 - B	-	000 - A	ooooo
공통장비	사무시설	21 - C		000 - A	ooooo
	기자재	22 - D	네트워크장비	010 - B	ooooo
			프린터	020 - C	ooooo
			팩시밀리	030 - D	ooooo
			사업용 휴대폰	040 - E	ooooo
			사업용 S/W	050 - F	ooooo
	기타	29 - E	-	999 - Z	ooooo

[표 3-2] NIDA 본사의 자산번호-RFID코드 매핑테이블의 일부분

대분류의 경우 총 17여개, 소분류의 경우 11개의 코드 분류가 존재하므로 각 분류당 1문자씩만 할당 되어도 NIDA에 있는 모든 자산의 대분류, 소분류를 모두 표현할 수 있다. 일련번호의 경우 0~9 를 A-J로 매핑시켜서 사용한다. 따라서 자산 중 대분류가 '기자재'이고 소분류가 '프린터'로 되어있는 자산번호가 2203000123 인 자산의 IC는 DB가 되고 일련번호는 AABCD 가 된다. 이를 토대로 대한민국(KKR)에 있는 NIDA 본사(ZZA)의 자산번호 2201000124번을 가진 자신의 kCode 는 다음과 같다.

코드타입	발급자코드	기관코드	Prefix	Item Code	Serial Code
ISO/IEC 15459-4	KKR	ZZA	B	DB	AABCD
	국가 : 대한민국	기관 : NIDA 본사	IC 길이 : 2	대분류+소분류	일련번호

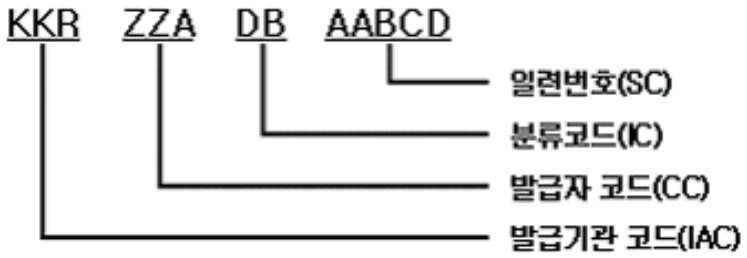
[표 3-3] ISO/IEC 15459-4의 kCode 예제 분석

위에서 설명한 내용을 바탕으로 코드 체계를 준용했을 경우 분류코드 2자리, 일련번호 5자리로 구성되며, [그림 3.1]의 예시와 같은 구조를 갖는다.

[필드설명]

① 발급기관 코드 - (IAC)

- 발급기관코드는 ISO/IEC 15459-2에서 'National Public Administration'을 위하여 첫 글자를 'K'로, 이후 두 글자는 ISO 3166에 정의된 국가코드 두문자(대한민국 : KR)가 오도록 정의하여 총 3문자의 **'KKR' 사용**하도록 한다. 하위영역은 IAC를 발급받은 기관인 Issuing Agency에 의해서 정의된다.



[그림 3.1] KS X ISO/IEC 15459를 준용한 코드 예시

② 발급자 코드 - (CC)

- Tag 코드를 발행하는 기관을 의미함.
- 발급기관에서 승인한 발급자 코드. 본 시스템에서는 본사 및 지사 각각에 발급자 코드를 할당하였음. 본사는 ZZA, 지사는 ZZB 로 고정

③ 구분자 - (Prefix)

- IC의 길이를 나타냄.
- IC의 길이에 따라 A-Z 사이의 값을 가지며 자세한 사항은 [표3-4] 참고.

④ 분류코드 - (IC)

- 해당 자산이 등록되어 있는 자산의 대/소분류 항목을 나타냄.
- 발급자로부터 부여받은 Tag를 인코딩하여 객체에 부착하는 주체가 됨.
- 2글자의 대문자 알파벳으로 676(26^2)개의 부서 및 분류코드 부여 가능.

구분	대분류	소분류
문자수	1	1
자산/부외자산	A～N/X,Y,Z	A～K
출입자	O, P, Q	

[표 3-4] 분류코드 구분

ⓐ 대분류 : 1자리 5bit. 자산의 1차 분류 기준 및 출입자의 고용형태에 따라 자산의 경우 "A~N" 까지 부여하고 출입자에겐 O, P, Q로 나눠서 부여하며 R~Z 까지는 차후 확장을 위해 예약한다.

ⓑ 소분류 : 1자리 5bit. 자산의 경우 2차 분류 기준을 의미하며 출입자의 경우 NIDA 정직원, NIDA 외주직원, 및 위탁업체 코드를 의미한다. 기본적으로 A~K 까지 지정되어 있으며 L~Z 까지는 차후 확장을 위해 예약한다.[8)]

⑤ 일련번호 - (SC)[9)]

- 각 자산에 부여되는 식별고유번호
- 5글자의 알파벳으로 이루어짐. 업체 및 기관별로 11,881,376(26^5)가지의 식별 고유번호 부여 가능

8) 원래 NIDA 자산 관리에서 쓰던 대분류 코드는 2자리, 소분류 코드는 3자리로 지정되어 있지만 실제로 쓰는 코드는 각 20개 정도로서 각각 1문자로 표현이 가능하다. [표3-2]을 참고하라.

9) 앞에서도 언급했다시피 일련번호의 각 자리수는 0~9까지 자연수로 이루어져있지만 5Bit 인코딩을 사용했기 때문에 숫자를 쓸 수 없어 A~J 로 매핑 시켜서 구현하였다.

3.2.1.2 인코딩

- ISO/IEC 646문자를 압축 및 부호화하기 위해 5bit 압축 인코딩 방식[10]을 적용

- 5비트 압축을 위해서는 모든 입력 바이트 41_H에서 $5F_H$ 범위내에 있어야 하며, 문자열은 4바이트 이상의 길이여야 한다.

- 5비트 압축의 규칙은

[1단계] 각 필드에 대한 코드 값을 선택한다.

[2단계] 각각의 문자에 대하여
a. 바이트 값의 범위가 41_H에서 $5F_H$까지의 범위내에 있는지 확인한다.
b. 2 바이트 값을 8비트의 2진수로 변환한다.
선행 3비트를 제거한다.
남은 5비트를 비트스트링에 연결한다.

[3단계] 생성된 비트 스트링을 8비트의 세그먼트로 분할한다. 비트 스트링이 8로 나누어 떨어지지 않을 경우에는 빈자리 만큼 "0" 을 넣어서 채운다.

[4단계] 8비트 세그먼트를 Tag UID 값으로 변환한다.

10) ISO/IEC 15962(KS X ISO/IEC 15962:2005)

● 태그 인코딩의 예 :

NIDA 본사 사무실에서 식별번호를 AABCD(00123) 쓰는 레이저 프린터에 대한 태그(96Bit)를 발행하려고 한다.

[0단계] 코드 정보를 태그에 인코딩 하기 앞서, 해당 태그가 어떤 방식으로 기록되어 있는지 정보를 담고 있는 인코딩 헤더 부분의 정보를 먼저 기입해야 한다. 제3장 3절에서 언급했다시피 이 부분은 DSFID, Precursor 정보 및 Object Length 로 이루어져 있으며 각 항목당 8Bit씩 할당되어 있다. 제3장 3절 2번에 예시를 참고하여 인코딩 헤더 정보를 작성한다.

[1단계] 각 필드에 대한 코드 값을 선택한다.

[1-1] 발급기관 코드와 발급자 코드를 선택한다.
발급기관 코드는 KKR로 고정된다.
발급자 코드는 NIDA로 발급된 본사 사무실을 의미하는 ZZA를 쓴다.

[1-2] 분류코드(IC)와 그에 따른 구분자를 선택한다.
NIDA 내부기준에 의해 대분류로 “기자재”는 "D" 란 코드를 부여받고, 레이저 프린터는 "프린터“ 로 분류되어 소분류 코드는 ”B" 를 부여받는다.
구분자의 경우 IC문자가 총 2문자이므로 “B"가 된다.[11)]

[1-3] 일련번호를 선택한다. (00127 을 문자로 매핑하므로 AABCD가 된다)

이상의 1단계가 끝나면 코드 구성표는 다음과 같다.

Encoding Header	IAC	CC	Prefix	IC	SC
생략	KKR	ZZA	B	DB	AABCD

[2단계] 각각의 문자의 아스키 값이 41_H에서 $5F_H$까지의
범위내에 있는지 확인.
[3단계] 객체의 내용을 Tag UID로 변환하고, 다시 2진수로 변환,
bit 8과 7 및 6을 제거 (5Bit 인코딩)

11) Prefix 는 A~Z 까지 정의되며 IC의 길이와 관계가 있다. 즉 IC길이가 1이면 A, 2이면 B, 3이면 C 이런식으로 매칭된다. 자세한 사항은 [표3-4] 를 참고하라.

객체의 내용	Tag UID	2진수
K	4B	01001011
K	4B	01001011
R	52	01010010
Z	5A	01011010
Z	5A	01011010
A	41	01000001
B	42	01000010
D	44	01000100
B	42	01000010
A	41	01000001
A	41	01000001
B	42	01000010
C	43	01000011
D	44	01000100

bit 6,7,8을 제거
01011
01011
10010
11010
11010
00001
00010
00100
00010
00001
00001
00010
00011
00100

[3, 4단계] 0단계에서 작성된 인코딩 헤더와 합쳐
8비트 세그먼트로 2진수 문자열을 그룹화한 후 "00" 비트를 덧댐

00000101 00110100 00001001 01011010 11100101 10101101
00000100 01000100 00010000 01000010 00100001 10010000

[5단계] Tag UID로 변환

05 34 09 5A E5 AD 08 46 20 42 11 04

3.2.1.3. 태그 데이터 구조와 활용안

본 규격에서 정의하는 kCode를 사용하는 경우 ISO/IEC 18000-6 Type C 태그의 메모리 구조는 [표 3-5]와 같다[12].

Bank	Bit No.	Field Name			bit	내 용
00		Reserved			64	-
01	16	EPC 영역	CRC-16		16	error check code
	17~21		PC	Length	5	encoding 체계표현 "001102"(610)
	22~23			RFU[13]	2	향후사용위해 "0"고정
	24			Toggle	1	"12"(ISO 15961, AFI사용)
	25~32			AFI	8	AFI Code
	33~128		EPC Tag Encoding		96	96bit의 UII
	-		Word Boundary Filler		0	-
10	-	TID			(32)	Tag 식별자
11	-	User			(288)	사용자 정의 데이터

[표 3-5] kCode를 위한 ISO/IEC 18000-6 Type C 태그 설정

- KS X ISO/IEC 15459(ISO/IEC 15459) 코드 체계를 사용하는 경우에는 PC Field의 8번째 bit(Toggle bit)에 "1_2"이 설정되고, 다른 RFID 시스템과의 충돌을 방지하기 위해 응용구분자인 ISO 15961 AFI를 9번째 bit(AFI bit)에 할당받은 AFI Code를 사용한다.

- KS X ISO/IEC 15459 규격으로 인코딩된 태그를 인식하기 위해서는 PC Field의 8번째 bit(Toggle bit)의 "1_2" 값을 코드 인식 모듈 또는 응용 S/W로 전달해 주어야 한다.

12) PC , CRC-16 영역은 2006년 10월 현재 이 두 영역을 읽고 쓸 수 있는 RFID 리더기가 시장에 나와 있지 않아 본 사업에서는 구현을 보류하였다. 즉 사용자는 현재 태그에 Bank 01:EPC Tag Encoding 영역만 기록할 수 있다.

13) RFU : Reserved for Future Use

- AFI(Application family identifier)는 태그가 반응해야하는 특정 어플리케이션을 지정한다. 복수의 태그 중에서 특정 어플리케이션이 요구하는 특정 태그를 추출하는데 사용된다.

AFI Code (10진수)	ASF Code (10진수)	SystemInfo Encodation (16진수)	Assigned to:
9	1	91	EAN.UCC system (to be defined before publication)
9	2	92	EAN.UCC system (to be defined before publication)
9	3	93	EAN.UCC system (to be defined before publication)
9	4	94	EAN.UCC system (to be defined before publication)
9	5	95	EAN.UCC system (to be defined before publication)
9	6	96	EAN.UCC system (to be defined before publication)
9	7	97	EAN.UCC system (to be defined before publication)
10	1	A1	ANS MH10.8.2 Data Identifiers for unique identifier for items
10	2	A2	ANS MH10.8.2 Data Identifiers for unique identifier for transport units
10	3	A3	ANS MH10.8.2 Data Identifiers for unique identifier for returnable transport items
11	1	B1	ISO/IEC 15459 unique identifier for items
11	2	B2	ISO/IEC 15459 unique identifier for transport units
11	3	B3	ISO/IEC 15459 unique identifier for returnable transport items(Pallet)
12	1	C1	IATA Baggage Tag

[표 3-6] Code Assignments for ApplicationFamilyId
(ISO/IEC 15961. AnnexB)

- AFI Code 결정 예 :

KS X ISO/IEC 15459 규격으로 NIDA 지사 사무실에서 식별번호 AAABA인 LCD 모니터에 개별적으로 태그를 붙이기 위해 태그(96bit)를 발행하려고 한다.

[1단계] Matching 테이블에서 Code값 찾기

Code Assignments for ApplicationFamilyId(ISO/IEC 15961. AnnexB) [표3-6]를 참조하여 해당하는 AFI, ASF, System-Info-Encodation의 Code 값을 찾는다.

이번 예에서 KS X ISO/IEC 15459 규격을 준용하는 품목(item) 태그이므로 AFI Code 11, ASF Code 3 즉, 16진수 B3에 해당된다.

▶ [표3-6] 음영부분에 해당됨

[2단계] 16진수 B3을 2진수로 변환

$B3_H = 10110010_2$

* 응용예제 : Pallet 1개에 LCD 모니터가 20개 적재되어있는 상태에서 LCD 모니터 박스 전체에 대한 불출처리를 가정하자. 불출처리를 위해 담당직원은 Pallet의 Tag만 인식 후 20개의 LCD 모니터에 대한 일괄처리를 하기 원할 것이다. 이때, 리더는 태그의 AFI Code를 인식하여 Pallet 태그만을 인식할 수 있다.

- Bank 10의 TID 영역에는 Tag를 유일하게 식별해주는 TID Code가 기록된다.

- Bank 11의 User Memory 영역은 사용자 정의 데이터가 기록된다. 이번 구현에서는 User Memory를 활용하지 않는다.

3.3 kCode 적용 예제

3.3.1. 개요

ISO/IEC 15459 표준문서에서는 RFID 리더쪽에서 Tag 내에 인코딩 된 16진수로 이루어진 Tag UID 값을 인코딩 헤더 정보에 따라 EPC, 혹은 ISO/IEC 표준 코드로 디코딩하여 미들웨어 혹은 응용 프로그램에 전달하는 것을 원칙으로 하고 있다.

그러나 현재 ISO/IEC 18000-6C(Gen 2) 규약의 RFID 리더를 만드는 제조업체들은 대부분 최소한의 승인 규약만 만족시킨 채 표준코드 디코딩 기능을 구현하지 않은채로 RFID Gen 2 규격 리더를 출시하고 있는 형편이다. 이렇게 최소한의 표준 규약으로 만든 리더는 ISO/IEC 15459 표준으로 제정된 태그 코드 체계를 해석하는 기능이 없으며 이 경우 코드체계 해석을 미들웨어에서 처리해야 한다.
이번 구현에서 쓰인 고정형 리더 역시 코드 디코딩 기능이 구현되어 있지 않고 단순히 16진수로 이루어진 Tag UID값만 미들웨어로 넘겨주는 형태로 되어있다.

본 구현에서는 kCode 디코딩 모듈을 미들웨어에 적재하여 처리하였으며 개발툴로 가장 많이 쓰이고 있는 MicroSoft 사의 C# .Net를 이용하여 구현하였다.[14]

3.3.2. 코드 인코딩

태그를 ISO/IEC 15459 표준에 맞게 인코딩 하려면 해당 Tag UID값을 코드체계로 변환하는데 필요한 인코딩 헤더 정보가 먼저 기록되어야 한다. 크기는 24비트이며 DSFID 부분, Precursor 부분, Object Length 부분 이렇게 세 부분으로 이루어져있으며 각 부분은 1바이트의 크기를 가지고 있다.
헤더 정보 후에는 코드에 관한 정보가 헤더에 기술된 방식에 의해 인코딩 된다.

14) kCode 인코딩/디코딩 개발툴은 MS사에서 출시한 Visual Studio .Net 2003 Ent. 버전을 사용하였다.

아래의 예제는 ISO/IEC 15459 표준 코드체계인 "KKR.ZZA.B.CD.EFGHI" 코드를 어떻게 태그에 인코딩 하는지 보여준다.

① DSFID 계산

구분	Access Method		Reserved	Data Format				
bit열	b8	b7	b6	b5	b4	b3	b2	b1
예시	0	0	0	0	0	1	0	1
의미	non-directory		예약	ISO/IEC 15459의 root-OID 사용				

[표 3-7] DSFID 세부구조

- b8 ~ b7 : Access Method가 non-directory[15] 방식임을 나타내는 '00_2'으로 정의
- b6 : 향후 사용을 위해 예약되어 Default 값인 '0_2'으로 설정
- b5 ~ b1 : KKR 코드는 ISO/IEC 15459의 root-OID {1 0 15459}를 나타내는 Data Format '0 0101_2'로 할당

위의 조건들을 종합해볼때 DSFID 값은 String 형태의 값으로 다음과 같이 할당될 수 있다.

```
//b8~~b7 - Access Method 가 넌디렉토리방식 : 00
//b6 : RFU - 0
//b5~b1 15459 의 Root-OID를 의미하는 0 0101
string _DSFID = "00000101";
```

15) non-directory : Precursor와 ObjectID 및 Object가 반복되어 사용되는 논리적 메모리 구조

② Precursor 계산

구분	offset & expansion	Compaction Type Code			Relative-OID			
bit열	b8	b7	b6	b5	b4	b3	b2	b1
예시 (15459-1)	0	0	1	1	0	0	0	1
예시 (15459-4)					0	1	0	0
의미	한개의 Object만 기록	5bit 압축			ISO/IEC 15459의 OID {1 0 15459} 하위 arc 1과 4의 값			

[표 3-8] Precusor의 세부구조

- b8 : KKR 코드는 한 개의 코드체계만 Bank 01에 기록하므로 '0_2'으로 정의됨
- b7 ~ b5 : 이번 구현은 ISO/IEC 646의 7bit 문자중 영문 대문자 만을 사용하므로 5bit 코드 압축을 의미하는 '011_2' 를 사용한다. 영문 대문자와 숫자만을 사용하므로 6bit 코드 압축을 의미하는 '100_2'을 사용
- b4 ~ b1 : ISO/IEC 15459-1 및 15459-4의 root-OID를 제외한 하위 arc는 한개만 존재하며, 그 하위 arc '1' 및 '4'는 1에서 14의 범위에 있으므로 Relative-OID 영역에 각각 '0001_2' 및 '0100_2'으로 기록

위의 조건들을 종합해볼때 Precursor 값은 String 형태의 값으로 다음과 같이 할당 될 수 있다.

```
//b8 : KKR 코드는 한개의 코드체계만 Bank 01에 기록함 : 0
//b7~b5 : 압축방식 - 5비트 이므로 : 011
//b4-b1 - 이번 구현은 15459-4 이니까 : 0100
string _Precursor = "00110100";
```

③ Object 계산

구분	Object	
	length of Object	Object
KKR 코드	bbbb $bbbb_2$	-

[표 3-9] Object의 세부구조

- length of Object : Object Byte 수
  · Object가 0에서 127 Byte이면, 1byte의 '0bbb $bbbb_2$'로 표시
    ※ 'bbb $bbbb_2$'는 Object의 Byte 수를 2진수로 표현한 값이며, 본 지침서에서는 127 Byte 이하의 코드체계만 다르므로 0에서 127 Byte Object 길이 계산법만 다루므로 세부적인 '내용은 ISO/IEC 15962'를 참조
- Object : Object 값, 즉 실제 RFID 코드체계가 들어가는 부분으로 Compaction Code Type에 따라 값이 변함

위의 조건들을 종합해볼때 Object Length 값은 String 형태의 값으로 다음과 같이 할당 될 수 있다.

```
// 인코딩된 데이터 - 총 72비트,
//즉 9바이트가 Object Length 가 된다.
// _ObjectLength = 9 즉 00001001 이 됨
string _ObjLength = Convert.ToString(9,2).PadLeft(8,'0');
// 9를 이진수로 변환한 후 8자리수를 채움.
```

여기까지 할당된 세 필드의 값을 모두 합치면 인코딩 헤더가 된다.

```
string resultBitCode = "";
resultBitCode += _DSFID + _Precursor + _ObjLength;
// 인코딩 정보 부분 기입 완료.
```

인코딩 헤더 기록이 끝났으면 코드값을 기록해야한다. 여기서는 ISO/IEC 646 표준 5Bit 인코딩 방법을 사용하며 인코딩 방법은 본 문서 **3.2.1.2 인코딩** 단락에 기술되어 있으므로 참고한다. 본 구현에서는 자체제작한 StrToBit()[16] 라는 5Bit 대문자 변환 함수를 사용하였다.

```
resultBitCode += StrToBit(uNCode); //  “KKR" 국가코드
resultBitCode += StrToBit(ulCode); // "ZZA" 기관코드
resultBitCode += StrToBit(_Prefix);// "B" PREFIX
resultBitCode += StrToBit(_IC);      // IC 부분 "CD"
resultBitCode += StrToBit(_Serial);// Serial "EFGHI"
resultBitCode += "00";   // 덧댐 (Padding)
```

상위의 PREFIX 정보는 IC의 길이와 연관되며 IC의 길이가 1문자이면 "A", 2문자이면 "B" 이런식으로 최대 26문자까지 기록할 수 있다. 본 예제인 "KKR.ZZA.B.CD.EFGHI" 코드는 IC부분인 "CD" 부분이 두 글자이기 때문에 기관코드 다음에 나오는 Prefix 값이 "B"로 결정되었으며 IC 부분의 길이가 변경될 경우 Prefix 부분도 값이 바뀌게 된다.

지금까지 기록된 정보의 길이는 인코딩헤더 24비트, 국가코드 15비트(5Bit * 3문자), 기관코드 15비트, PREFIX 부분 5비트, IC부분이 10비트, Serial 부분이 25비트로 **총 94비트**이다. 통상적으로 가장 많이 쓰이는 ISO/IEC 18000-6C 태그는 96비트의 UID 메모리를 가지고 있으므로 나머지 두 비트는 “00” 으로 덧댐한다.[17]

이렇게 만들어진 Encoded Bit 문자열을 Tag UID 값으로 변환이 필요하다.
이러한 변환은 다음처럼 제공되는 유틸리티 클래스 함수의 호출을 통해 쉽게 할 수 있다.

16) StrToBit(string Chars) : 문자열을 넘겨받아 5Bit 알파벳으로 변환후 비트 문자열로 변환된 값을 반환해주는 함수

17) 덧댐 : Padding, 모자란 자리수를 채우기 위해 다른 대체문자를 덧붙이는 것.

```
public string BitToHex(string resultBitCode)
{
   char[] arr;
   string tmpStr = "";
   int tmpInt = 0;
   string resultStr = "";
   for(int i=0;i<96/4;i++)
   {
      arr=resultBitCode.ToCharArray(i*4,4);
      tmpStr = arr[0].ToString()
      + arr[1].ToString()
      + arr[2].ToString() + arr[3].ToString();
      tmpInt = Convert.ToInt32(tmpStr,2);
      resultStr += Convert.ToString(tmpInt,16);
   }
   return resultStr;
}
```

이렇게 완성된 소스코드의 전문은 아래와 같다.

```
using System;

namespace ODSExample
{
   class ODSTestApp
   {
      [STAThread]
      static void Main(string[] args)
      {
         string uNCode = "KKR";
         string uICode = "ZZA";
         string _IC = "CD";
         string _Serial ="EFGHI";

         EncUtil util = new EncUtil();

         // _Prefix 부분은 _IC의 길이에 따라 결정된다.
         // _IC 길이가 1이면 A, _IC 길이가 2이면 B ....
         string _Prefix;
```

```
            _Prefix = Convert.ToChar(_IC.Length + 64).ToString();

            //b8~~b7 - Access Method 가 넌디렉토리방식 : 00
            //b6 : RFU - 0
            //b5~b1 15459 의 Root-OID를 의미하는 0 0101
            string _DSFID = "00000101";

            //b8 : KKR 코드는 한개의 코드체계만 Bank 01에 기록함 : 0
            //b7~b5 : 압축방식 - 5비트 이므로 : 011
            //b4-b1 - 이번 구현은 15459-4 이니까 : 0100
            string _Precursor = "00110100";

            // 인코딩된 데이터 - 총 72비트,
            //즉 9바이트가 Object Length 가 된다.
            // _ObjectLength = 9 즉 00001001 이 됨
            string _ObjLength = Convert.ToString(9,2).PadLeft(8,'0');
            // 9를 이진수로 변환한 후 8자리수를 채움.

            string resultBitCode = "";
            resultBitCode += _DSFID + _Precursor + _ObjLength;
            // 인코딩 정보 부분 기입 완료.

            resultBitCode += util.StrToBit(uNCode); //  "KKR" 국가코드
            resultBitCode += util.StrToBit(ulCode); // "ZZA" 기관코드
            resultBitCode += util.StrToBit(_Prefix);// "B" PREFIX
            resultBitCode += util.StrToBit(_IC);     // IC 부분 "CD"
            resultBitCode += util.StrToBit(_Serial);// Serial "EFGHI"
            resultBitCode += "00";      // Padding

        string encodedStr = util.BitToHex(resultBitCode).ToUpper();
        Console.WriteLine("KKR.ZZA.CD.EFGHI Encoded as : " + encodedStr);
        }
    }
}
```

또한 위에서 사용되는 EncUtil 클래스의 소스 전문은 다음과 같다.

```
using System;

namespace ODSExample
{
    /// <summary>
```

```
/// 각종 ISO/IEC 646 인코딩 관련 클래스 모음입니다.
/// </summary>
public class EncUtil
{
    public EncUtil()
    {
    }
    public string BitToHex(string resultBitCode)
    {
        char[] arr;
        string tmpStr = "";
        int tmpInt = 0;
        string resultStr = "";

        for(int i=0;i<96/4;i++)
        {
            arr=resultBitCode.ToCharArray(i*4,4);
            tmpStr = arr[0].ToString()
            + arr[1].ToString()
            + arr[2].ToString() + arr[3].ToString();
            tmpInt = Convert.ToInt32(tmpStr,2);
            resultStr += Convert.ToString(tmpInt,16);
        }
        return resultStr;
    }

    public string StrToBit(string tagData)
    {
        string result = "";
        string _CHARSET = "!ABCDEFGHIJKLMNOPQRSTUVWXYZ!";
        char[] arr;
        char[] tagChar;
        tagChar = tagData.ToCharArray();
        arr = _CHARSET.ToCharArray();
        int i = 0;

        foreach (char t in tagChar)
        {
            i = 0;
            foreach (char c in arr)
            {
        if(c.ToString().ToUpper().Equals(t.ToString().ToUpper()) == true)
        {
```

```
                string tmp = Convert.ToString(i,2).PadLeft(5,'0');
                result += tmp;
                break
            }
            else i++;
            }
            if(i >= arr.Length)
                result += Convert.ToString(50,2);
        }
        return result;
        }
    }
}
```

위에서 다룬 인코딩에 관한 .Net 구현 소스 일체는 http://www.ods.or.kr/ 의 자료실에서 "자산출입관리시스템 가이드 프로그램 소스" 자료를 다운받아 "kCode 인코딩" 패키지 부분에서 볼 수 있다. 위 패키지는 C# 닷넷으로 구성되어 있고 MS사에서 출시한 Visual Studio .net 2003(이하VS.net) 이상의 버전이 인스톨 된 컴퓨터에서 열어볼 수 있다.

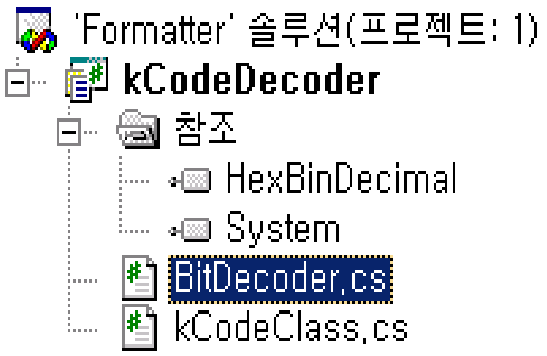
만일 VS.net 이 인스톨되지 않은 컴퓨터라면 컴파일하기는 어렵고 텍스트를 이용하여 소스코드만 열람할 수 있다. 소스코드만 열람하려면 울트라에디터, 아크로에디터 등을 이용하여 kCodeClass.cs 파일과 EncUtil.cs 파일을 열면 소스코드를 볼 수 있다.

"kCode 인코딩" 패키지에서 제공되는 함수 기능들을 정리하면 다음과 같다.

ClassName	Function	설명
EncUtil	-	● 각종 ISO/IEC 646 인코딩 관련 함수 모음 클래스
EncUtil	String BitToHex(string)	● 96비트의 비트 문자열(2진수)을 받아 Tag UID로 변환 ● String 형태의 Tag UID 값을 반환
EncUtil	string StrToBit(string)	● 문자열을 입력받아 5비트 인코딩한 2진수 문자열로 변환 ● 아스키문자에 해당하는 5Bit 인코딩 값을 2진수 스트링으로 반환

3.3.3. 코드 디코딩

본 구현에 사용된 미들웨어에는 ISO/IEC 15459 코드 체계로 인코딩 된 태그의 헤더 정보를 읽어 그에 따라 kCode로 변환할 수 있는 모듈이 탑재되어있다.
그러나 현재 대부분의 미들웨어에는 ISO/IEC 15459 코드 변환 및 해석 모듈이 탑재되어 있지 않으며 미들웨어에 별도의 모듈을 탑재하여 기능을 확장시켜야 한다. 이 장에서는 현재 적용된 kCode 디코딩 모듈의 소스 중 필요한 부분을 공개하여 차후 kCode를 지원하는 미들웨어 확장 모듈 개발에 중요한 도움이 될 수 있도록 하였다.



[그림 3.2] kCode 디코딩 솔루션 구성

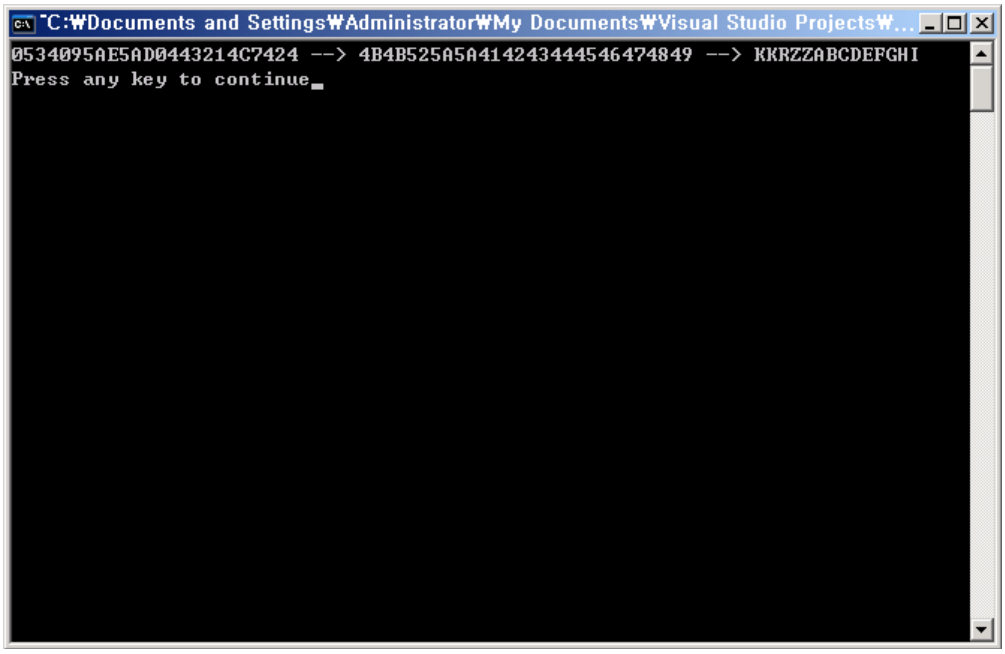
우선 http://www.ods.or.kr/ 의 자료실에서 "kCode 디코딩 패키지" 를 다운받아 압축을 풀면 [그림 3.2]과 같은 솔루션이 구성되어있다.

메인 클래스가 있는 파일은 kCodeClass.cs 파일이며 kCode 파일의 소스코드 전문은 다음과 같다.

```
using System;
using HexBinDecimal;
using Resolver;
using System.Text;

namespace ODSExample
{
class ODSTestApp
{

    [STAThread]
```

```
   static void Main(string[] args)
   {
      string tagUID = "0534095AE5AD0443214C7424";
      string tagBin = Conversion.HexToBin(tagUID);

      string _DSFID = tagBin.Substring(0,8);
      // DSFID 의 후반 5비트 부분이 ISO/IEC 15459 표준에서는
      // 15459 의 Root OID 를 의미하는 0 0101 로 세팅되어있다.
      if(_DSFID.Substring(3).Equals("00101") == false)
      {
         Console.WriteLine("ISO/IEC 15459 코드체계가 아닙니다");
         return
      }
      // PRECURSOR 부분의 후반 4비트 부분이 ISO/IEC 15459-4는 0101
      // ISO/IEC 15459-1은 0001로 세팅되어있다.
      string _PRECURSOR = tagBin.Substring(8,8);
      if(_PRECURSOR.Substring(4).Equals("0100")           ==           false           &&
_PRECURSOR.Substring(4).Equals("0001") == false)
      {
         Console.WriteLine("ISO/IEC 15459 코드체계가 아닙니다");
         return
      }

      int _BitLength = 0;
      if(_PRECURSOR.Substring(1,3).Equals("100") == true)
      {
         // 필드가 "100" : 6비트 인코딩방식
         _BitLength = 6;
      }
      else if(_PRECURSOR.Substring(1,3).Equals("011") == true)
      {
         // 필드가 "011" : 5비트 인코딩방식
         _BitLength = 5;
      }
      else
      {
         Console.WriteLine("ISO/IEC 646 인코딩 방식이 아닙니다");
         return
      }

      // Object Length 부분은 kCode 데이터가 총 몇바이트 기록되어있는지 알려준다.
      int _ObjLength = Conversion.BinToInt(tagBin.Substring(16,8));
      // 데이터 길이를 문자열비트로 나누면 몇글자가 데이터에 기록되어있는지 알수있다.
```

```
        int _TotalChars = (_ObjLength * 8) / _BitLength;

        // 헤더길이(24비트)를 제외한 데이터영역에서 코드를 추출해낸다.
        string kCode = "";
        for(int i = 0;i<_TotalChars;i++)
        {
            if(_BitLength == 5)
            kCode += BitDecoder.Bit5Char(tagBin.Substring(24+_BitLength * i,_BitLength));
            else if(_BitLength == 6)
            kCode += BitDecoder.Bit6Char(tagBin.Substring(24+_BitLength * i,_BitLength));
        }

        // kCode = "KKRZZABFBAAAEI" 식으로 값이
        // 들어가있으며 이를 16진수 아스키값으로 변환한다
        char[] temp = kCode.ToCharArray();
        string tagHex = "";

        // 각 문자마다 16진수 형태로 변환
        foreach (char c in temp)
        {
            tagHex += Convert.ToString(c,16).ToUpper();
        }

        byte[] encodeTemp;
        encodeTemp = new Hex(tagHex).ToBytes(); // hex 코드를 byte 형 배열로
        string kkrCode = Encoding.ASCII.GetString(encodeTemp);

        Console.WriteLine(tagUID + " --> " + tagHex + " --> " + kkrCode);
    }
}
}
```



[그림 3.3] kCodeClass.cs 디코딩 프로그램 실행화면 결과

위 소스코드의 실행 화면은 [그림 3.3]과 같다.
보이는 것처럼 "0534095AE5AD0443214C7424" 라는 인코딩 데이터가 해당하는 16진수 아스키 코드값으로 디코딩되고 그것이 다시 kCode로 변환되었음을 알 수 있다. 이 값은 본문 '4.2.3. 검색서비스(ODS) 연동 방안'에서 ODS와 통신할 때 중요하게 쓰이게 된다.

지금까지 3.3.3에서 다룬 디코딩에 관한 .Net 구현 소스 일체는 http://www.ods.or.kr/ 자료실에서 "자산출입관리시스템 가이드 프로그램 소스" 자료를 다운받아 "kCode 디코딩" 패키지 부분에서 볼 수 있다. 이 패키지는 C# 닷넷으로 구성되어 있고 VS.net 2003 이상의 버전이 인스톨 된 컴퓨터에서 열어볼 수 있다. 만일 VS.net 이 인스톨되지 않은 컴퓨터라면 컴파일하기는 어렵고 텍스트를 이용하여 소스코드만 열람할 수 있다. 소스코드만 열람하실 분은 울트라에디터, 아크로에디터 등을 이용하여 압축 파일내의 kCodeClass.cs 파일과 BitDecoder.cs 파일을 열면 소스코드를 볼 수 있다.

"kCode 디코딩"에서 제공되는 함수 기능들을 정리하면 다음과 같다.

ClassName	Function	설명
BitDecoder	-	● ISO/IEC 646 표준에 의거하여 비트스트링을 5비트 혹은 6비트 디코딩한 결과를 반환하는 클래스
BitDecoder	String Bit5Char(string)	● 들어온 비트문자열을 5비트 디코딩 결과값으로 반환. ● 디코딩된 문자 1개를 반환하며 에러시 '!' 문자 혹은 예외를 반환.
BitDecoder	string Bit6Char(string)	● 들어온 비트문자열을 5비트 디코딩 결과값으로 반환. ● 디코딩된 문자 1개를 반환하며 에러시 '!' 문자 혹은 예외를 반환.
Conversion	-	● 2진수 <-> 10진수 <-> 16진수 상호변환 함수를 제공하는 클래스
Conversion	string HexToBin(string)	● 16진수 문자열을 변환하여 2진수 문자열로 바꿔 반환 ● 에러시 예외를 반환
Conversion	int BinToInt(string)	● 2진수 문자열을 변환하여 Int 형태로 반환 ● 에러시 예외를 반환

제4장 자산출입관리 시스템 구축 사례

4.1 도입배경

4.1.1. 도입배경

RFID 객체 검색 서비스는 NIDA가 주도하여 오랜 시간에 걸쳐 연구가 이루어지고 구현된 우수한 서비스로 현재 여러 기관에게 서비스 중에 있다. 본 사업에서는 기존에 구현된 NIDA의 RFID 객체 검색 서비스를 활용/연계하여 자산 관리 확대 및 출입 관리제도 강화를 위한 시스템을 구축하였다. 이번에 구축된 시스템은 RFID 객체 검색 서비스를 활용하여 자산의 이동 및 존재 여부를 실시간 확인하고 OIS/OTS 등과 연계하여 자산 이력사항을 조회할 수 있어 RFID 객체 검색 서비스의 실증 사례를 제시하였다.

4.2 시스템 구성

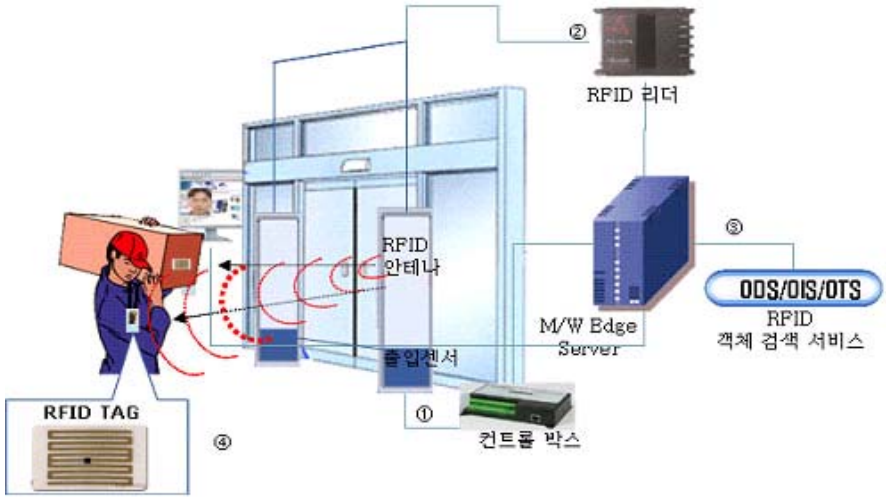
4.2.1. 시스템 구성내용

본 사업에서 구현된 시스템은 NIDA 사무실 내에 설치되었으며 NIDA 사무실이 건물의 본사와 지사에 나뉘어 있는 것을 이용하여 본사와 지사를 각각 독립된 별개의 기관으로 가정하여 설계되었다. 따라서 각 본사/지사에 독립된 RFID 검색 시스템이 구축되었고 National ODS 를 이용하여 두 건물간의 연동을 구현하였다.

이러한 방법으로 구축된 각 건물의 출입 게이트 부분을 도식화하면 [그림 4.1]과 같다. 각 건물에 설치된 미들웨어 Edge 서버는[18] RFID 리더 및 컨트롤 박스와 연계된 출입센서를 이용하여 태그의 감지 및 입출입을 판별한다. 판별 후 데이터를 적절한 메시지로 가공하여 데이터 처리 미들웨어[19] 서버로 전송한다.

18) Middleware Edge Server : 차세대 분산형 미들웨어의 형태. RFID 리더 등 외부 장비와 통신하고 제어하는 기능을 따로 분리해놓은 미들웨어로서 데이터 처리 모듈의 부하를 줄이기 위해 고안되었다.

데이터 처리 미들웨어 서버는 미들웨어 Edge 서버로부터 데이터를 넘겨받아 사용자의 요구, 혹은 필요에 따라 ODS/OIS/OTS 등의 RFID 객체 검색 서비스와 통신하며 미들웨어 Edge 서버와 사용자 및 RFID 객체검색 서비스 사이에서 중계자 역할을 담당한다.

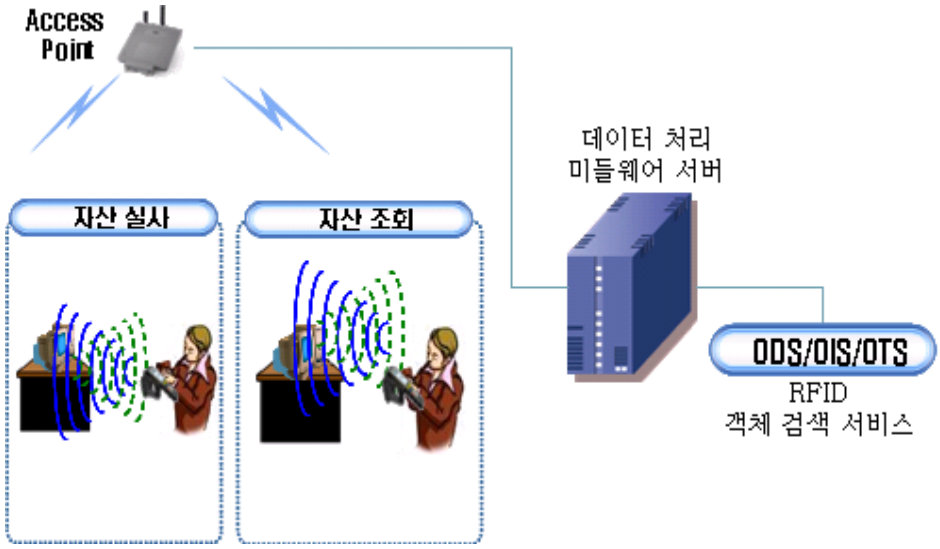


[그림 4.1] 출입 게이트 도식도

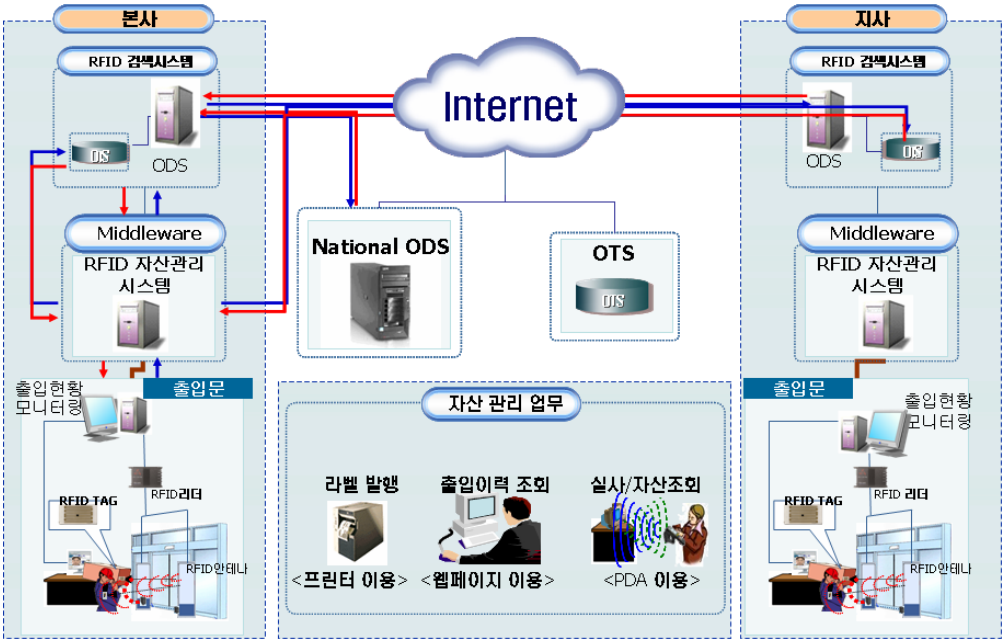
[그림 4.2]는 자산 조회 및 실사 서비스의 서비스 구성도이다. 자산 조회 및 실사 부문은 RFID 리더를 장착한 PDA를 이용하여 구현되었다. 자산 조회의 경우 자산에 부착되어 있는 태그 값을 PDA로 읽어 데이터 처리 미들웨어 서버로 전송하면 서버에서 RFID 객체 검색 시스템을 통해 해당 태그에 대한 정보를 받아온다. 사용자는 PDA를 이용하여 데이터 처리 미들웨어 서버에서 처리된 정보를 확인한다.

자산 실사의 경우 관리자 PC에서 실사 계획을 등록하여 자산 실사 대상 리스트를 설정한다. PDA에서는 실사 대상 리스트를 다운 받은 후 현장에서 자산에 부착된 태그를 읽어 실 자산과 DB에 등록된 자산이 일치하는지를 검사하고 맞지 않을 경우 수정작업을 통해 실 자산 정보와 DB에 등록된 정보를 일치시킨다. 최종 실사 결과는 PDA에서 데이터 처리 미들웨어 서버를 통해 DB에 반영된다.

19) Data Process Middleware Server : 사용자 요구에 따른 데이터 처리 및 RFID 검색 시스템과의 통신 기능 등이 분리되어있는 차세대 분산형 미들웨어의 형태.



[그림 4.2] 자산 조회 및 실사 서비스 구성도



[그림 4.3] 시스템 구성도

관리자 PC에서는 출입자 및 자산 정보에 대하여 조회, 수정, 추가, 삭제 등의 작업뿐만 아니라 스마트 라벨 프린터를 이용하여 RFID 태그를 발행하고 등록하는 작업까지 모두 한 자리에서 이루어진다.

위에서 본 것처럼 미들웨어는 사용자 및 RFID 리더, 센서, PDA 모두를 아울러 통신하고 필요한 정보를 제공해준다. 즉 RFID 객체 검색 시스템은 모두 미들웨어를 통하여 이용하게 된다.

[그림 4.3]은 본 구현에 적용된 시스템의 구성도를 간략하게 표시한 것이다. 본사와 지사는 서로 다른 기관으로 설정되어 각각 Local OIS와 Local ODS 를 가지고 있으며 이력서비스를 해주는 OTS와 Local ODS를 연동해주는 National ODS는 외부네트워크에 위치해 있다.

4.2.2. RFID 장비 및 미들웨어 소개

4.2.2.1 RFID 고정형 리더

900MHz 대역을 사용하며 EPC global 프로토콜(Class 0, Class 0+. Class1, Class1 Gen2) 지원이 가능한 멀티 프로토콜 리더를 채택하였다. 본 구현에서 필요한 리더의 기능으로는 탁월한 성능의 안티컬리전[20] 기능과 설치의 용이를 위해 부피가 작은 모델을 선택하였다. 또한 차후의 성능확장을 위해 펌웨어[21]의 확장 가능성 및 응용 프로그램 개발을 위한 지원성도 고려하였다. 이러한 사항을 고려하여 채택된 모델의 사양을 간략히 정리하면 아래와 같다.

구분	시스템 구성
아키텍처	Ultra3 SCSI/Ultra-160 SCSI, LVD
주파수 대역	910MHz~914MHz
RFID Tag 지원	EPC C0, C0+, C1, C1G2
네트워크 프로토콜	DHCP, TCP/IP, SNTP
리더 프로토콜	Alien Reader Protocol™, EPC Reader Protocols
호핑방식/ 호핑채널	FHSS / 15개
규 격	385mm(W) × 220mm(H) × 56mm(D)
채널 간격	150 KHz
통신 방법	RS-232, TCP/IP
송신전력	30dBm, 4W EIRP
무 게	2.0 Kg
작동 온도	-20℃ ~ 50℃ (MIC 인증시 검증 항목)
소프트웨어 지원	Java and .NET API 지원

[표 4-1] 본 구현에 사용된 고정형 리더 사양

20) Anti-Collision : 복수개의 태그를 읽을 경우 전파 간섭현상이 생겨 인식률이 떨어지는 현상을 방지하는 기능.

21) Firmware : 시스템의 효율을 높이기 위해 ROM에 들어 있는 기본적인 프로그램으로서 ROM에 고정되어 있기 때문에 하드웨어의 특성도 가지고 있으나, 실제로는 소프트웨어에 더 가깝다고 볼 수 있다.

4.2.2.2. RFID 안테나

안테나는 고정형 리더와 동일한 제조사에서 제작한 안테나를 채택하였다. 같은 회사의 리더와 안테나가 사용될 때 리더가 최적화 되어 제 성능을 발휘할 수 있으며 안테나가 가볍고 부피가 작아 필요한 상황에 맞게 커스터마이징[22]하여 배치가 가능하다는 점이 주요 선정 요인으로 작용하였다.

안테나는 특히 주변 환경에 노출이 되어있으므로 강우, 방진, 발열, 냉온에 강하고, 생활방수 기능이 상당히 높아야 반영구적으로 쓸 수 있다. 그 외에도 지그나 함체 등을 이용해서 안테나 제작 시 원하는 모양으로 내부 부착 가능해야 하기 때문에 용도에 따른 다양한 변형성도 고려되었다.

마지막으로 설치 및 유지비용을 고려했을 때 이번에 채택된 모델은 Poly-Carbonate와 Poly-Acrylic의 포장 재질로 구성되어 있어, 별도 방수, 내진, 발열, 냉온처리가 불필요하여 설치 및 유지비용이 적게 든다는 잇점이 있다.

[표 4-2] 에서 본 구현에 채택된 안테나에 관한 사양을 간략히 정리하였다.

구분	시스템 구성
주파수 범위	890MHz ~ 940MHz
이득/ 임피던스	최고 6dBi / 50 ohm
전파 넓이/ 편광각	40 degress nominal/ 직선
Cross Polarization Rejection	20 dB, min
Input Impedance	50 Ohm nominal
Return Loss	-15 dB across frequency range
커넥터	Reverse polarity TNC connector
크기/ 무게	284mm x 195mm x 43mm/ 680g

[표 4-2] 본 구현에 사용된 안테나 사양

22) Customizing :필요에 따라 개조, 변형 등을 통해 최적의 성능을 낼 수 있게 만들어주는 작업.

4.2.2.3. RFID 휴대형 리더

본 구현에 있어서 자산 조회 및 실사 작업과 금속태그 태깅을 위해 휴대형 리더가 필요하였다. 채택된 휴대형 리더는 전자파에 의한 통신장애 및 기기 오작동에 대한 테스트를 거쳤으며, 국내 표준 규격에 맞추어 정보통신부의 MIC 인증 획득한 제품으로 고성능 전용 CPU와 Inter XScale™ , 64MB RAM 을 탑재하여 자산 실사 등 대용량의 작업에 유리하다. 또한 무선 랜, 바코드를 모두 지원하여 자산에 부착된 태그가 외부 충격 등에 의해 기능을 상실한 경우에 바코드를 이용하여 작업을 계속 진행할 수 있어 높은 안정성을 지닌다. 또한 우수한 RFID 인식 성능으로 특수 제작된 RFID 안테나를 통해 최대 3.5m의 인식 거리를 유지하며 다중 프로토콜 지원으로 Gen 2 이전의 EPC Class 0, 0+, 1 및 ISO18000-6B 프로토콜 뿐만 아니라 EPC Class 1 Gen2 (ISO18000-6C) 까지 지원하여 현재 시장에 있는 거의 대부분의 태그를 인식 할 수 있다. [표 4-3]에서 본 사업에 채택된 휴대형 리더에 관한 사양을 정리하였다.

구분	시스템 구성
작동 주파수	910MHz ~ 914MHz
인식 거리, 속도	MAX 3.5m 이내, 10 Tag/sec 이상
메모리	ROM 64MB, RAM 128MB
디스플레이	3.5” TFT Color LCD 터치 스크린
HOST OS	Windows CE .NET v3.2
배터리	리튬 이온 충전형 7.4V, 1900mAh
지원 프로토콜	EPC Class 0, 0+, 1, ISO 18000-6B EPC Class1 Gen2 지원
규격	102(W)×62.5(D)×244(L)mm, 1400g

[표 4-3] 본 구현에 사용된 휴대형 리더 사양

4.2.2.4. 미들웨어

기업 체계 하에서의 미들웨어란 자동 인식을 담당하는 기기로부터 데이터를 빠르고 정확하게 수신하여 데이터의 신뢰성을 확보한 후 데이터 관리 애플리케이션에게 전달하는 소프트웨어를 말한다. 이러한 미들웨어는 장비 연결을 담당하는 Edgeware와 상위 애플리케이션들간의 연결을 담당하는 EAI(Enterprise Application Integration), 그리고 이러한 일련의 비즈니스 프로세스를 자동화하고 가시화하는 BPM(Business Process Management) 기능을 포함한다. 이번 구현에서는 현장에서 빠르고 정확한 데이터의 수집이 주요 요소이며, 이를 해결하기 위한 신뢰성 있는 미들웨어를 도입하였다.

특히 다량의 데이터 처리와 정확한 데이터 인식을 중요시하는 RFID 시장에서 다른 이기종 기기와의 연동 기능은 마들웨어의 중요한 기능이라고 할 수 있다. 본 시스템에 탑재된 미들웨어는 확장이 용이한 컴포넌트 구성 방식으로 이루어져 있으며 해당 기능들을 [표 4-4] 에 정리해 두었다.

기능명	내용
리더연동	고정형 RFID 리더와의 통신
리더제어	고정형 RFID 리더 설정 및 상태 제어.
데이터 필터 모듈	리더로부터 읽은 데이터를 사용자가 원하는 형태로 가공.
장애처리 모듈	통신장애 등 돌발적 장애로 인하여 연결 끊김시 자동 복구.
데이터 전송 모듈	가공한 데이터를 필요한 곳으로 전송해주는 기능.
컨트롤 박스 연동 모듈	컨트롤 박스와 연계하여 각종 H/W를 제어.
센서 방향 판별 모듈	센서를 이용하여 게이트 출입시 방향을 판별.
웹서비스 연동 모듈	SOAP, HTTP 등 프로토콜을 이용하여 각종 웹서비스 연동.
원격모니터링 모듈	통신프로토콜을 이용한 원격지 미들웨어 관리 기능.
이기종 리더 연동 모듈	다양한 종류의 리더기와 연동이 가능.
필터 추가 모듈	차후 필요한 데이터 필터의 간단한 확장성 지원.
로깅 모듈	데이터의 흐름, 예외 발생을 쉽게 알아볼 수 있는 로깅 기능.
상위 서비스 모듈과의 연계 기능	RFID 객체 검색 시스템과의 연동 기능.
자산/사람의 통합 프로세스 관리 기능	미들웨어상의 자산/사람 태그 판별, 관리 기능.
모니터링 기능	객체의 이동 이력현황을 실시간으로 조회하는 기능.

[표 4-4] 본 미들웨어 시스템에 탑재된 기능 리스트

4.2.3. 검색서비스(Object Directory Service) 연동방안

4.2.3.1. 개요

미들웨어는 리더로부터 kCode 정보를 받는다.[23] 이러한 kCode를 이용해 해당 태그에 대한 데이터 서비스가 가능한 OIS, OTS 등의 위치 정보를 알기 위해서는 미들웨어와 검색서비스간의 연동을 통해 위치 정보를 받아와야 한다. 본 구현에서는 기존에 NIDA에서 개발한 .NET Resolver 패키지를 사용하였다. .NET Resolver 패키지를 이용하면 RFID 검색서비스와 미들웨어 혹은 어플리케이션과의 인터페이스 역할을 담당하며 미들웨어로부터 kCode를 넘겨받아 이를 식별하고 FQDN 형식으로 변환, DNS의 NAPTR 타입으로 RFID 검색서비스에 질의하여 서비스의 종류 및 위치 값을 되돌려 받는 기능을 쉽게 수행할 수 있다.

4.2.3.2. 구현 (C# .Net)

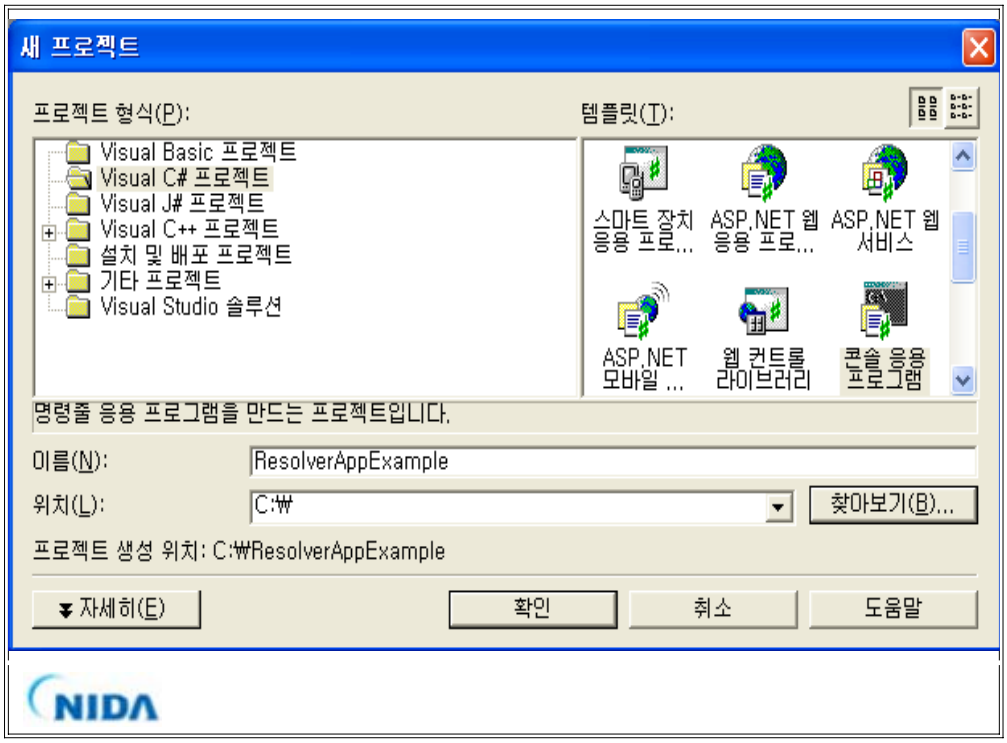
본 구현에서는 NIDA에서 개발한 .NET Resolver 패키지를 사용하였다. .NET Resolver 패키지에는 kCode 문자열을 각 필드별로 자동 분리, 검증하고 FQDN을 생성해주며 ODS 질의 기능까지 구현되어 있어 개발 시간을 크게 단축시켜준다. .Net Resolver 를 작성하고 실행하기 위해서는 VS.net 이 컴퓨터에 인스톨 되어 있어야 하며 그렇지 않은 경우 컴파일은 어렵고 소스코드 열람만 가능하다. 소스코드만 열람하실 분은 울트라에디터, 아크로에디터 등을 이용하여 확장자가 *.cs 인 파일들을 열면 소스코드를 볼 수 있다.

.NET Resolver를 이용하여 간단한 프로젝트를 생성하고 실행하는 과정은 다음과 같다.

① 새 프로젝트 생성

Visual Studio .NET 2003에서 새 프로젝트를 생성한다. 프로젝트 형식은 “Visual C# 프로젝트”로 하고, 템플릿은 “콘솔 응용 프로그램”을 선택한다. 새 프로젝트의 이름과 위치를 적당히 선택하여 입력하도록 한다.

23) 리더가 RFID 태그로부터 읽어들인 16진수 형태의 Tag UID를 kCode 형식으로 변환해주는 것은 리더쪽에서도 수행할 수 있고 미들웨어쪽에서도 수행할 수 있다. 본 구현에서는 미들웨어 모듈에서 kCode 변환작업을 수행하였다.



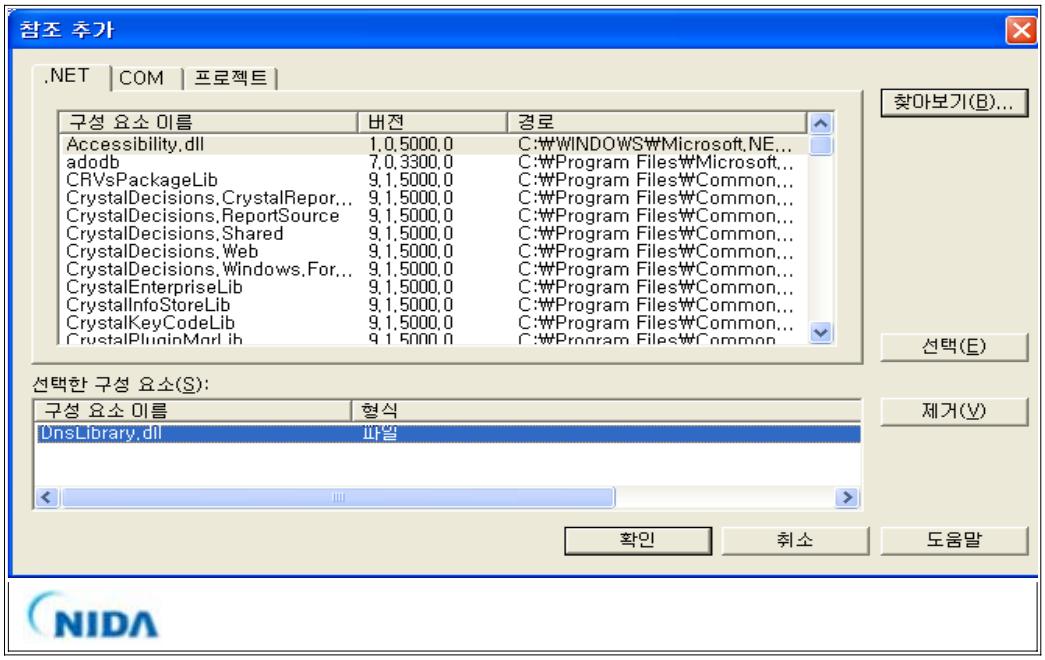
[그림 4.4] 새 프로젝트 생성

② 참조추가

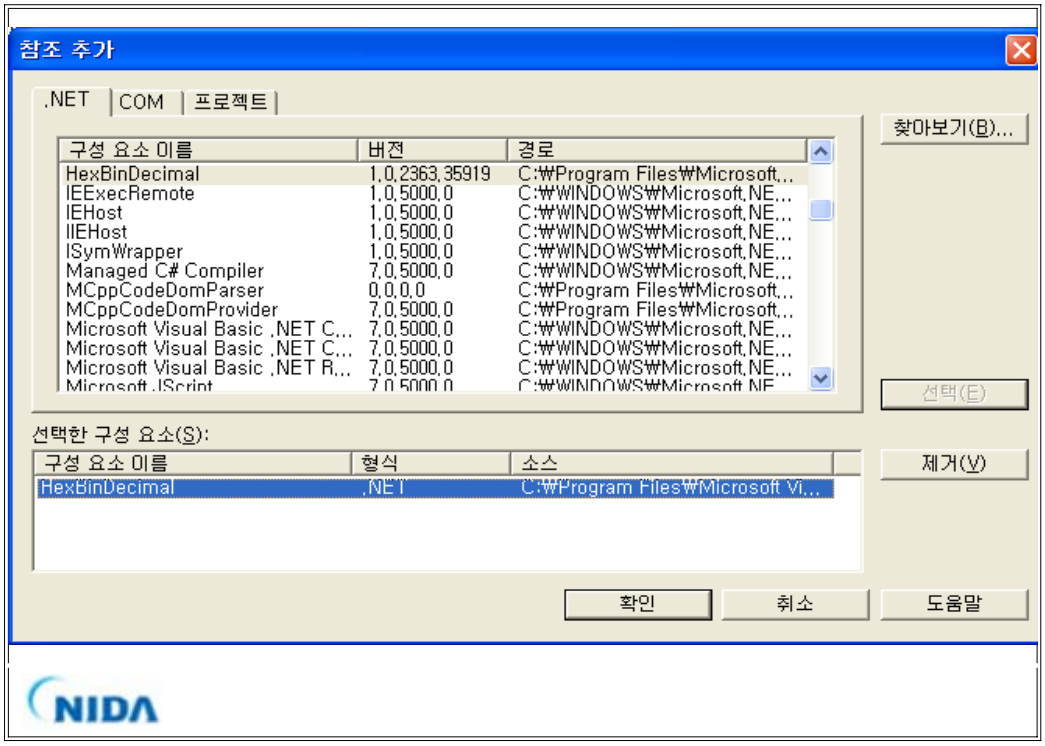
프로젝트가 생성되면 솔루션 탐색기에서 “참조”라고 되어 있는 폴더 모양 아이콘을 찾아 마우스 오른쪽 버튼을 클릭하여 “참조 추가”를 선택한다.

[그림 4.5] 참조 추가 1

아래와 같이 "참조 추가" 화면이 나오면 "찾아보기" 버튼을 눌러 다운받은 DnsLibrary.dll과 HexBinDecimal.dll을 찾아 선택한 뒤 "확인" 버튼을 누른다.



[그림 4.6] 참조 추가 2


참조 추가
.NET
COM
프로젝트
찾아보기(B)...
구성 요소 이름
버전
경로
HexBinDecimal 1.0.2363.35919 C:₩Program Files₩Microsoft...
IEExecRemote 1.0.5000.0 C:₩WINDOWS₩Microsoft.NE...
IEHost 1.0.5000.0 C:₩WINDOWS₩Microsoft.NE...
IIEHost 1.0.5000.0 C:₩WINDOWS₩Microsoft.NE...
ISymWrapper 1.0.5000.0 C:₩WINDOWS₩Microsoft.NE...
Managed C# Compiler 7.0.5000.0 C:₩WINDOWS₩Microsoft.NE...
MCppCodeDomParser 0.0.0.0 C:₩Program Files₩Microsoft...
MCppCodeDomProvider 7.0.5000.0 C:₩Program Files₩Microsoft...
Microsoft Visual Basic .NET C... 7.0.5000.0 C:₩WINDOWS₩Microsoft.NE...
Microsoft Visual Basic .NET C... 7.0.5000.0 C:₩WINDOWS₩Microsoft.NE...
Microsoft Visual Basic .NET R... 7.0.5000.0 C:₩WINDOWS₩Microsoft.NE...
Microsoft.IScript 7.0.5000.0 C:₩WINDOWS₩Microsoft.NE
선택(E)
선택한 구성 요소(S):
구성 요소 이름
형식
소스
HexBinDecimal .NET C:₩Program Files₩Microsoft Vi...
제거(V)
확인
취소
도움말
NIDA

[그림 4.7] 참조 추가 3

③ 참조 확인

솔루션 탐색기의 “참조” 폴더를 열어 DnsLibrary.dll과 HexaBinDecimal이 참조에 포함되어 있는지 확인한다.

[그림 4.8] 참조 확인

④ 메인 클래스 생성

다음으로 리졸버를 실행해 볼 수 있는 간단한 클래스를 작성하여야 한다.

입력으로 kCode를 Hexadecimal 값으로 받도록 하고 이를 코드 분석기로 넘겨주면 코드 형식에 맞는 FQDN으로 변환, 로컬 ODS에 질의하여 서비스의 위치를 넘겨받는 상황을 가정한다.

메인 클래스 작성 예는 다음과 같다.

```
using System;
using System.Text;
using System.Text.RegularExpressions;
using Resolver;
using HexBinDecimal;

namespace Resolver
{
    class Formatter
    {
        [STAThread]
        static void Main(string[] args)
        {
```

```
            Console.WriteLine("Please enter the code(hex): (type 'quit' to exit)");

            while(true)
            {
                Console.Write(">");
                string code = Console.ReadLine();

                if (code.ToLower() == "quit") break

                if (code=="" || Regex.IsMatch(code, @"[g-z,G-Z]|WW"))
                {
                    Console.WriteLine("Illegal code !! (NULL or not Hexadecimal type)");
                }
                else if (code.Length < 8)
                {
                    Console.WriteLine("code length : minimal-32bits");
                }
                else
                {
                    string hexCode = code.ToUpper();

                    CodeAnalyzer ca = new CodeAnalyzer();
                    ca.ParseHexCode(hexCode);
                }
            }
        }
    }
}
```

⑤ ResolverApp 클래스 생성

ResolverApp 클래스에서 질의할 로컬 ODS 서버에 대한 IP 설정을 하게 되며, 설정한 로컬 ODS 서버 주소와 도메인, DNS 타입을 가지고 쿼리를 생성하게 된다. 도메인은 입력 받은 코드의 FQDN 표현이며 DNS 타입은 NAPTR 타입만을 이용한다.

```
using System;
using System.Net;
using System.Collections;
using DnsLibrary;

namespace Resolver
{
    public class ResolverApp
    {
        public ResolverApp(string fqdn)
        {
            string ip = "127.0.0.1"; //DNS server IP
            IPAddress dnsServer = IPAddress.Parse(ip);
```

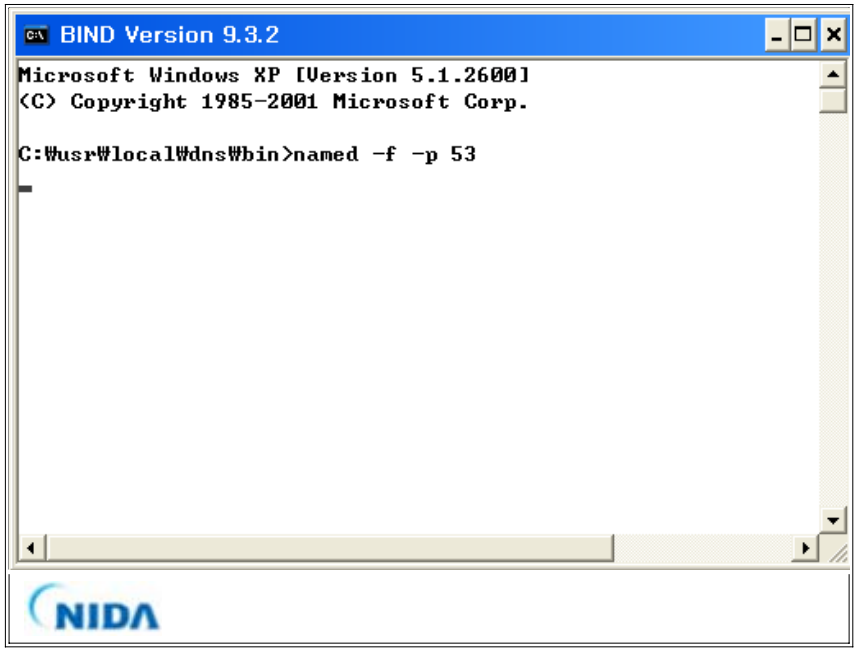
```
            string domain = fqdn;

            Query(dnsServer, domain, DnsType.NAPTR);

            ArrayList data = new ArrayList();

            data = new NaptrRecord().Make; //NAPTR Record

            //for (int i=0; i<data.Count; i++)
            //  Console.WriteLine(data[i]);
        }

        private static void Query(IPAddress dnsServer, string domain, DnsType type)
        {
            try
            {
                // create a DNS request
                Request request = new Request();

                // create a question for this domain and DNS CLASS
                request.AddQuestion(new Question(domain, type, DnsClass.IN));

                // send it to the DNS server and get the response
                Response response = Resolver.Lookup(request, dnsServer);
                // check we have a response
                if (response == null)
                {
                    Console.WriteLine("No answer");
                    return
                }
                // display each RR returned

Console.WriteLine("----------------------------------------------------------------");

                // display whether this is an authoritative answer or not
                if (response.AuthoritativeAnswer)
                {
                    Console.WriteLine("authoritative answer");
                }
                else
                {
                    Console.WriteLine("Non-authoritative answer");
                }

                // Dump all the records
                foreach (Answer answer in response.Answers)
                {
                    Console.WriteLine("{0} ({1}) : {2}", answer.Type.ToString(),
              answer.Domain, answer.Record.ToString());
```

```
            }

        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
    }
  }
}
```

⑥ 리졸버 테스트

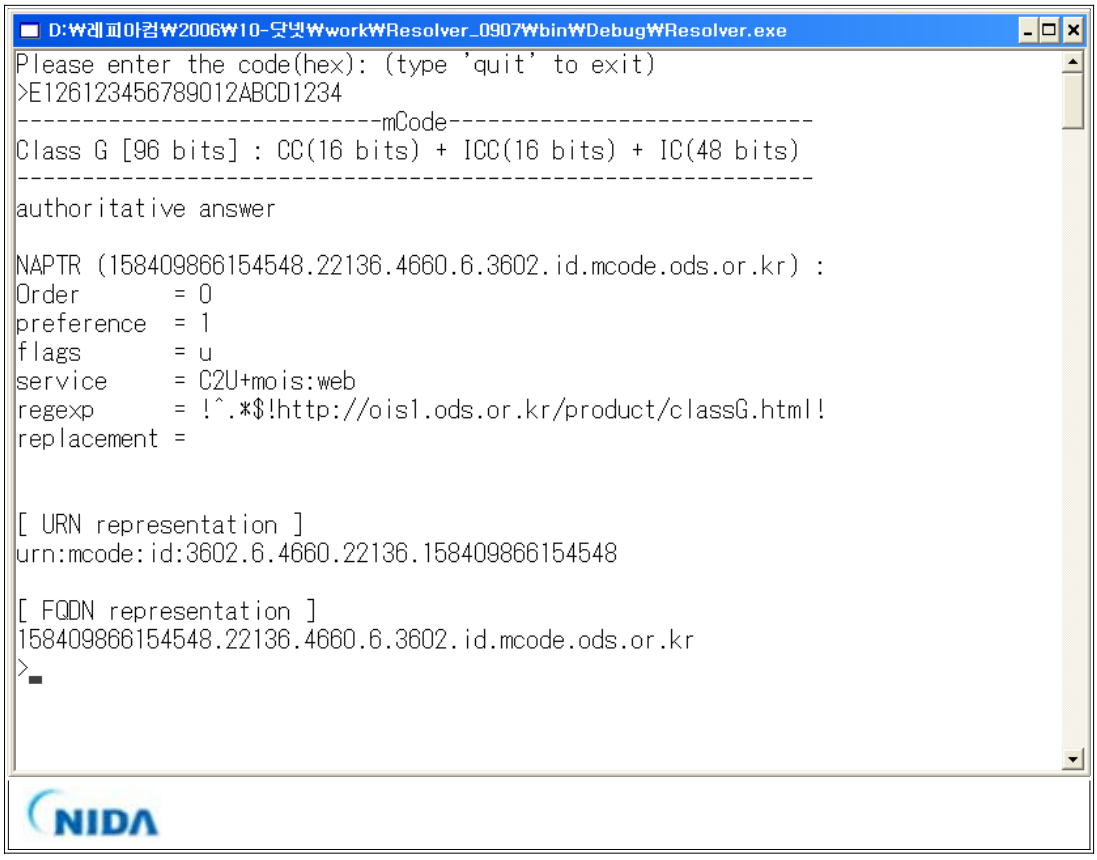
메인 클래스 작성 뒤 컴파일이 정상적으로 완료 되었다면, 리졸버가 올바르게 실행되는지 테스트 해보도록 한다. 리졸버를 테스트하기 위해서는 로컬 ODS가 구축 되어 있어야한다.

로컬 ODS 구축이 완료 되었다면 다음과 같이 네임 서버를 실행시킨다.



[그림 4.9] BIND 실행

네임 서버를 가동시킨 뒤 프로그램을 실행하여 질의하고자 하는 코드를 Hexadecimal 값으로 입력한다. 프로그램에서 입력 코드의 분석과 DNS 질의를 통해 해당 서비스의 위치가 담긴 NAPTR Record를 되돌려주게 된다. 실행 예로 mCode중 Class G에 해당하는 코드 값인 E126123456789012ABCD1234를 질의하여 그 결과를 확인해 보았다.



[그림 4.10] .NET Resolver 실행 예제

위의 패키지와 관련하여 자세한 사항은 http://www.ods.or.kr/ 의 자료실의 '.NET Resolver 활용지침서 V1.0' 문서를 참고하기 바란다.

4.2.4. 객체정보서비스(Object Information Service) 구현방안

4.2.4.1. 개요

객체정보서비스는 객체에 대한 정보를 제공하며, 기관 자체적 또는 위탁 관리 및 운영된다. 따라서 다양한 형태로 구현될 수 있다. 이와 같이 다양한 형식의 OIS 정보에 접근을 위한 메시지 타입과 인터페이스에 대한 표준이 필요하다. 또한 OIS는 이력정보의 저장과 함께 OTS에 이를 등록하기 위해 요청 하는 메시지를 전송하는 기능도 갖추어야 한다. 객체정보서비스는 기존의 물품 카탈로그나 창고관리 DB 등의 이미 구축되어 있는 정보를 사용하여 서비스 가능하며, 다양하게 저장되어진 정보에 접근하기 위한 표준화된 API를 사용하여야한다. OIS는 각 기관별로 다양한 형태로 구성이 가능하며, 본 사업에서는 XML 기반의 Java 웹서비스로 구현하였다.

4.2.4.2. OIS DataBase 구성

객체정보서비스는 HTML, RPC, DataBase 등 다양한 형태로 구현이 가능하다. 본 구현에서는 "자산 및 출입자 관리"라는 테마에 초점을 맞추고 Oracle DB를 이용하여 구현하였다. 이번 구현에 있어서 OIS의 DB는 Instance DB 와 History DB 및 Static DB로 구성되며 Instance DB에는 각 자산 및 출입자의 이미지 정보와 기존에 쓰이던 자산 코드 번호 및 발급된 Tag UID 번호가 쌍으로 들어가 있다. History DB 에는 각 자산 및 출입자의 이력정보가 들어간다. Static DB는 RFID 검색 시스템을 도입하기 기 운영중이던 자산 관리 ERP 시스템에서 이용하던 DB로 자산 코드 번호를 키 값으로 하여 자산 이름, 담당자, 담당부서, 구입가격 등에 이르기까지 이미지 정보를 제외한 자세한 개별 정보가 저장되어 있다.[24] 각 DB는 다시 자산의 정보가 기록된 자산DB와 출입자의 정보가 기록된 출입자DB로 나누어진다. [표 4-5] 에서는 이번 구현에 쓰인 OIS DB의 구성을 표로 수록하였다.[25]

---

24) 이번 구현에서는 기 구축 되어있던 자산코드번호 기반의 ERP 시스템을 활용하여 Tag UID와 코드를 매핑시켜 주는 Instance DB가 필요했지만 경우에 따라서는 Instance DB를 빼놓고 구현할 수도 있고 Instance DB를 단순한 매핑 테이블이 아닌 개별상품의 세세한 정보로 구현할 수 도 있다.

25) 자산의 Static DB는 기존 자산 코드번호 기반의 ERP 시스템을 활용하였으며 OIS 인터페이스에서 그 연동을 담당한다. 따라서 자산 OIS 의 DB는 Instance, Historical 테이블 만으로 구성되었다.

OIS DB 구분	필드 설명
출입자 OIS (Static & Instance)	EMP_UID : 출입자 Tag UID
	EMP_DEP : 출입자 소속 부서
	EMP_NAME : 출입자 성명
	EMP_PW : 출입자 비밀번호
	EMP_NO : 출입자 사원번호
	EMP_IMG : 출입자 사진 정보
자산 OIS (Instance)	AST_UID : 이동 자산 Tag UID
	AST_CD : 이동 자산 고유 코드 번호
	AST_IMG : 이동 자산 사진 정보
출입자 OIS (Historical)	EMP_UID : 출입자 Tag UID
	MOVE_DDTT : 출입자 출입시 입실/퇴실 시간
	INOUT_STATE : 출입자의 입실/퇴실 상태
자산 OIS (Historical)	AST_UID : 자산 Tag UID
	MOVE_DDTT : 자산 반출입시 반입/반출 시간
	INOUT_STATE : 자산의 반입/반출 상태
	EXTRA_UID : 자산 반출입시 자산 소지자 UID

[표 4-5] 본 OIS 구현에 사용된 DB 구성

이에 따라 본 구현에서 사용자의 요구하는 데이터 유형에 따라 OIS 인터페이스가 수행하고 돌려주는 데이터 타입은 다음과 같다.

사용자 요구 데이터유형	OIS 가 수행하는 작업내용
Static	미들웨어로부터 Tag UID 값을 받아 Instance DB에 기재되어 있는 Tag UID값과 매핑되어 있는 자산/출입자 코드번호를 찾아낸 후 코드번호를 Key값으로 사용자가 요구하는 정보들을 Static DB에서 받아와 XML 형식으로 돌려준다.
History	미들웨어로부터 Tag UID 를 받아 Historical DB에서 해당되는 태그의 이력정보를 검색하여 XML 형식으로 돌려준다.

[표 4-6] 데이터 유형에 따른 OIS 인터페이스의 역할

본 시스템에서의 DB 구성은 자산의 경우 Static DB, Instance DB, History DB가 모두 존재하지만 출입자의 경우 새로 구축하게 되어 History DB 및 Instance DB에 Static DB를 통합한 통합 DB, 이렇게 두 부분으로 되어있다. OIS는 미들웨어로부터 Tag UID 값을 받아 kCode 체계에 맞게 디코딩한 값으로 정보의 유효성을 판별한 후 해당 코드에 맞는 데이터를 가져오게 된다.

4.2.4.3. OIS 프로토콜 구성

기존의 분산 시스템은 플랫폼, 개발 언어, 디바이스에 의존적이고, 서로 다른 통신 프로토콜을 사용하여 상호 운용성을 제공하기 어려웠다. 반면, 웹 서비스는 플랫폼, 개발 언어, 디바이스 및 통신 프로토콜에 상관없이 어플리케이션 간의 데이터 공유와 통신이 가능하고 독립적인 개발 및 실행이 가능하여 시스템 간의 상호 운용성을 제공하도록 설계 되었다.

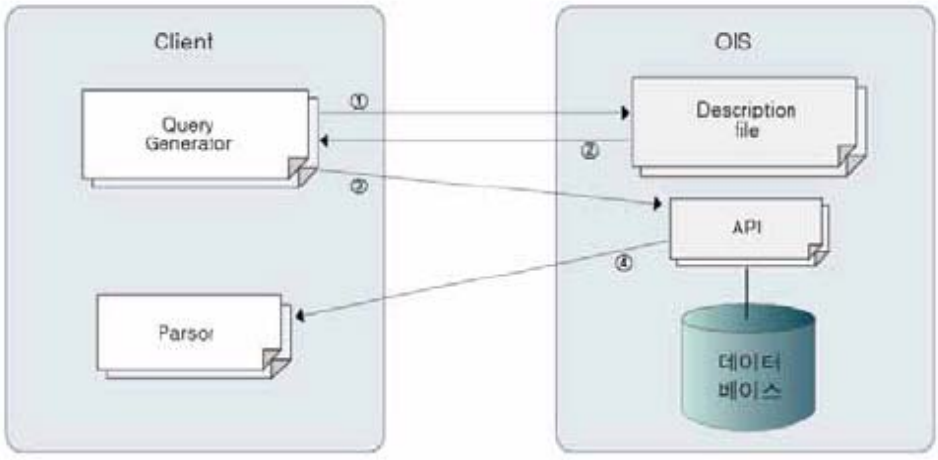
본 구현에서는 Tomcat 버전 3.1에서 SOAP 기반의 Axis 웹서비스를 이용해 구현하였다. Axis 는 표준 WSDL[26]을 지원하여 호환성과 확장성이 좋을 뿐만 아니라 어떠한 플랫폼에도 (.Net, Solaris 등) 연동이 잘 되며 빠른 처리속도를 가지고 있다. 여기서 쓰인 Axis 버전은 1.1 이며 상위버전이 개발 중에 있다.

기본적으로 미들웨어와 OIS 와의 데이터를 주고받는 과정은 다음과 같다.

① ODS를 통해 얻은 OIS의 Service URI로 접근정보파일(Description file)을 요청한다.
② OIS의 접근을 위한 접근정보파일(Description file)을 응답한다.
③ OIS에서 제공된 접근정보파일을 보고 OIS에 RFID 코드에 관련된 정보를 요청한다.
④ 요청된 RFID 코드에 해당하는 정보를 제공한다.

---

26) WSDL(Web Service Description Language)은 XML 용어의 하나로 프로그래밍 단계의 자동적 통합에 요구되는 모든 기술 세목들을 구성한다. WSDL은 어떻게 보면 방법론 디스크립션 부분, 아규먼트 형태 및 결과값에 있어서, 또는 WSDL이 서비스의 호출을 원하는 이용자에게 상세한 구현 절차 를 설명함으로 IDL에 비교된다. WSDL은 밑단의 구현 절차를 개의치 않고 메시지 프로토콜의 요건적 정의의 설명에 주력한다.



[그림 4.11] OIS 서버 접근 절차

[단계 1]에서 주고 받게 되는 정보가 WSDL 정보이다. WSDL 정보는 본 시스템 본사 OIS에 접근하기 위한 정보를 XML 방식으로 담고 있다. 미들웨어에서는 이 정보를 이용하여 OIS 와 통신하고 원하는 데이터를 질의한다. 이번 미들웨어구현에 사용된 개발툴인 .Net 2003 버전에서는 WSDL 을 이용한 웹서비스 통신 표준을 안정적으로 지원한다. [단계 2,3]에서 미들웨어는 접근 정보 파일을 넘겨받아 다시 OIS에게 RFID 코드와 관련된 정보를 요청하게 된다.[27]

이번 OIS 에서 지원하는 User 용도의 데이터 입출력 API를 살펴보면 다음과 같다.[28] 단, 여기에 쓰인 소스코드는 소스 파일이 Solaris 기반인 만큼 메모장에서 열어서는 알아보기 힘들며 별도의 텍스트 뷰어 (울트라에디터, 아크로에디터 등)를 이용해야 소스코드를 제대로 열람할 수 있다.

---

27) 현재 미들웨어가 OIS와 제대로 통신하기 위해서는 접근 정보 파일을 분석하여 그에 맞는 OIS 접근 프로토콜을 동적으로 생성해야 한다. 이는 결과적으로 구현에 어려움이 따를 수 있어 설계 당시(2006년 9월) 모바일RFID포럼 OIS 표준 제정이 진행중이므로, 이번 구현에서는 임시적으로 제작한 동적 구성 프로토콜을 사용하였다.

28) OIS 데이터 입출력 API 는 호출시 별도의 IP Chain 등 인증절차를 거치며 인증 과정에서 오류가 발생할 경우 보안모듈에 의하여 즉시 호출이 차단된다.

1) DataSet GetStaticData(String[] DataParameter)
- DataParameter 로 넘어오는 Tag UID 정보에 해당하는 OIS Static 정보 질의결과를 DataSet 형태로 리턴해준다. DataParameter 배열에는 Tag UID 뿐만 아니라 생성날짜, 이름, 소속 부서명 등의 다양한 검색 정보가 들어갈 수 있으며 OIS 는 이 배열을 분석하여 사용자의 요구에 맞는 정보로 DataSet 을 구성하여 넘겨준다.

2) DataSet GetHistoryData(String[] DataParameter)
- DataParameter 로 넘어오는 Tag UID 정보에 해당하는 OIS 이력정보 질의결과를 DataSet 형태로 리턴해준다. DataParameter 배열에는 Tag UID 및 이력 기록날짜, 출입 정보의 검색 정보가 들어갈 수 있으며 OIS 는 이 배열을 분석하여 사용자의 요구에 맞는 정보로 DataSet 을 구성하여 넘겨준다.

3) Bool SetHistoryData(DataSet DataParameter)
- DataSet 형태로 넘어오는 데이터를 검사하여 OIS Historical DB 형식에 적합한지 판단한 후 OIS Historical DB 에 이력 정보를 등록한다. 여기서 넘어오는 DataSetParameter 는 이전의 StaticData API에서 넘겨받는 DataSet 과는 다른 형태의 이력정보로 구성이 되어 있으며 등록이 성공할 경우는 True, 실패할 경우는 False 값을 반환한다.

또한 OIS 에서 지원하는 관리자 용도의 데이터 입출력 API 를 살펴보면 다음과 같다.

1) Bool SetStaticData(DataSet DataParameter)
- DataSet 형태로 넘어오는 데이터를 검사하여 OIS DB 형식에 적합한지 판단한 후 OIS Static DB 혹은 Instance DB에 등록한다. 등록이 성공할 경우는 True, 실패할 경우는 False 값을 반환한다.

2) int DelStaticData(DataSet DataParameter)
- DataSet 형태로 넘어오는 Tag UID 정보에 해당하는 OIS 정보를 DB 상에서 삭제한다. 삭제 시 쓰이는 키는 Tag UID 외에도 다양한 조건의 지정이 가능하지만 한번 삭제된 데이터는 다시 복구하는데 많은 어려움이 따르기 때문에 사용 시에 주의를 기울여야 한다. 성공적으로 삭제 시 0 이상의 자연수를 리턴한다. 이때 자연수는 지워진 OIS 정보 레코드의 개수이다. 삭제 실패 시 -1 이 리턴된다.

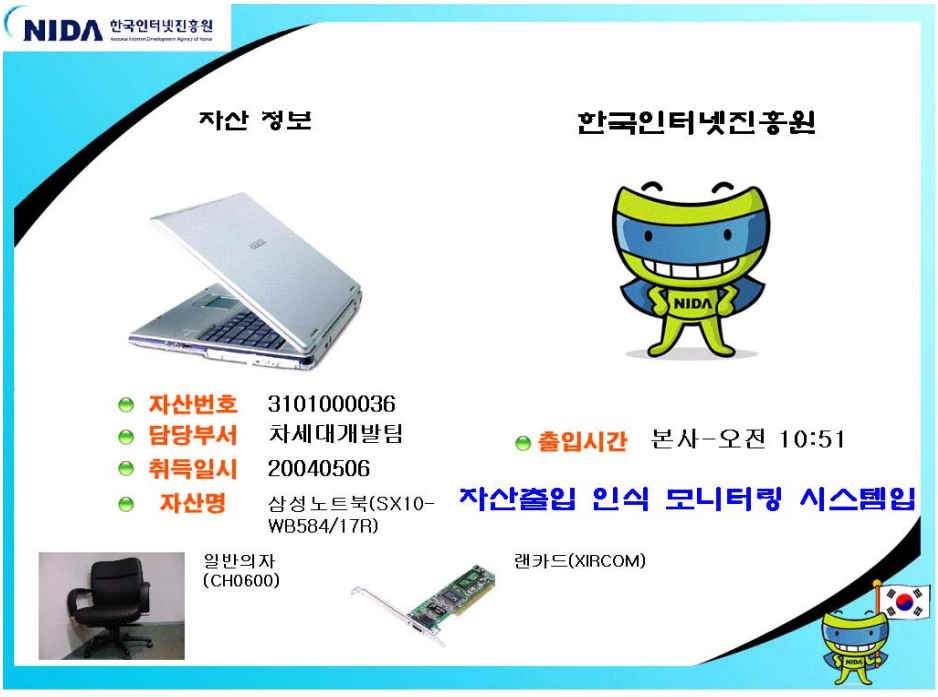
3) Bool ModifyStaticData(DataSet DataParameter)
- DataSet 형태로 넘어오는 Tag UID 정보에 해당하는 OIS Static & Instance 정보를 찾아 수정한다. 이때 쓰이는 키는 Tag UID만 쓸 수 있다. 즉 한번에 한 레코드만 수정이 가능하다. 성공적으로 정보를 수정했을시 True, 실패 시 False 를 리턴한다.

4) int DelHistoryData(DataSet DataParameter)
- DataSet 형태로 넘어오는 Tag UID 정보에 해당하는 OIS 이력 정보를 Historical DB 상에서 삭제한다. 삭제 시 쓰이는 키는 Tag UID 외에도 다양한 조건의 지정이 가능하지만 한번 삭제된 이력정보는 다시 복구하는데 많은 어려움이 따르기 때문에 사용 시에 주의를 기울여야 한다. 성공적으로 삭제 시 0 이상의 자연수를 리턴한다. 이때 자연수는 지워진 OIS 정보 레코드의 개수이다. 삭제 실패 시 -1 이 리턴된다.

4.2.4.4. OIS 구동 시나리오

RFID 자산 및 출입관리 게이트는 NIDA의 본사와 지사에 설치되어 있으며 각 층마다 고정형 리더기와 연결되어있는 미들웨어 Edge 서버에는 실시간 출입자 및 자산 모니터링 어플리케이션이 탑재되어있다.
출입자가 자산을 소지한 채로 게이트를 지나면 게이트에 부착된 센서에서 방향을 판별하고 고정형 리더기에서 읽은 태그 리스트와 함께 미들웨어로 전송된다. 미들웨어에서는 데이터를 특정 메시지 형태로 가공한 후 OIS 등에 질의하여 각 Tag ID 에 해당하는 정보를 가져오게 된다. [그림 4.12] 는 본사에 설치된 출입문에 자산이 통과했을 때의 모니터에서 보여지는 화면이다. 한 화면에 4개까지 자산 목록이 보여지며, 출입시간이 같이 기록된다.

미들웨어에서는 태그의 Bank 01에 있는 00F~20F 영역을 읽어 ISO/IEC 15962 표준에 부합되는 태그인지 판별한 후 태그의 인코딩 정보를 읽어 디코딩을 하게 된다. 이러한 과정을 거쳐 FQDN 을 생성하며 자산번호 및 사번을 추출해낸다. 미들웨어는 여기서 생성된 FQDN 으로 Local ODS에게 질의하여 해당 태그의 정보를 가진 OIS의 URL을 찾는다. 해당 정보를 가진 OIS URL을 Local ODS로부터 받아오면 해당 OIS에 자산 및 출입자 정보에 관련된 질의를 하게 된다. 이때 앞에서 소개한 DataSet GetStaticData(DataSet DSParameter) 형태의 API 가 호출된다. 미들웨어는 GetStaticData API를 호출함으로서 OIS 로부터 Static 정보 및 Instance 정보를 받아와 실시간 모니터링 어플리케이션에 전송한다. 또한 미들웨어는 이력관리를 위해 자신의 위치에서 특정 출입자와 자산이 지나갔다는 사실을 본사에 있는 Local OIS에게 알린다. 이를 위해서 위에서 소개한 Bool SetHistory(DataSet DSParameter) 형태의 API 가 호출된다. 이 API 가 호출됨으로서 OIS 는 이력정보 DB에 해당 태그의 출입자와 자산이 입실/퇴실했는지를 기록하고 OTS를 호출하여 OIS 자신이 가진 이력정보에 변동이 있었음을 알린다.



[그림 4.12] 자산 통과시 모니터 실행 화면

4.2.5. 객체이력관리서비스(Object Traceability Service) 구현방안

4.2.5.1. 개요

객체이력관리서비스는 객체의 이동해간 OIS의 위치정보를 저장한다.
이력정보는 객체의 이동에 따라 저장되는 OIS의 위치가 변경될 때마다 추가적으로 OIS의 위치가 기록된다. 따라서 관리하는 서버별로 정보의 저장 크기를 제한하여 방대한 데이터가 지속적으로 늘어나는 것을 방지하여야 한다.
OTS의 관리는 각 기관별 혹은 공공기관에서 관리가 가능하며, 저장 방식이나 구축형태도 관리하는 곳에 따라 다양한 형태의 구축이 가능하다.
그러나 외부로부터의 질의 및 응답에 관한 메시지의 표준은 유비쿼터스 센서 네트워크가 연동되기 위해서 필수적으로 제공되어야 한다. 이에 따른 객체이력관리서비스(OTS) 표준은 모바일RFID포럼(MRF)에서 진행중이다.
본 구현에서는 객체이력서비스 구축형태와 메시지를 XML 기반의 Java 웹 서비스로 구현하였다.

4.2.5.2. OTS DataBase 구성

객체이력정보서비스는 OIS 와 마찬가지로 다양한 형태로서 구현이 가능하다. 본 구현에서는 차후 확장성 및 데이터 처리 용량 등을 고려하여 OIS 와 동일하게 Oracle DB를 이용하여 구현하였다. 이번 구현에 있어서 OTS의 DB는 기본적인 OTS의 기능을 수행하기 위해 비교적 가볍고 간단하게 설계 되었다.
본 구현에 쓰인 DB 구조도는 다음과 같다.

구분	필드 설명
OTS (자산)	UID : 이동 자산 Tag UID
	MOVE_DDTT : 이동 자산 반입시간
	OIS_URI : 이동한 자산의 반입 OIS URI
OTS (출입자)	UID : 출입자 Tag UID
	MOVE_DDTT : 출입자의 출입시간
	OIS_URI : 이동한 출입자의 출입 OIS URI

[표 4-7] 본 OTS 구현에 사용된 DB 구성

UID 필드는 Tag UID 정보가 들어간다. Tag UID 는 기본키로 지정되어 있으며

OIS_URI 는 Tag UID가 거친 OIS의 URI 주소를 나타낸다.

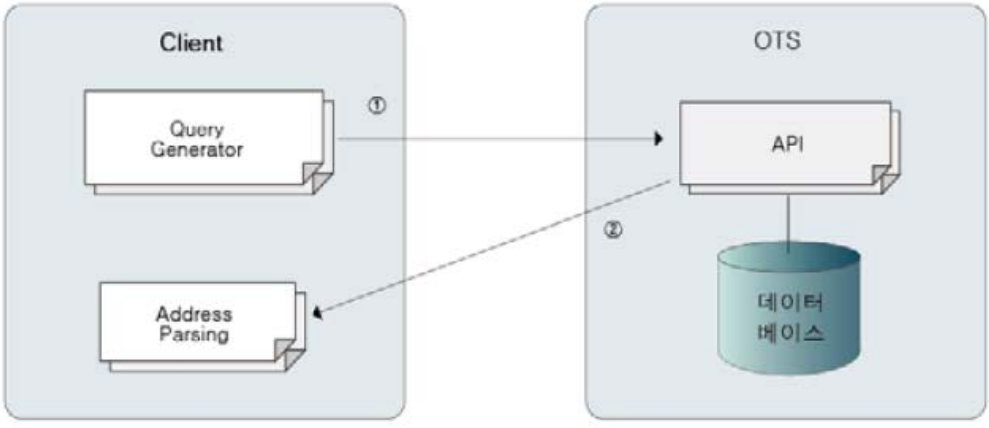
OTS는 미들웨어로부터 Tag UID 값을 받아 kCode 체계에 맞게 디코딩한 값으로 유효한 정보인지 판별한 후 해당 코드에 맞는 OIS URI List 를 가져오게 된다. 즉 OTS에는 이력정보 자체가 저장되어 있는 것이 아니라 해당 Tag UID 가 거쳐온 OIS 의 목록이 저장되어 있는 것으로 실제 이력 정보는 미들웨어가 OTS로부터 OIS의 목록을 넘겨받아 각 OIS에 일일이 이력 정보를 질의하여 가져오게 된다.

4.2.5.3. OTS 프로토콜 구성

본 구현에서의 OTS는 앞에서 소개한 OIS 프로토콜의 구성과 마찬가지로 웹서비스의 장점을 살려 플랫폼, 개발 언어, 디바이스 및 통신 프로토콜에 상관없이 어플리케이션 간의 데이터 공유와 통신이 가능하고 독립적인 개발 및 실행이 가능하여 시스템 간의 상호 운용성을 제공하도록 설계 되었다.

기본적으로 미들웨어와 OTS 와의 데이터를 주고 받는 과정은 다음과 같다.

① Tag UID로 Local ODS에서 OTS의 URI를 받아 해당 OTS에 RFID 코드의 이력을 가지고 있는 OIS_URI List를 요청한다.
② OTS는 요청된 Tag UID 가 거쳐간 OIS_URI 리스트를 제공한다.



[그림 4.13] OTS 서버 접근 절차

즉 미들웨어는 Tag UID에 해당하는 정보를 찾기 전에 해당 Tag UID에 관한 이력 정보가 어디있는지 Local ODS에 질의 하게 된다. Local ODS는 질의를 받아 OTS의 URI를 검색하여 미들웨어에게 넘겨주면 미들웨어는 해당 OTS에게 RFID 코드와 관련된 OIS_URI 리스트를 요청하게 된다.

이번 구현된 OTS 에서 지원하는 데이터 입출력 API 를 살펴보면 다음과 같다.

1) DataSet GetTraceData(String TagUID, Date sDate, Date eDate)
- 인자값으로 넘어오는 Tag UID 정보에 해당하는 OIS URIs 를 DataSet 형태로 리턴해준다. 검색은 날짜별로 지정해 줄 수 있으며 날짜가 생략된 경우에는 가지고 있는 모든 레코드를 DataSet 형태로 반환한다.

2) Bool SetTraceData(String[] DataParameter)
- DataParameter 로 넘어오는 Tag UID 정보를 분석하여 OTS DB에 기록한다. 이 함수는 미들웨어가 아닌 OIS에서 호출되며 사전에 인가된 OIS 로부터의 호출이 아닐 경우 호출이 차단될 수 있다.[29)]

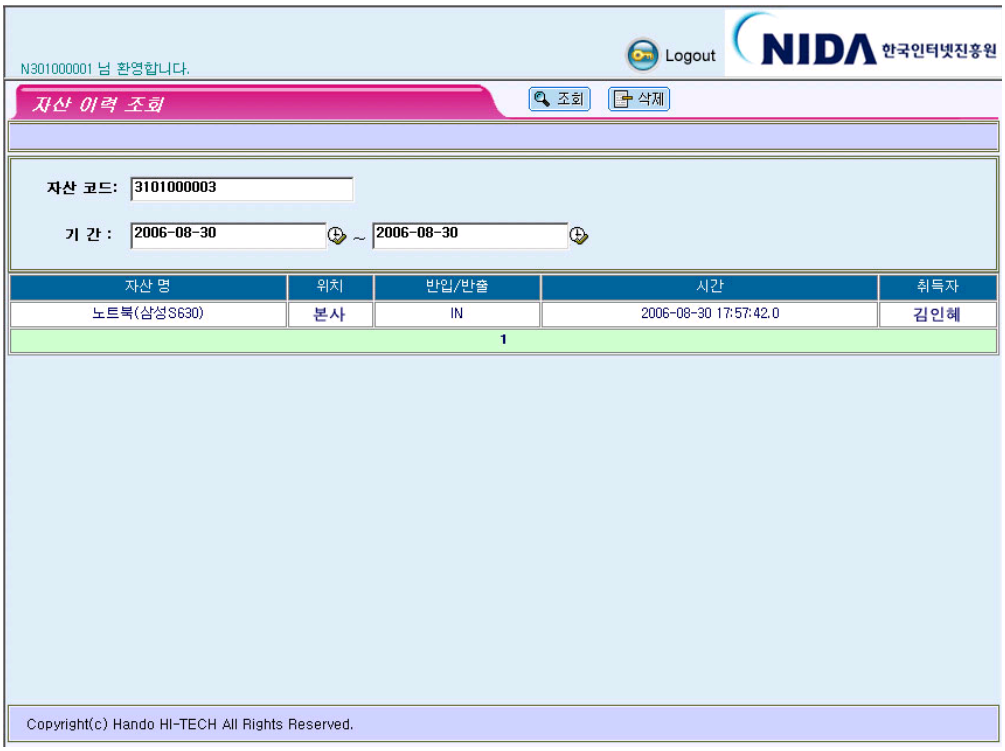
3) int DelTraceData(DataSet DataParameter)
- OTS 정보는 태그의 이동이 있을때마다 누적되며 기록된다. 따라서 데이터가 일정 수준 이상이 되면 제일 오래된 데이터부터 순차적으로 삭제하거나 관리자에 의해 수동으로 삭제하는 기능이 필요하다. 수동 삭제는 이 함수를 호출하여 실행할 수 있으며 관리자는 함수의 인자에 필요한 삭제 요건을 지정한다. 이 함수가 호출되면 OTS는 삭제할 레코드의 OIS URI 리스트를 검색하여 각 OIS 들에 해당 레코드의 이력을 삭제해줄 것을 요청한다.

4.2.5.4. OTS 구동 시나리오

본사와 지사에 설치된 게이트를 통과할 때마다 출입자 및 자산의 이동정보가 OIS 의 이력정보에 기록되며 동시에 OIS는 OTS에게 이력정보가 자신의 DB에서 갱신되었음을 알려준다. [그림 4.14] 은 출입자가 자산을 소지하고 지사 게이트를 통과했을 때 관리자 화면에 기록된 자산의 이력정보를 보여준다. 자산명과 함께 OIS의 위치, 반입/반출 상황, 출입 시간 및 자산을 소지하고 있었던 사람의 이름 등의 정보를 조회할 수 있다.

---

29) 대부분의 OTS 데이터 입출력 API 역시 OIS와 마찬가지로 호출시 별도의 IP Chain 등 인증 절차를 거치며 인증 과정에서 오류가 발생할 경우 보안모듈에 의하여 즉시 호출이 차단된다. 이는 DDoS 등 해킹 공격을 차단하기 위한 것으로 사전에 인가된 클라이언트가 아니면 호출이 불가능하다.



[그림 4.14] 관리자 페이지의 자산 이력 조회 화면

이력 정보는 계속 누적되며 시간순서대로 정렬되므로 이력 조회 화면에서 이제까지 자산이 이동해온 경로를 파악할 수 있다.

[그림 4.14]에서 볼 수 있는 것처럼, 자산번호 3101000003 을 가진 노트북 자산은 2006년 8월 30일 오후에 취득자 김인혜에 의해 본사로 반입되었던 사실을 확인할 수 있다.

4.2.6. 로컬 ODS 설치

4.2.6.1. 개요

Local ODS는 기관 네트워크 내부에서 관리되는 객체정보서버 위치는 자신의 존 파일 리소스 레코드(resource record)에서 관리하며 질의 (Authoritative)에 대해 응답하고, 자신의 존 파일에서 찾지 못하는 경우 상위의 National ODS로 질의 후 다른 Local ODS의 주소값을 획득하여 이 주소의 Local ODS로 질의 (Recursive)를 통해 객체정보서버의 위치를 응답한다.

RFID Directory Service는 현재까지 EPC의 ONS와 uIDcenter의 Ubiquitous ID Resolution Server 가 있다. 그러나 uIDcenter의 Ubiquitous ID Resolution Server의 경우는 UC(Ubiqutous Communicator) 상에서 검색서비스만을 제공하고 있으며, DNS 기반이 아닌 RP(Resolution Protocol)을 사용하여 서비스를 제공한다. 즉 인터넷 기반이지만 외부의 접속을 통한 검색서비스는 불가능하다.
EPC의 ONS는 DNS의 기반 하에 구현되었으나 EPC 코드만 인식이 가능하다. Local ONS는 EPC 코드에 대한 도메인 네임으로 변환된 형태의 질의(DNS query)에 대해 존 파일 내에 있는 해당 도메인의 NAPTR 값인 EPCIS의 주소값을 제공한다.

현재 주류를 이루고 있는 RFID 코드들은 EPCglobal이 제안한 전자상품코드(EPC : Electric Product Code)와 일본 u-ID센터의 ucode, 세계표준화기구(ISO)의 ISO/IEC-15459 코드, ISO/IEC-15963, ISO/IEC-11784 등이 있다.
본 구현에서는 ISO/IEC 15459 기반의 kCode 체계를 사용하였다.
이 절에서는 본 구현에 쓰인 Local ODS 의 설정 및 적용방법을 소개한다.

4.2.6.2. 구성

로컬 ODS는 태그에 삽입된 RFID 코드와 관련된 물품정보가 있는 서버의 위치(URL:Uniform Resource Location)를 알려주는 서비스로서 DNS(Domain Name System) 기술을 기반으로 사용한다.
로컬 ODS 구축을 위해서는 네임서버 기능을 제공할 수 있는 프로그램이 필요하며, 그 중 BIND(Berkeley Internet Name Domain)가 가장 널리 이용된다. BIND 설치 파일은 “http://www.isc.org”에서 다운받을 수 있다.

NIDA에서 제공하는 로컬 ODS 패키지는 BIND를 설치한 뒤 구동하는데 필요한 설정 파일 예제와, 로컬 ODS의 구축 및 테스트에 매뉴얼로 구성되어 있다.

로컬 ODS 패키지의 구성을 간단하게 나타내면 다음과 같다.

구성	비고
bind-config	BIND에 필요한 설정 파일 예제 (named.conf, Zone file 등) - etc (named.conf) - var (named, run, tmp)
구축 매뉴얼	로컬 ODS 구축에 대한 상세 안내 자료

이 부분에서는 로컬 ODS에 관한 기본적인 개념만 다루었으며 이에 대한 자세한 설명은 http://www.ods.or.kr 자료실의 "RFID 로컬 ODS 설치 및 구축 메뉴얼 -v1.2.2" 를 다운받아 참고하기 바란다.

제5장 기대 효과 및 의의

본 구현에서는 RFID 객체검색 서비스의 효용성을 십분 활용한 자산 및 출입관리 시스템을 구축하였다. 구축된 시스템이 갖는 기대효과 및 의의를 간략히 정리하면 [표5-1] 과 같다.

구분	개선효과
자산관리부문	● RFID에 의한 자산관리를 통해 자산현황 및 자산 이동 소유권 이전 등에 대한 정확한 관리 ● 신뢰성 있는 시스템의 확보 자산 이동에 따른 이력관리 ● OIS, OTS를 활용한 검색 기능 강화
출입관리부문[30]	● UHF대의 RFID 태그를 채용하여 태그 인식 거리를 충분히 확보하여 양손이 자유로우므로 자산 운반이 편리한 자동 정보 인식 구현 ● 자산출입 이력 조회 가능
RFID 객체검색 서비스 부문	● 현재 구축/운영중인 RFID ODS 서비스의 효용성 증대 ● OIS, OTS의 확산에 따른 향후 다양한 부가 서비스 및 타 시스템과의 연계성 강화 와 신규사업 기초 마련 ● 현존하는 RFID 최고 기술로 이루어진 시스템 구축의 결정체

[표 5-1] 본 구현의 기대효과 및 의의

이번 시스템으로 인해 무엇보다도 눈에 띄게 향상된 것은 바로 자산관리 부문의 효율성과 생산성 향상을 들 수 있다. 자산의 대량판독과 Read Time이 획기적으로 감소하고 원거리 인식이 가능해짐과 동시에 다중인식의 효율성이 부각되었다. 또한 데이터 수집과 확인작업의 실수를 줄이고, 낭비를 최소화하여 자산관리 업무 작업시간과 인력을 크게 단축시켰다는 점에서 의의를 갖는다.

이와 함께 자산이 반입 반출되는 상황이 실시간으로 모니터링PC 화면에 나타나고 출입 이력이 관리되기 때문에 도난, 분실 등을 사전에 막는 효과도 확인되었다.

30) 출입자 관리 부분이 설계되었으나 실제로 적용하지는 않았으므로 본 지침서에 출입자 관리에 대한 부분은 다루지 않는다.

추가적으로 본 시스템을 통해 이력정보, 자산정보 등 다양한 통계정보가 구성되고 이러한 통계정보들을 다각적으로 분석하여 차후 자산관리정책 수립 시 여러 개선점을 반영할 수 있다는 점이 장점으로 꼽힐 수 있다.

여기에 한걸음 더 나아가서 현재 운영중인 RFID 객체검색 서비스에 대한 실용성을 입증하였다. 즉 위에 언급했던 본 시스템은 수많은 장점과 개선점들은 전부 RFID 객체검색서비스를 기반으로 하고 있으며 각 프로세스마다 RFID 객체검색 서비스의 핵심기술이 반영되어있다. 이는 RFID 객체검색 서비스의 우수성을 보여주는 대표적인 본보기가 될 것이며, 더불어 다양한 응용 시스템과의 연동 가능성을 확인하는 기회가 되었다.

또한, RFID Global Network 구성이라는 유비쿼터스 사회의 최종적인 청사진에 기틀을 마련할수 있도록 국제규격 ISO/IEC 15459 코드 표준안을 따른 kCode 가 적용되었다. 이는 단순한 로컬 네트워크 적용사례에 그치지 않고 RFID Global Network로 확장할 수 있는 기틀을 마련해놓았다는 점에서 큰 의의를 갖는다고 하겠다.

참고문헌

ISO/IEC15961, "Information technology -- Radio frequency identification(RFID) for item management -- Data protocol : application interface," Oct., 2004

ISO/IEC15962, "Information technology -- Radio frequency identification(RFID) for item management -- Data protocol : data encoding rules and logical memory functions," Oct., 2004

ISO/IEC15459-1, “Information technology -- Unique identifiers -- Part 1: Unique identifiers for transport units,” Dec., 2005

ISO/IEC15459-2, "Information technology -- Unique identifiers -- Part 2: Registration procedures," Dec., 2005

ISO/IEC15459-3, "Information technology -- Unique identifiers -- Part 3: Common rules for unique identifiers," Dec., 2005

ISO/IEC15459-4, "Information technology -- Unique identifiers -- Part 4: Unique identifiers for supply chain management," Dec., 2005

ISO/IEC18000-6C, "Information technology -- Radio frequency identification(RFID) for item management -- Part 6C : Parameters for air interface communications at 860 MHz to 960 MHz," Jul., 2005

EPCglobal, "EPC^{TM} Tag Data Standards Version 1.3," Jul., 2005

EPCglobal, "EPC^{TM} Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz - 960 MHz Version 1.1.1," Dec., 2005

E.J. Park “Code Filtering Algorithm for multi-code directory service in global RFID net-work." Ajou Univ, February 2005.

B.J. Choi “A Study on analyze the possibility of DDoS attack and defense mechanism on RFID MDS system." Ajou Univ, August 2005.

한국인터넷진흥원, “.NET Resolver 활용지침서 V1.0,” 2006년 12월

한국인터넷진흥원, “RFID 검색시스템 구축 및 운영지침서 v1.1," 2005년 10월

한국인터넷진흥원, “USN 기반 환경정보 검색시스템 선도연구,” 2005년 12월

한국인터넷진흥원, “로컬 ODS 설치 및 구축 매뉴얼-v1.2,” 2006년 10월

유승화, “유비쿼터스 사회의 RFID," 2005년 3월

용어정리

AFI	Application Family Identifier 다양한 타입의 태그가 대량으로 존재하는 RFID 환경에서 희망하는 RFID 태그에 초점을 맞추어 데이터 처리를 하기위하여 사용. Application Family 영역과 하위 Application Family 영역으로 구성
Air Interface	interrogator와 태그 사이의 전체 통신 회선 (물리계층, 충돌 조정 알고리즘, 명령 및 응답 구조, 데이터 부호화 방법 등 포함)
CRC	Cyclic Redundancy Check message 전송상에 발생하는 error를 검출하는 방법으로, message를 보내는 시스템이 전송되는 message의 bit수를 계산하여 그 결과를 message안에 담아서 보내면, 그 message를 받는 시스템에서 받은 message의 bit수를 같은 방법으로 계산해서 그 결과를 message안에 담겨있는 것과 비교
CRM	Customer Relationship Management 고객관계관리. 기업이 상품이나 서비스를 고객에게 지속적으로 팔기 위하여 고객과의 커뮤니케이션을 최적화하는 마케팅 사고방식
EPC	Electronic Product Code 상품의 식별을 위하여 RFID 태그에 삽입되는 코드 체계
ERP	Enterprise Resource Planning 기업내부의 전사적 자원관리 시스템. 기업의 영업판매, 자재구매 생산관리, 고객관리, 인적자원관리와 재무 및 관리 회계, 원가관리 등의 운영시스템을 하나로 통합하는 것
FQDN	Fully Qualified Domain Name 시스템을 지칭하는 완전한 이름. 호스트 이름과 도메인 이름으로 구성 (예) www(호스트 이름), ods.or.kr(도메인 이름) -> www.ods.or.kr
G2B	Government To Business 국가종합전자조달시스템. 전자정부 구현사업의 핵심사업으로, 구매요청에서부터 대금지불까지의 모든 국가조달행정절차를 온라인화하여, 공공기관 및 조달업체의 모든 사용자에게 G2B의 조달단일창구를 통한 one-stop 조달서비스를 제공
IA	Issuing Agency 발행기관. 물품 관리 적용 용도의 단일 식별자를 할당하기 원하는 단체를 승인

IATA	International Air Transport Association 국제항공운송협회. UN의 항공운송 관련 전문 상임기구인 국제 민간 항공기구(ICAO)의 협의 기구. ICAO와 함께 세계 항공운송 관련 양대 기구 중 하나. 운송회의에서 결정되는 항공운임 및 항공운송조건, 절차와 대리점에 관한 규정 등은 항공사와 대리점에 구속력을 가짐
ISO/IEC JTC1/SC31	International Organization for Standardization/International Electrotechnical Commission Joint Technical Committee 1/Sub-Committee 31 자동식별 및 데이터 획득을 위한 바코드 및 RFID에 대한 국제 표준화 담당 기구
JDK	Java Development Kit 의 약자로 운영 플랫폼에 상관없이 Java 로 된 어플리케이션을 구동시키거나 소스코드를 컴파일 할 수 있게 해주는 키트.
ITU-T	International Telecommunication Union - Telecommunication Standardization Sector 국제전기통신연합. 전기통신 개선, 전파 관련 국제적 협력과 의견 조정 기구
MDM	Multi-code Decoding Module 다양한 RFID 코드를 인지하고 각 코드별 필드를 구분하여 FQDN이나 URN등의 원하는 형태로 변환하는 모듈
NEN	Nederlands Normalisatie-instituut 네덜란드의 표준화 기구. 1916년에 설립되어 네덜란드 국가 표준(NEN)을 제정하는 기관. 국제 표준화 기구(ISO), 국제 전기 표준 회의(IEC) 등 국제 표준 기구와 유럽 표준 위원회(CEN), 유럽 전기 표준 회의(CENELEC) 등 유럽 표준 기구에서 네덜란드의 창구 역할을 담당하며, ISO/IEC JTC 1에서는 기능 표준 특별 그룹(SG-FS)의 간사국 업무를 담당
.Net	Dot Net 이라 읽으며 Microsoft 사가 제안한 Windows 기반의 OOP 설계 및 구현 방식으로 C,C#,VB, Java 등 언어에 상관없이 비슷한 개발 환경을 제공하며 이러한 이점으로 현재 많은 사용자들을 확보하고 있다.
ODS	Object Directory Service RFID 코드를 그 RFID 코드의 정보를 가지고 있는 서버의 URL로 바꾸어주는 분산형 데이터베이스 서비스
OID	Object Identifier 객체 식별자

Passive Tag	수동 태그. RF 필드에 의해 송수신기에 전원이 공급되는 태그
RA	Registration Authority ISO/IEC 15459 등록기관. ISO/IEC JTC 1/SC 31의 회원 중의 하나 또는 ISO/IEC JTC 1/SC 31에 의해 승인된 단체
RFID	Radio Frequency IDentification 전파 신호를 통해 비접촉식으로 사물에 부착된 얇은 평면 형태의 태그를 식별하여 정보를 처리하는 시스템
RFU	Reserved for Future Use 향후에 사용하기 위해 미리 예약해 놓은 데이터 단위
SCM	Supply Chain Management 제조, 물류, 유통업체 등 유통공급망에 참여하는 모든 업체들이 협력을 바탕으로 정보기술(Information Technology)을 활용, 재고를 최적화하고 리드타임을 대폭적으로 감축하여 결과적으로 양질의 상품 및 서비스를 소비자에게 제공함으로써 소비자 가치를 극대화하기 위한 기업의 생존 및 발전전략
TDS	Tag Data Standards EPC의 태그 데이터 표준에 관해 기술한 문서. EPC 각 코딩 체계에 대해 육안판독형 인코딩 및 디코딩 법칙의 관점으로 EPC의 세 가지 표시 방식인 바이너리, 태그 인코딩 URI(Uniform resource identifiers), Pure Identity URI 사이의 변환을 설명
TID	Tag Identifier 태그내에 TID 필드에 기록되어 태그의 유일한 식별 및 객체의 유일 식별을 위해 사용된다. 예를 들면 ISO/IEC 15963 가 TID에 속한다.
UID	Unique Identifier 유일 개체 식별자로서 태그 UII영역에 기록되는 통상적인 16진수 값을 통칭한다. EPC 1 - Gen 1 이상의 프로토콜에서 Read/Write 가능하며 읽기전용인 TID와 구별된다.
UII	Unique Item Identifier 유일 아이템 식별자
ULD	Unit Load Device 종래의 벌크화물을 항공기의 탑재에 적합하도록 설계한 일종의 화물 운송용기로 단위탑재용기인 컨테이너나 팔레트를 말함
URI	Uniform Resource Identifier 텍스트의 한 페이지나 비디오 또는 사운드 클립, 이미지, 동영상 또는 프로그램 등의 콘텐츠 들 중 어느 하나를 인식하기 위한 수단. 가장 보편적인 형태의 URI가 바로 웹페이지 주소 즉, URL인데, 이는 URI의 특별한 형태이자 부분집합임.
URN	Uniform Resource Name URI(Uniform Resource Identifiers)의 한 형태로서 정보의 실제 위치에 관계없이 해당 정보에 접근할 수 있는 것으로 물리적으로 정보가 바뀌더라도 해당 정보에 대한 URN은 일정하게 유지된다.

자료 다운로드 및 오류신고 안내

한국인터넷진흥원에서 발간한 “사례제시를 통한 RFID 적용 본사, 지사간 자산출입관리 시스템 구축 가이드”가 여러분께 많은 도움이 되기를 바랍니다.

본 책자에 소개된 소스 및 책자 내용은 **RFID ODS 홈페이지 (http://www.ods.or.kr)의 자료실을 통하여 배포**하고 있사오니 많은 이용을 부탁드립니다.

또한, NIDA는 여러분의 의견에 항상 귀기울이고 있습니다.
본 문서를 읽으시던 도중 발견한 기술적인 오류, 누락된 부분 및 궁금한 점 등은 “**ods-webmaster@nida.or.kr**”로 연락 주시면 성의껏 답변해 드리겠습니다.
감사합니다.

사례제시를 통한
RFID 적용 본사, 지사간
자산출입관리 시스템 구축 가이드

2006년 12월 인 쇄
2006년 12월 발 행

◎ 발행인 : 송 관 호
◎ 발행처 : 한국인터넷진흥원
◎ 주 소 : 서울 서초구 서초2동 1321-11 KTF 빌딩 3층
Tel : 02-2186-4500 홈페이지 : http://www.nida.or.kr
◎ 인 쇄 : 오리엔스(Tel 02-3391-3861)

<비매품>