

# 서버 설정/개발 가이드

---

21.0.0.1700

**NEXACRO**

이 문서에 잘못된 정보가 있을 수 있습니다. 투비소프트는 이 문서가 제공하는 정보의 정확성을 유지하기 위해 노력하고 특별한 언급 없이 이 문서를 지속적으로 변경하고 보완할 것입니다. 그러나 이 문서에 잘못된 정보가 포함되어 있지 않다는 것을 보증하지 않습니다. 이 문서에 기술된 정보로 인해 발생할 수 있는 직접적인 또는 간접적인 손해, 데이터, 프로그램, 기타 무형의 재산에 관한 손실, 사용 이익의 손실 등에 대해 비록 이와 같은 손해 가능성에 대해 사전에 알고 있었다고 해도 손해 배상 등 기타 책임을 지지 않습니다.

사용자는 본 문서를 구입하거나, 전자 문서로 내려 받거나, 사용을 시작함으로써, 여기에 명시된 내용을 이해하며, 이에 동의하는 것으로 간주합니다.

각 회사의 제품명을 포함한 각 상표는 각 개발사의 등록 상표이며 특허법과 저작권법 등에 의해 보호를 받고 있습니다. 따라서 본 문서에 포함된 기타 모든 제품들과 회사 이름은 각각 해당 소유주의 상표로서 참조용으로만 사용됩니다.

---

발행처 | (주)투비소프트

발행일 | 2024/07/03

주소 | (06083) 서울시 강남구 봉은사로 617 인탑스빌딩 2-5층

전화 | 02-2140-7700

홈페이지 | [www.tobesoft.com](http://www.tobesoft.com)

고객지원센터 | [support.tobesoft.co.kr](http://support.tobesoft.co.kr)

제품기술문의 | 1588-7895 (오전 10시부터 오후 5시까지)

유지보수정책 | 유지보수기간과 범위는 제품 라이선스 계약에 따라 다릅니다.

## 변경 이력

---

버전	변경일	내용
21.0.0.1700	2024-06-28	<a href="#">App Builder 설치</a> 설명 중 지원 사양 관련 항목을 수정했습니다.
21.0.0.100.9	2024-04-01	<a href="#">App Builder</a> 항목에 App Builder 관련 문서 링크를 추가했습니다.
21.0.0.100.8	2024-03-13	Jakarta EE 지원으로 인한 수정
21.0.0.100.7	2024-02-28	<a href="#">App Builder 설치</a> , <a href="#">서버 환경 설정</a> 항목 내 사용하는 버전 정보를 업데이트했습니다.
21.0.0.100.6	2023-10-25	<a href="#">com.nexacro.java.xapi.tx</a> 항목 내 HttpPlatformRequest에서 Type을 지정하지 않은 경우 코드 오류를 수정했습니다.
21.0.0.100.5	2023-07-07	<a href="#">설치 확인</a> 경로 오류를 수정했습니다.
21.0.0.100.4	2022-06-14	기술지원사이트 관련 정보 수정
21.0.0.100.3	2022-04-22	<a href="#">nexacro-xeni</a> 항목 설명 보완
21.0.0.100.2	2022-03-11	<a href="#">넥사크로 X-API (Java)</a> 항목에 Jakarta EE 관련 주의 문구 추가
21.0.0.100.1	2021-12-21	<a href="#">nexacro-xeni</a> 설치 항목에서 제공 버전 표기 오류를 수정했습니다.

# 차례

---

저작권 및 면책조항	2
변경 이력	3
차례	4

## 파트 I. 넥사크로 X-API (Java) 9

1. 설치	10
1.1 설치	10
1.2 라이선스 인증	12
1.3 버전 확인	12
1.4 설치 확인	13
2. com.nexacro.java.xapi.data	14
2.1 시작하기	14
2.2 데이터 구조	16
2.3 데이터 형식	17
2.4 Variable의 단일 데이터 조작	17
2.5 DataSet의 2차원 데이터 참조	19
2.6 DataSet 생성	20
2.7 ColumnHeader의 속성 참조	21
2.8 ColumnHeader를 이용한 DataSet의 열(column) 추가	22
2.9 ColumnHeader를 이용한 DataSet의 데이터 참조	23
2.10 ConstantColumnHeader가 가지는 열(column)의 상수값	25
2.11 DataSet의 원본 데이터와 변경된 데이터	25
2.12 DataSet의 이벤트	27
2.13 DataSet의 선택사항	29
2.14 데이터 설정 또는 반환시의 데이터 형식(type)과 데이터 변환	30
2.15 데이터의 Debug 정보	31
3. com.nexacro.java.xapi.tx	33

3.1	시작하기	33
3.2	데이터 송수신	35
3.3	데이터 송수신의 내부 흐름	36
3.4	Server 상의 HTTP 데이터 통신	37
3.5	Client 상의 HTTP 데이터 통신	38
3.6	데이터 송수신 형식	39
3.7	데이터 프로토콜 형식	40
3.8	데이터 분할 송신	41
3.9	HTTP GET 방식의 데이터	44
3.10	파일 업로드	46
3.11	파일 다운로드	49
3.12	Stream을 이용한 데이터 통신	51
3.13	파일로부터의 데이터 적재와 저장	53
3.14	PlatformData를 XML 문자열로 변환	54
3.15	추가, 변경, 삭제된 데이터 송신	55
3.16	StreamLog를 이용한 송수신 데이터(stream) 저장	56
3.17	localhost 테스트	56
3.18	X-API의 내부 로그 출력하기	57
3.19	데이터에 포함된 null 문자(0x00) 오류	59
3.20	JSP에서 데이터 전송시의 java.lang.IllegalStateException 예외	59

## 파트 II. 넥사크로 X-API (C#) ..... 61

4.	소개 및 구성	62
4.1	구성	62
4.2	라이선스	62
5.	com.nexacro.dotnet.xapi.data (C#)	64
5.1	시작하기	64
5.2	데이터 구조	66
5.3	Variable의 단일 데이터 조작	67
5.4	DataSet의 2차원 데이터 참조	67
5.5	DataSet 생성	69
5.6	ColumnHeader의 속성 참조	70
5.7	ColumnHeader를 이용한 DataSet의 열(column) 추가	71
5.8	ColumnHeader를 이용한 DataSet의 데이터 참조	72
5.9	ConstantColumnHeader가 가지는 열(column)의 상수값	73
5.10	DataSet의 원본 데이터와 변경된 데이터	74
6.	com.nexacro.dotnet.xapi.tx (C#)	76
6.1	시작하기	76

6.2	데이터 송수신	78
6.3	데이터 송수신의 내부 흐름	78
6.4	Server 상의 HTTP 데이터 통신	79
6.5	데이터 송수신 형식	80
6.6	데이터 프로토콜 형식	81
6.7	추가, 변경, 삭제된 데이터 송신	82
<b>7.</b>	<b>NLog config</b>	<b>83</b>
7.1	Web.config에 직접 설정	83
7.2	환경 변수 및 XML 설정 참고	85
<b>파트 III.</b>	<b>nexacro-xeni</b>	<b>86</b>
<b>8.</b>	<b>설치</b>	<b>87</b>
8.1	설치	87
8.2	설치 확인	90
8.3	주요 설정	90
8.4	예제	91
8.4.1	Export	91
8.4.2	Import	91
<b>9.</b>	<b>export 기능</b>	<b>93</b>
9.1	Export 처리	93
9.2	실행 샘플	94
9.2.1	nexacro platform 화면	94
9.2.2	nexacro platform 소스	95
9.2.3	ExcelExportObject Event	96
9.3	오류 대응	96
9.3.1	파일 대화 상자가 열리지 않거나 파일이 깨져서 표시되는 경우	96
<b>10.</b>	<b>import 기능</b>	<b>97</b>
10.1	Import 처리	97
10.2	실행 샘플	98
10.2.1	nexacro platform 화면	98
10.2.2	nexacro platform 소스	99
10.2.3	ExcelImportObject Event	100
<b>11.</b>	<b>nexacro-xeni 확장 인터페이스</b>	<b>101</b>
11.1	개요	101
11.2	메소드	102
11.2.1	(InputStream) loadTargetStream	102
11.2.2	(String) saveImportStream	102
11.2.3	(int) saveExportStream	104

11.2.4 (Dataset) saveExportStream	105
11.3 사용 설정	105
<b>12. nexacro-xeni 확장 인터페이스 작성 예 - DRM</b>	<b>107</b>
12.1 DRM이 적용된 Excel 파일 Import/Export 시나리오	107
12.1.1 확장 인터페이스 상속	108
12.1.2 메소드 구현	108
<b>13. web.xml</b>	<b>113</b>
13.1 Export 관련 설정	113
13.1.1 export file path	113
13.1.2 관리 실행 주기	113
13.1.3 파일 저장 시간	114
13.2 Import 관련 설정	114
13.2.1 import file path	114
13.2.2 파일명 지정	115
13.3 기타 설정	115
13.3.1 파일 관리 실행	115
13.3.2 텍스트 한정자	115
<b>14. xeni.properties</b>	<b>117</b>
14.1 xeni.exportimport.storage	117
14.2 xeni.multipart.proc	118
<b>파트 IV. App Builder</b>	<b>119</b>
<b>15. App Builder 설치</b>	<b>120</b>
15.1 App Builder 설치 전 필요한 작업	120
15.1.1 App Builder 설치 공통	120
15.1.2 Android	121
15.1.3 iOS, macOS	121
15.2 App Builder 설치	121
<b>16. 서버 환경 설정</b>	<b>124</b>
16.1 General	126
16.2 iOS & macOS Configuration	126
16.3 In-house Distribution	127
16.4 Android Configuration	128
<b>17. 디플로이 서버 설정</b>	<b>129</b>
17.1 Deploy Server 설정하기	130
17.2 nexacro Project URL 변경하기	131
17.3 Deploy Server에 리소스 업로드하기	132

<b>18. 운영체제별 서명 설정</b>	<b>134</b>
18.1 Android	137
18.2 iOS	137
18.3 macOS	138
<b>19. 사용자 라이브러리 설정</b>	<b>140</b>
<b>20. 넥사크로 라이브러리 설정</b>	<b>142</b>
<b>21. 사용자 설정</b>	<b>145</b>



**파트 I.**

---

**넥사크로 X-API (Java)**

# 1.

## 설치

넥사크로 X-API는 서버모듈로 제공되며 서버와 클라이언트 간에 데이터 처리를 위해 필요한 기능을 제공합니다. 기본적인 데이터 송신, 수신 기능과 간단하게 필요한 데이터를 가공하는 기능을 제공해 데이터 처리 과정을 단순화합니다.

### 1.1 설치

X-API는 자바 기반 서버모듈로 제공되며 JDK 또는 JRE 1.4 이상 버전이 필요합니다.

배포 파일 내 lib 디렉터리에 포함된 jar 파일을 WAS의 /WEB-INF/lib 디렉토리 또는 정의된 클래스 경로에 복사합니다. 제공되는 jar 파일은 아래와 같습니다.

파일명	필수여부	설명	참조
nexacro-xapi-java-x.x.x.jar nexacro-xapi-java-jakarta_x.x.x.jar	Y	X-API	1.0.12 이후 버전은 Jakarta EE 스펙으로 구현된 WAS에서 사용할 수 있는 X-API를 제공합니다. 파일명에 "_jakarta_"가 포함된 파일

파일명	필수여부	설명	참조
			을 내려 받아 사용하세요. <a href="https://jakarta.ee/compatibility/">https://jakarta.ee/compatibility/</a>
commons-logging-x.x.x.jar	Y	X-API 내부 로깅	<a href="#">Apache Commons Logging</a>
json-simple-x.x.x.jar	Y	JSON 데이터 처리	<a href="#">json-simple</a>



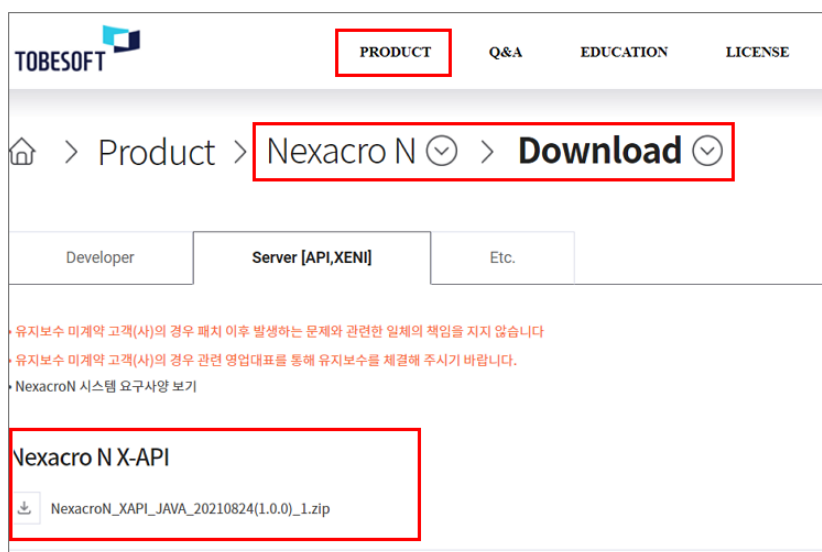
jar 파일명은 버전정보를 포함하고 있습니다. 이전 버전의 파일을 삭제하지 않은 경우 잘못된 버전의 jar 파일이 적용될 수 있습니다. 예를 들어 1.0.0 버전을 1.0.1 버전으로 업데이트하는 경우에는 1.0.0 버전 파일은 삭제해줍니다.

nexacro-xapi-java-1.0.0.jar (삭제)

nexacro-xapi-java-1.0.1.jar (추가)

기술지원 사이트 메뉴 PRODUCT > Nexacro N > Download > Server [API, XENI]에서 jar 파일이 포함된 압축 파일을 내려받을 수 있습니다.

[http://support.tobesoft.co.kr/Support/?menu=Download\\_N](http://support.tobesoft.co.kr/Support/?menu=Download_N)



## 1.2 라이선스 인증

라이선스 파일(NexacroN\_server\_license.xml)을 jar 파일과 같은 디렉토리 또는 정의된 클래스 경로에 복사합니다.



1개 이상의 라이선스 파일이 다른 경로로 복사된 경우에는 jar 파일과 같은 디렉토리에 있는 라이선스 파일을 먼저 적용합니다.



서버 환경 설정에 따라 jar 파일과 라이선스 파일을 복사하고 WAS 재시작이 필요할 수도 있습니다.

## 1.3 버전 확인

jar 파일명이 "1.x.x" 처럼 3자리로 작성된 경우에는 파일명에 포함된 "1.x.x"가 버전입니다.

jar 파일명이 "1.0"으로 작성된 경우 X-API 버전은 다음 명령어로 "Implementation-Version" 항목을 확인합니다.

```
java -jar nexacro-xapi-java-1.0.0.jar
```

```
--- Manifest ---
```

```
Manifest-Version: 1.0
```

```
Built-By: Server Team
```

```
Created-By: 1.5.0_22-b03 (Sun Microsystems Inc.)
```

```
Ant-Version: Apache Ant 1.7.1
```

```
Main-Class: com.nexacro.java.xapi.util.JarInfo
```

```
Built-Date: August 19 2021
```

```
Name: com.nexacro.java.xapi
```

```
Specification-Title: X-API
```

```
Implementation-Title: X-API
```

```
Specification-Version: 1.0
```

```
Specification-Vendor: TOBESOFT CO., LTD.
```

```
Implementation-Vendor-Id: com.nexacro
```

```
Implementation-Version: 1.0.0.7805
```

```
Implementation-Vendor: TOBESOFT CO., LTD.
```

## 1.4 설치 확인

X-API 정상 설치 여부는 간단하게 아래 샘플 JSP 파일을 작성해 확인할 수 있습니다.

```
<%@ page contentType="text/html; charset=UTF-8" %>

<html>
  <head>
    <title>JarInfo</title>
    <style>
      * { font-family: Verdana }
    </style>
  </head>
  <body>
    <pre>
<%
  new com.nexacro.java.xapi.util.JarInfo().info(out);
%>
    </pre>
  </body>
</html>
```

정상적으로 설치가 되었다면 X-API 버전 정보와 제품 라이선스 정보가 출력됩니다.

## 2.

---

# com.nexacro.java.xapi.data

X-API의 데이터 구조를 정의합니다.

클라이언트와 서버간의 송수신하는 데이터는 단일 데이터도 존재하고, DB의 Table과 유사한 2차원적인 데이터도 존재합니다. 이러한 데이터를 송수신 또는 조작하기 위한 데이터 구조를 정의합니다. 주요 클래스는 PlatformData, DataSet와 Variable 등입니다.

## 2.1 시작하기

다음은 X-API의 데이터를 참조하는 간단한 예제입니다.

데이터 참조

```
1 PlatformData department = ...;
2
3 // VariableList 참조
4 VariableList varList = department.getVariableList();
5
6 // VariableList으로부터 값 참조
7 String name = varList.getString("name");
8 String location = varList.getString("location");
9 int number = varList.getInt("number");
10
11 // ...
12
13 // DataSet 참조
14 DataSet employees = department.getDataSet("employees");
15
16 // DataSet의 행(row)수만큼 순환
17 for (int i = 0; i < employees.getRowCount(); i++) {
```

```

18 // DataSet의 데이터 참조
19 int id = employees.getInt(i, "id");
20 String firstName = employees.getString(i, "firstName");
21 String lastName = employees.getString(i, "lastName");
22 boolean manager = employees.getBoolean(i, "manager");
23
24 // ...
25 }

```

다음은 X-API의 데이터를 생성하는 간단한 예제입니다.

데이터 생성

```

1 // PlatformData 생성
2 PlatformData department = new PlatformData();
3
4 // VariableList 참조
5 VariableList varList = department.getVariableList();
6
7 // VariableList에 값 추가
8 varList.add("name", "R&D Center");
9 varList.add("location", "222 Jamsil-Dong, Songpa-Ku, Seoul");
10 varList.add("number", 99);
11
12 // DataSet 생성
13 DataSet employees = new DataSet("employees");
14
15 // 열(column) 추가
16 employees.addColumn("id", DataTypes.INT);
17 employees.addColumn("firstName", DataTypes.STRING, 16);
18 employees.addColumn("lastName", DataTypes.STRING, 8);
19 employees.addColumn("manager", DataTypes.BOOLEAN);
20
21 // 행(row) 추가
22 int row = employees.newRow();
23
24 // 추가된 행(row)의 데이터 설정
25 employees.set(row, "id", 0);
26 employees.set(row, "firstName", "John");
27 employees.set(row, "lastName", "Jones");
28 employees.set(row, "manager", false);
29

```

```

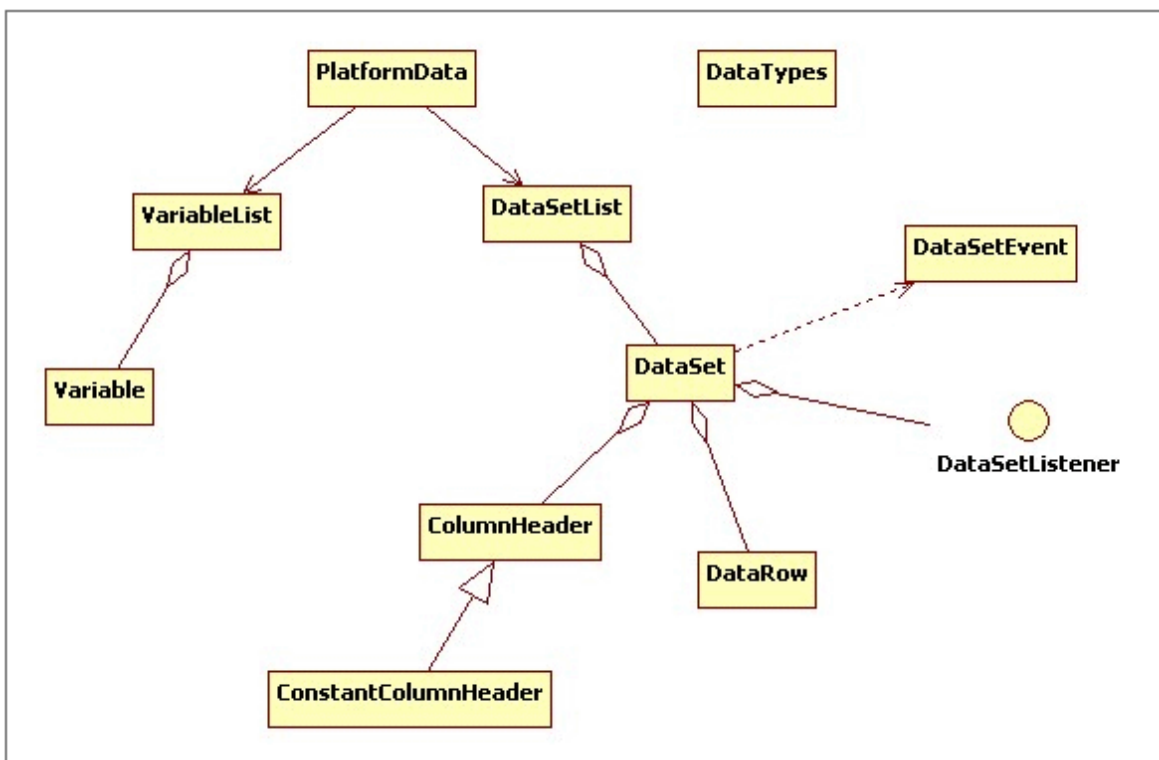
30 // 행(row) 추가
31 row = employees.newRow();
32
33 // 추가된 행(row)의 데이터 설정
34 employees.set(row, "id", 1);
35 employees.set(row, "firstName", "Tom");
36 employees.set(row, "lastName", "Glover");
37 employees.set(row, "manager", true);
38
39 // DataSet을 PlatformData에 추가
40 department.addDataSet(employees);

```

## 2.2 데이터 구조

데이터는 크게 단일 데이터와 2차원 데이터로 구분합니다. 단일 데이터는 데이터를 구분할 수 있는 식별자(name)와 값(value)을 가지고 있으며, 이는 VariableList에 저장됩니다. 2차원 데이터를 저장하는 DataSet은 열(column)과 행(row)으로 구성되어 있으며, DataSetList를 통해 저장 또는 참조됩니다.

VariableList와 DataSetList를 가지고 있는 PlatformData는 데이터 구조의 최상위에 위치하고 있으며, 데이터 이동과 데이터 송수신의 기본 단위로 사용되어집니다.





## 2.3 데이터 형식

X-API에서 지원하는 데이터의 형식(type)은 DataTypes에 정의되어 있습니다.

데이터의 형식은 넥사크로와 X-API간의 약간의 차이가 존재하고, X-API가 약간 더 세분화되어 있습니다. 그러나, 기본적으로 넥사크로와 X-API간의 통신중의 데이터 손실은 발생하지 않습니다.

NRE	Javascript	X-API	Java	설명
STRING	String	DataTypes.STRING	String	문자열
INT	Int	DataTypes.INT	int	4 byte 정수
INT	Int	DataTypes.BOOLEAN	boolean	참 또는 거짓 (1 또는 0)
BIGDECIMAL	BigDecimal	DataTypes.LONG	long	8 byte 정수
FLOAT	BigDecimal	DataTypes.FLOAT	float	4 byte 실수
FLOAT	BigDecimal	DataTypes.DOUBLE	double	8 byte 실수
BIGDECIMAL	BigDecimal	DataTypes.BIG_DECIMAL	java.math.BigDecimal	-
DATE	Date	DataTypes.DATE	java.util.Date	일자 (yyyyMMdd)
TIME	Date	DataTypes.TIME	java.util.Date	시간 (HHmmssSSS)
DATETIME	Date	DataTypes.DATE_TIME	java.util.Date	일자와 시간 (yyyyMMddHHmmssSSS)
BLOB	미지원	DataTypes.BLOB	byte[]	byte 배열

## 2.4 Variable의 단일 데이터 조작

Variable은 데이터를 저장하는 변수를 의미하며, 이는 식별자(name)와 값(value)으로 구성된다. 값(value)은 데이터의 형식(type)에 따라 변환된 후 저장됩니다.

Variable 생성과 데이터 설정은 3가지 방식을 지원합니다.

- Variable(name, type, value)의 생성자 호출
- Variable(name) 또는 Variable(name, type)의 생성자 호출 후 set(value) 메소드 호출
- Variable.createVariable(name, value)의 정적 메소드 호출을 통한 생성

VariableList의 Variable 추가는 Variable 생성 후 add(var) 메소드를 통하여 추가합니다. 또한, Variable을 생성하지 않고 add(name, value) 메소드를 통하여 직접 값으로 추가할 수 있습니다.

단일 데이터(Variable) 추가

```

1  // PlatformData 생성
2  PlatformData department = new PlatformData();
3
4  // VariableList 참조
5  VariableList varList = department.getVariableList();
6
7  // VariableList에 값을 직접 추가
8  varList.add("name", "R&D Center");
9
10 // Variable을 생성한 후 VariableList에 값을 추가
11 Variable location = new Variable("location");
12 location.set("222 Jamsil-Dong, Songpa-Ku, Seoul");
13 varList.add(location);
14
15 // VariableList에 int 형의 값을 직접 추가
16 varList.add("number", 99);

```

Variable에 저장된 데이터는 getObject()과 getString()등의 필요한 데이터 형식에 따른 메소드를 이용하여 값을 참조할 수 있습니다.

주의할 점은 원 데이터의 형식과 다른 데이터 형식으로 반환을 요청한 경우 데이터의 변형이 발생할 수 있습니다.

VariableList에 저장된 Variable은 get(name) 메소드를 통해 참조하며, getObject(name)과 getString(name)등의 메소드를 통하여 직접 값을 참조할 수도 있습니다.

#### 단일 데이터(Variable) 참조

```

1  PlatformData department = ...;
2
3  // VariableList 참조
4  VariableList varList = department.getVariableList();
5
6  // VariableList으로부터 값을 직접 참조
7  String name = varList.getString("name");
8
9  // Variable을 통한 값을 참조
10 Variable locationVar = department.getVariable("location");
11 String location = locationVar.getString();
12
13 // VariableList으로부터 int 형의 값을 직접 참조
14 int number = varList.getInt("number");

```

## 2.5 DataSet의 2차원 데이터 참조

DataSet은 열(column)과 행(row)으로 구성되며, 2차원 데이터를 저장합니다. 구조는 DB의 Table과 유사하며, 열(column)에 대한 정보는 ColumnHeader에 의해 저장되고, 데이터는 내부 클래스인 DataRow에 의해 행(row) 단위로 저장됩니다.

DataSet에 저장된 데이터는 행(row)의 위치(index)와 열(column)의 이름(name) 또는 위치(index)로 참조하며, Variable과 동일한 방식으로 getObject(rowIndex, columnIndex)과 getString(rowIndex, columnIndex)등의 필요한 데이터 형식에 따른 메소드를 이용하여 값을 참조할 수 있습니다.

마찬가지로 주의할 점은 원 데이터의 형식과 다른 데이터 형식으로 반환을 요청한 경우 데이터의 변형이 발생할 수 있습니다.

식별자(name)를 이용한 DataSet의 데이터 참조

```

1 PlatformData department = ...;
2
3 // DataSet을 식별자(name)를 이용하여 참조
4 DataSet employees = department.getDataSet("employees");
5
6 // DataSet의 행(row)수만큼 순환
7 for (int i = 0; i < employees.getRowCount(); i++) {
8     // DataSet의 데이터를 식별자(name)를 통하여 참조
9     Object name = employees.getObject(i, "name");
10    String jobTitle = employees.getString(i, "jobTitle");
11    int number = employees.getInt(i, "number");
12    boolean manager = employees.getBoolean(i, "manager");
13    // ...
14 }
```

위치(index)를 이용한 DataSet의 데이터 참조

```

1 PlatformData department = ...;
2
3 // DataSet을 위치(index)를 이용하여 참조
4 DataSet employees = department.getDataSet(0);
5
6 // DataSet의 행(row)수만큼 순환
7 for (int i = 0; i < employees.getRowCount(); i++) {
8     // DataSet의 데이터를 열(column)의 위치(index)를 통하여 참조
9     Object name = employees.getObject(i, 0);
10    String jobTitle = employees.getString(i, 1);
```

```

11     int number = employees.getInt(i, 2);
12     boolean manager = employees.getBoolean(i, 3);
13
14     // ...
15 }

```

## 2.6 DataSet 생성

DataSet 생성은 다음과 같은 과정으로 이루어집니다.

1. DataSet 생성
2. 열(column) 추가
3. 행(row) 추가
4. 데이터 설정

DataSet 생성과 데이터 추가

```

1 PlatformData department = new PlatformData();
2
3 // DataSet 생성
4 DataSet employees = new DataSet("employees");
5
6 // 열(column) 추가
7 employees.addColumn("name", DataTypes.STRING, 8);
8 employees.addColumn("jobTitle", DataTypes.STRING, 16);
9 employees.addColumn("number", DataTypes.INT);
10 employees.addColumn("manager", DataTypes.BOOLEAN);
11
12 // 행(row) 추가
13 int row = employees.newRow();
14
15 // 데이터 설정
16 employees.set(row, "name", "John Jones");
17 employees.set(row, "jobTitle", "developer");
18 employees.set(row, "number", 1234);
19 employees.set(row, "manager", false);
20
21 // 행(row) 추가

```

```

22 row = employees.newRow();
23
24 // 데이터 설정
25 employees.set(row, "name", "Tom Glover");
26 employees.set(row, "jobTitle", "manager");
27 employees.set(row, "number", 9876);
28 employees.set(row, "manager", true);
29
30 // DataSet을 PlatformData에 추가
31 department.addDataSet(employees);
32
33 // ...

```

## 2.7 ColumnHeader의 속성 참조

DataSet의 열(column)에 대한 정보는 ColumnHeader에 의해 저장되며, 열(column)에 대한 정보는 다음과 같습니다.

속성명	변수명	데이터 형식	유효한 값
식별자	name	String	null과 ""를 제외한 DataSet 내에서 유일한 문자열
열(column)의 형식	type	int	일반적인 열(TYPE_NORMAL)과 상수값을 가진 열(TYPE_CONSTANT)
데이터 형식	dataType	int	DataTypes에 정의된 상수 참조
데이터 크기	dataSize	int	정수값
값	value	Object	ConstantColumnHeader 내에서만 유효

ColumnHeader의 속성 참조

```

1 PlatformData department = ...;
2
3 // DataSet 참조
4 DataSet employees = department.getDataSet("employees");
5
6 // DataSet의 열(column)수만큼 순환
7 for (int i = 0; i < employees.getColumnCount(); i++) {

```

```

8    // DataSet으로부터 ColumnHeader 참조
9    ColumnHeader columnHeader = employees.getColumn(i);
10
11   // 열(column)의 속성 참조
12   String name = columnHeader.getName();
13   int type = columnHeader.getType();
14   int dataType = columnHeader.getDataType();
15   int dataSize = columnHeader.getDataSize();
16   boolean isConstant = columnHeader.isConstant();
17
18   // 상수값을 가진 ColumnHeader인 경우
19   Object value = null;
20   if (isConstant) {
21       value = ((ConstantColumnHeader) columnHeader).getValue();
22   }
23
24   // ...
25 }

```

## 2.8 ColumnHeader를 이용한 DataSet의 열(column) 추가

DataSet에 열(column)을 추가하는 방법은 addColumn(name, dataType, dataSize)를 통하거나, 직접 ColumnHeader를 생성하여 추가할 수 있습니다.

ColumnHeader를 이용한 DataSet의 열(column) 추가

```

1  // DataSet 생성
2  DataSet employees = new DataSet("employees");
3
4  // DataSet에 열(column) 추가
5  employees.addColumn(new ColumnHeader("name", DataTypes.STRING, 8));
6  employees.addColumn(new ColumnHeader("jobTitle", DataTypes.STRING, 16));
7  employees.addColumn(new ColumnHeader("number", DataTypes.INT));
8  employees.addColumn(new ColumnHeader("manager", DataTypes.BOOLEAN));
9
10 // 행(row) 추가

```

```

11  int row = employees.newRow();
12
13  // 추가된 행(row)의 데이터 설정
14  employees.set(row, "name", "John Jones");
15  employees.set(row, "jobTitle", "developer");
16  employees.set(row, "number", 1234);
17  employees.set(row, "manager", false);
18
19  // 행(row) 추가
20  row = employees.newRow();
21
22  // 추가된 행(row)의 데이터 설정
23  employees.set(row, "name", "Tom Glover");
24  employees.set(row, "jobTitle", "manager");
25  employees.set(row, "number", 9876);
26  employees.set(row, "manager", true);

```

## 2.9 ColumnHeader를 이용한 DataSet의 데이터 참조

때로는 DataSet의 열(column)에 대한 정보를 참조해야 하는 경우도 있습니다. 예를 들어, 각각의 열(column)에 대한 정보를 알지 못하는 경우나, DataSet의 데이터를 공통적으로 처리하여야 하는 경우일 것입니다.

DataSet의 getColumn(index)를 호출하여 열(column)의 갯수만큼 ColumnHeader를 참조하고, ColumnHeader 으로부터 식별자(name), 데이터 형식(dataType), 데이터 크기(dataSize) 등을 참조하여, 이에 따라 원하는 동작을 수행하면 될 것입니다.

ColumnHeader를 이용한 DataSet의 데이터 참조

```

1  PlatformData department = ...;
2
3  // DataSet을 식별자(id)를 이용하여 참조
4  DataSet employees = department.getDataSet("employees");
5
6  // DataSet의 행(row)수만큼 순환
7  for (int i = 0; i < employees.getRowCount(); i++) {
8      // DataSet의 열(column)수만큼 순환
9      for (int j = 0; j < employees.getColumnCount(); j++) {
10         // DataSet으로부터 ColumnHeader 참조
11         ColumnHeader columnHeader = employees.getColumn(j);

```

```
12     // 열(column)의 식별자(name) 참조
13     String name = columnHeader.getName();
14
15     // 데이터의 형식(dataType)에 따른 구분
16     switch (columnHeader.getDataType()) {
17     case DataTypes.STRING:
18         String str = employees.getString(i, name);
19
20         // ...
21
22         break;
23     case DataTypes.INT:
24         int n = employees.getInt(i, name);
25
26         // ...
27
28         break;
29     case DataTypes.BOOLEAN:
30         boolean bool = employees.getBoolean(i, name);
31
32         // ...
33
34         break;
35     default:
36         Object obj = employees.getObject(i, name);
37
38         // ...
39
40         break;
41     }
42 }
43 }
```



## 2.10 ConstantColumnHeader가 가지는 열(column)의 상수값

ConstantColumnHeader는 상수값을 가진 열(column)을 의미합니다.

즉, DataSet에 addConstantColumn(name, value)을 호출하여 열(column)을 추가하거나, ConstantColumnHeader를 생성하여 추가한다면, 해당 열(column)의 값은 행(row)의 위치(index)와 관계없이 일정한 상수값을 가지게 됩니다.

DataSet의 상수값을 가진 열(column) 추가

```
1 // DataSet 생성
2 DataSet employees = new DataSet("employees");
3
4 // DataSet에 일반 열(column) 추가
5 employees.addColumn("name", DataTypes.STRING, 8);
6 employees.addColumn("jobTitle", DataTypes.STRING, 16);
7 // DataSet에 상수값을 가진 열(column) 추가
8 employees.addConstantColumn("city", "Seoul");
9 employees.addColumn(new ConstantColumnHeader("company", "Tobesoft"));
```

## 2.11 DataSet의 원본 데이터와 변경된 데이터

DataSet은 데이터가 추가, 변경, 삭제된 경우 변경된 상태와 변경 이전의 원본 데이터를 저장합니다. 데이터가 변경되는 경우에는 원본 데이터를 별도로 저장하고, 현재 데이터를 변경하며, 삭제되는 경우에는 현재 데이터에서는 삭제되지만, 별도의 삭제된 데이터에 저장됩니다. 변경된 상태는 행(row) 단위로 저장되며, DataSet의 getRowType(index)를 호출하여 현재의 상태를 확인할 수 있습니다.

상수값	설 명
DataSet.ROW_TYPE_NORMAL	일반적인 행(row)
DataSet.ROW_TYPE_INSERTED	추가된 행(row)
DataSet.ROW_TYPE_UPDATED	변경된 행(row), 원본 데이터 존재할 수 있음
DataSet.ROW_TYPE_DELETED	삭제된 행(row), 다른 데이터와는 별도로 저장됨

저장 여부는 DataSet의 startStoreDataChanges()를 호출하여 활성화시키고, stopStoreDataChanges()를 통하여 저장을 중지하며, startStoreDataChanges()를 호출하는 시점의 데이터를 기준 데이터로 설정됩니다.

startStoreDataChanges()가 호출되면 이전에 저장된 원본 또는 삭제된 데이터는 삭제되므로, 데이터 유지가 필요한 경우 startStoreDataChanges(true)를 호출하여야 합니다. 반대로 stopStoreDataChanges()가 호출되면 이전에 저장된 원본 또는 삭제된 데이터는 보존되므로, 보존을 원하지 않는 경우 stopStoreDataChanges(false)를 호출합니다.

또한, 각각의 상태에 따른 데이터는 다음 메소드를 통하여 참조할 수 있습니다.

- 현재 데이터 : getObject(rowIndex, columnIndex) 등
- 변경 이전의 원본 데이터 : getSavedData(rowIndex, columnIndex) 등
- 삭제된 데이터 : getRemovedData(rowIndex, columnIndex) 등

주의할 점은 DataSet의 기본 설정은 변경되는 상태와 데이터를 저장하는 것입니다. 즉, DataSet의 생성과 동시에 startStoreDataChanges()가 자동적으로 호출되어 있는 것입니다.

이것이 의미하는 것은 사용자가 startStoreDataChanges()를 별도로 호출되지 않는 이상 DataSet에 저장되는 모든 데이터의 상태는 ROW\_TYPE\_INSERTED일 것입니다. 예를 들어, DataSet을 생성한 후에 데이터를 추가하고, 추가된 데이터를 다시 변경하더라도 데이터의 상태는 여전히 ROW\_TYPE\_INSERTED입니다. 이유는 DataSet의 기존 데이터는 생성한 직후가 되기 때문에, 즉 데이터가 없는 상태를 기준으로 본다면 데이터는 여전히 추가된 상태인 것입니다. 마찬가지로, 데이터를 추가한 후에 삭제하더라도 데이터의 상태는 ROW\_TYPE\_DELETED가 아니고, 데이터가 없는 상태를 기준으로 본다면 어떤 변경도 없었던 것입니다.

따라서, DataSet의 추가, 변경, 삭제된 상태와 데이터가 필요한 경우 적절한 시점에 startStoreDataChanges()가 호출되어야 합니다. 물론 DataSet을 생성하지 않고, 통신을 통해 전달받고, 이 시점을 기준 데이터로 본다면 굳이 호출하지 않아도 됩니다.

추가, 변경, 삭제된 DataSet의 행(row)

```

1 PlatformData department = ...;
2
3 // DataSet을 식별자(id)를 이용하여 참조
4 DataSet employees = department.getDataSet("employees");
5
6 // 변경 정보 저장 시작
7 employees.startStoreDataChanges();
8
9 // DataSet의 데이터 추가, 변경, 삭제 수행
10 ...
11
12 // 변경 정보 저장 중지
13 employees.stopStoreDataChanges();
14
15 // DataSet의 행(row)수만큼 순환
16 for (int i = 0; i < employees.getRowCount(); i++) {
17     // 행(row)의 상태 참조

```

```

18  int rowType = employees.getRowType(i);
19
20  if (rowType == DataSet.ROW_TYPE_NORMAL) {
21      // 일반적인 행(row)인 경우
22      Object name = employees.getObject(i, "name");
23
24      // ...
25  } else if (rowType == DataSet.ROW_TYPE_INSERTED) {
26      // 추가된 행(row)인 경우
27      Object name = employees.getObject(i, "name");
28
29      // ...
30  } else if (rowType == DataSet.ROW_TYPE_UPDATED) {
31      // 변경된 행(row)인 경우
32      Object name = employees.getObject(i, "name");
33      Object savedName = employees.getSavedData(i, "name");
34
35      // ...
36  } else {
37      // 발생 않음
38  }
39 }
40
41 for (int i = 0; i < employees.getRemovedRowCount(); i++) {
42     // 삭제된 행(row)인 경우
43     Object removedName = employees.getRemovedData(i, "name");
44
45     // ...
46 }

```

## 2.12 DataSet의 이벤트

DataSet은 구조가 변경되거나 데이터가 변경된 경우 DataSetEvent를 발생시킵니다. 만약에 DataSet 변경에 따른 특정 행동 또는 처리가 필요한 경우 DataSetListener를 구현하여 DataSet에 등록하여야 합니다.

이벤트가 호출되는 시점은 다음과 같이 4가지 경우입니다.

호출되는 메소드	설 명
DataSetListener.structureChanged	열(column)의 추가 등 구조가 변경된 경우
DataSetListener.rowInserted	행(row)이 추가된 경우
DataSetListener.dataUpdated	데이터가 변경된 경우
DataSetListener.rowRemoved	행(row)이 삭제된 경우

#### DataSet에 DataSetListener 등록

```

1  DataSet employees = new DataSet("employees");
2
3  // DataSet에 DataSetListener 등록
4  DataSetListener listener = new DataSetEventHandler();
5  employees.addDataSetListener(listener);
6
7  // 열(column) 추가, DataSetListener의 structureChanged 호출됨
8  employees.addColumn("name", DataTypes.STRING, 8);
9
10 // 행(row) 추가, DataSetListener의 rowInserted 호출됨
11 int row = employees.newRow();
12
13 // 데이터 설정, DataSetListener의 dataUpdated 호출됨
14 employees.set(row, "name", "John Jones");
15
16 // ...

```

#### DataSetListener의 구현체

```

1  class DataSetEventHandler implements DataSetListener {
2
3      public void structureChanged(DataSetEvent e) {
4          // 열(column)의 삭제등 구조가 변경된 경우
5          DataSet ds = (DataSet) e.getSource();
6
7          // ...
8      }
9
10     public void dataUpdated(DataSetEvent e) {
11         // 데이터가 변경된 경우
12         DataSet ds = (DataSet) e.getSource();
13         int firstRow = e.getFirstRow();
14         int lastRow = e.getLastRow();
15         int column = e.getColumn();

```

```

16
17     // ...
18 }
19
20 public void rowInserted(DataSetEvent e) {
21     // 행(row)이 추가된 경우
22     DataSet ds = (DataSet) e.getSource();
23     int firstRow = e.getFirstRow();
24     int lastRow = e.getLastRow();
25
26     // ...
27 }
28
29 public void rowRemoved(DataSetEvent e) {
30     // 행(row)이 삭제된 경우
31     DataSet ds = (DataSet) e.getSource();
32     int firstRow = e.getFirstRow();
33     int lastRow = e.getLastRow();
34
35     // ...
36 }
37 }

```

## 2.13 DataSet의 선택사항

DataSet은 몇가지 선택사항을 가지고 있으며, 필요에 따라 선택사항의 값을 변경합니다. 다음은 DataSet의 선택사항들입니다.

선택사항	기본값	설 명
isStoreDataChanges	true	데이터 변경 정보에 대한 저장 여부, 자세한 사항은 11. DataSet의 원본 데이터와 변경된 데이터 참조
isCheckingGetterDataIndex	false	데이터 반환시 행(row) 또는 열(column)의 위치(index)에 대한 검사 여부
isCheckingSetterDataIndex	true	데이터 설정시 행(row) 또는 열(column)의 위치(index)에 대한 검사 여부
changeStructureWithData	false	데이터가 존재하는 경우 구조 변경의 가능 여부
isConvertingToDataType	true	데이터 설정시 열(column)의 데이터 형식(type)으로의 변환 여부

## 2.14 데이터 설정 또는 반환시의 데이터 형식(type)과 데이터 변환

Variable과 DataSet은 데이터와 더불어 데이터의 형식(type)을 가지고 있습니다. 데이터의 형식(type)에 대한 자세한 내용은 [데이터 형식](#)을 참조해주세요. 만약에 설정된 데이터의 형식(type)과 다른 형식(type)의 데이터를 저장하거나, 저장된 데이터와 다른 형식(type)의 데이터로 반환을 요구하는 경우 데이터의 변환이 발생하게 됩니다.

예를 들어, Variable의 데이터 형식(type)은 String이지만, 숫자 123을 저장하는 경우 숫자 123은 문자열 "123"으로 변환되어 저장되어야 합니다. 또는 문자열 "123"을 저장하고 있는 상태에서 int 형식의 데이터를 요구하는 경우 문자열 "123"은 숫자 123으로 변환되어 반환되어야 합니다.

단, DataSet의 isConvertingToDataType가 false인 경우 데이터 설정시 위와 같은 변환은 이루어지지 않습니다.

Variable과 DataSet은 위와 같은 데이터 변환을 VariableDataConverter과 DataSetDataConverter에 위임합니다. Variable에 기본적으로 설정된 VariableDataConverter는 DefaultVariableDataConverter이며, DataSet에 기본적으로 설정된 DataSetDataConverter는 DefaultDataSetDataConverter입니다.

만약에 기본적으로 설정된 DataSet의 DataSetDataConverter를 이용하지 않고, 다른 방식의 변환을 원하는 경우 DataSetDataConverter를 직접 구현하거나, 또는 DefaultDataSetDataConverter를 상속받아 원하는 부분만 재정의하여 DataSet의 setDataConverter(DataSetDataConverter)을 호출하여 설정하면 됩니다. DataSet의 메소드와 그 내부에서 호출되는 DataSetDataConverter의 메소드와의 관계는 DataSetDataConverter를 참조해주세요. Variable도 DataSet과 동일한 방식으로 설정하면 됩니다.

DataSet의 사용자 정의 DataSetDataConverter 등록

```
1 DataSet ds = new DataSet("ds");
2 // "yyyy-MM-dd" 형식의 문자열도 Date로 변환해주는 DataSetDataConverter 설정
3 ds.setDataConverter(new UserDataConverter());
4
5 // 열(column)과 행(row) 추가
6 ds.addColumn("date", DataTypes.DATE_TIME, 256);
7 ds.newRow();
8
9 // "yyyy-MM-dd" 형식의 데이터 설정
10 ds.set(0, "date", "2008-12-25");
```

사용자 정의 DataSetDataConverter

```
1 class UserDataConverter extends DefaultDataSetDataConverter {
2
3     public Object convert(DataSet ds, int row, int column, String value, int type) {
4         return convert(ds, row, column, value, type, null);
5     }
6 }
```

```

5    }
6
7    public Object convert(DataSet ds, int row, int column, String value, int type, String
charset) {
8        // 데이터의 형식(type)이 DataTypes.DATE_TIME 형식인 경우
9        // 기본적으로 지원하지 않는 "yyyy-MM-dd" 형식의 문자열도 Date로 변환
10       if (type == DataTypes.DATE_TIME) {
11           int len = (value == null) ? -1 : value.length();
12
13           if (len == 10) {
14               try {
15                   return new SimpleDateFormat("yyyy-MM-dd").parse(value);
16               } catch (ParseException ex) {
17                   ;
18               }
19           }
20       }
21
22       return super.convert(ds, row, column, value, type, charset);
23   }
24 }

```

## 2.15 데이터의 Debug 정보

개발시에는 데이터의 내용을 파악하기 위해 출력이 필요한 경우가 발생합니다. Debug를 위하여 데이터의 내용을 출력하기 위해서는 Debugger를 이용하여 출력합니다.

다음은 Debugger를 이용하여 데이터를 출력하는 예제와 출력 결과입니다.

출력 결과 예

```

1 PlatformData department = ...;
2
3 // Debugger 생성
4 Debugger debugger = new Debugger();
5
6 // Debug 정보 출력
7 log(debugger.detail(department));

```

```

variable=[
index=0 (name, string, "R&D Center")
, index=1 (location, string, "222 Jamsil-Dong, Songpa-Ku, Seoul")
, index=2 (number, int, "99")
]
----- index=000 -----
name=employees, alias=employees, columnCount=4, rowCount=2, charset=null, isStoreDataChanges=
true
, column=[
index=0 (id, int, 4)
, index=1 (firstName, string, 16)
, index=2 (lastName, string, 8)
, index=3 (manager, bool, 2)
]
, row=[
index=0 inserted ("0", "John", "Jones", "false")
, index=1 inserted ("1", "Tom", "Glover", "true")
]

```



## 3.

---

# com.nexacro.java.xapi.tx

X-API의 데이터 통신을 수행합니다.

넥사크로 또는 PlatformData를 이용하는 모든 클라이언트와 데이터 송수신을 수행합니다. 데이터 통신의 대부분의 경우 HTTP 상에서 수행되며, XML 등의 형식으로 변환된 후 송수신됩니다. 주요 클래스는 HttpPlatformRequest, HttpPlatformResponse 등입니다.

## 3.1 시작하기

다음은 X-API를 이용하여 데이터를 송수신하는 간단한 JSP 예제입니다.

X-API를 이용한 JSP 예제

```
1  <%@ page import="com.nexacro.java.xapi.data.*" %>
2  <%@ page import="com.nexacro.java.xapi.tx.*" %>
3
4  <%@ page contentType="text/xml; charset=UTF-8" %>
5
6  <%
7  // 버퍼(buffer) 초기화
8  out.clearBuffer();
9
10 // HttpServletRequest를 이용하여 HttpPlatformRequest 생성
11 HttpPlatformRequest req = new HttpPlatformRequest(request.getInputStream());
12
13 // 데이터 수신
14 req.receiveData();
15
16 // 수신받은 데이터 획득
17 PlatformData reqData = req.getData();
```

```

18 VariableList reqVarList = reqData.getVariableList();
19
20 // 부서명 획득
21 String name = reqVarList.getString("name");
22
23 // 송신할 데이터 생성
24 PlatformData resData = new PlatformData();
25 VariableList resVarList = resData.getVariableList();
26
27 // 부서별 인원을 저장할 DataSet 생성
28 DataSet employees = new DataSet("employees");
29
30 // DataSet에 열(column) 추가
31 employees.addColumn(new ColumnHeader("name", DataTypes.STRING, 8));
32 employees.addColumn(new ColumnHeader("jobTitle", DataTypes.STRING));
33 employees.addColumn(new ColumnHeader("number", DataTypes.INT));
34 employees.addColumn(new ColumnHeader("manager", DataTypes.BOOLEAN));
35
36 // 부서별 인원 데이터 추가
37 if ("R&D Center".equals(name)) {
38     // 행(row) 추가
39     int row = employees.newRow();
40
41     // 추가된 행(row)의 데이터 설정
42     employees.set(row, "name", "John Jones");
43     employees.set(row, "jobTitle", "developer");
44     employees.set(row, "number", 1234);
45     employees.set(row, "manager", false);
46
47     // ...
48
49     // 정상 수행
50     resData.addDataSet(employees);
51     resVarList.add("ERROR_CODE", 200);
52 } else if ("Quality Assurance".equals(name)) {
53     // 행(row) 추가
54     int row = employees.newRow();
55
56     // 추가된 행(row)의 데이터 설정
57     employees.set(row, "name", "Tom Glover");
58     employees.set(row, "jobTitle", "manager");

```

```

59 employees.set(row, "number", 9876);
60 employees.set(row, "manager", true);
61
62 // ...
63
64 // 정상 수행
65 resData.addDataSet(employees);
66 resVarList.add("ERROR_CODE", 200);
67 } else {
68     // 오류 발생
69     resVarList.add("ERROR_CODE", 500);
70 }
71
72 // HttpServletResponse를 이용하여 HttpPlatformResponse 생성
73 HttpPlatformResponse res = new HttpPlatformResponse(response);
74 res.setData(resData);
75
76 // 데이터 송신
77 res.sendData();
78 %>

```

## 3.2 데이터 송수신

넥사크로와의 데이터 통신은 대부분의 경우 HTTP 상에서 수행되며, 특정 형식으로 변환된 후 송수신됩니다.

송수신 형식(contentType)은 데이터가 송수신되기 위해 객체에서 특정 형식의 데이터(stream)으로 변환되는 것을 의미하며, 프로토콜 형식(protocolType)은 데이터의 압축, 암호화 등을 수행하는 것을 의미합니다.

송수신 형식(contentType)과 프로토콜 형식(protocolType) 등 데이터 변환에 대한 주요 인터페이스는 다음과 같습니다.

인터페이스명	설 명
DataSerializer	PlatformData를 특정 형식의 데이터(stream)으로 변환
DataDeserializer	특정 형식의 데이터(stream)를 PlatformData로 변환
ProtocolEncoder	데이터(stream)의 압축, 암호화 등을 수행
ProtocolDecoder	압축, 암호화 등이 적용된 데이터(stream)를 압축 해제, 복호화 등을 수행

### 3.3 데이터 송수신의 내부 흐름

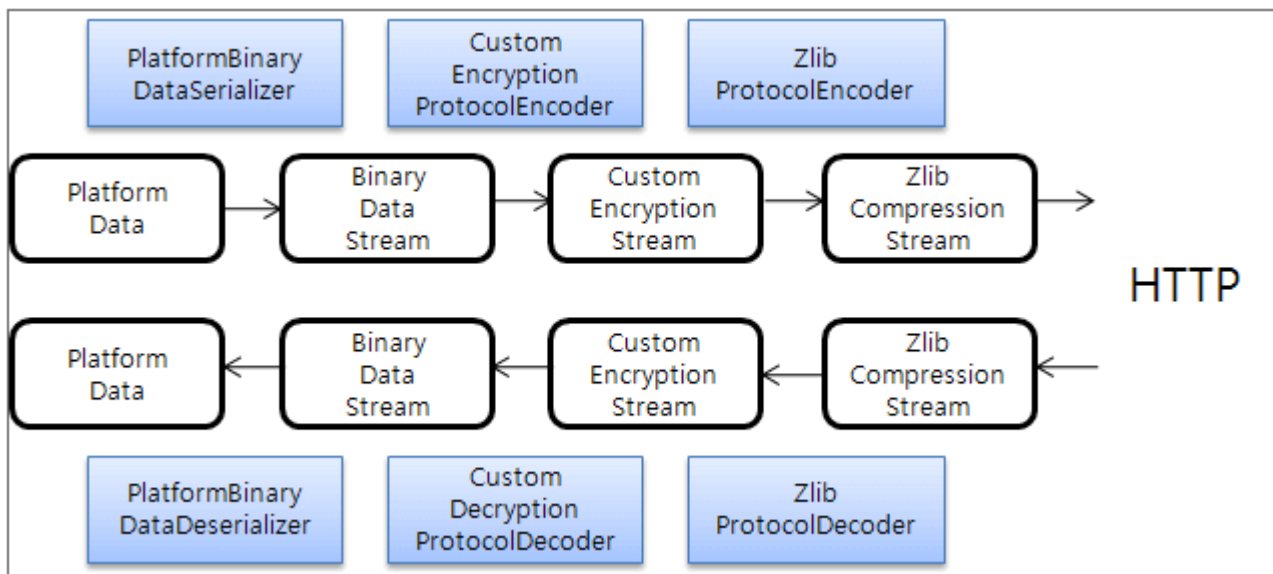
#### 데이터 송신 흐름

1. Server에서 데이터를 PlatformData에 저장한 후 송신함
2. DataSerializer에 의해 PlatformData가 특정 형식의 데이터(stream)로 변환됨
3. ProtocolEncoder에 의해 압축, 암호화 등이 적용됨
4. HTTP 상으로 데이터(stream)가 송신됨

#### 데이터 수신 흐름

1. HTTP 상에서 압축, 암호화 등이 적용된 데이터(stream)를 수신받음
2. ProtocolDecoder에 의해 압축 해제, 복호화 등이 수행됨
3. DataDeserializer에 의해 특정 형식의 데이터(stream)가 PlatformData로 변환됨
4. Client에서 데이터가 저장된 PlatformData를 수신받음

다음은 PlatformData를 Binary 송수신 형식으로 암호화와 압축을 수행하는 경우의 내부 흐름 예이다.



## 3.4 Server 상의 HTTP 데이터 통신

넥사크로와 HTTP 상에서 데이터 통신을 하기 위해서는 `HttpPlatformRequest`와 `HttpPlatformResponse`를 이용하여 JSP 또는 Servlet을 작성합니다.

`HttpPlatformRequest`와 `HttpPlatformResponse`는 `Jakarta.servlet.http.HttpServletRequest`와 `Jakarta.servlet.http.HttpServletResponse`(또는 `javax.servlet.http.HttpServletRequest`와 `javax.servlet.http.HttpServletResponse`)를 이용하여 HTTP 통신을 수행하며, `HttpPlatformRequest`는 넥사크로로 데이터(stream)를 수신받은 후 `PlatformData`으로 변환하고, 그와는 반대로 `HttpPlatformResponse`는 `PlatformData`를 데이터(stream)으로 변환한 후 넥사크로로 송신합니다.

`HttpServletRequest`으로부터 데이터 수신

```
1 // HttpServletRequest를 이용하여 HttpPlatformRequest 생성
2 HttpPlatformRequest req = new HttpPlatformRequest(request.getInputStream());
3
4 // 데이터 수신
5 req.receiveData();
6
7 PlatformData data = req.getData();
```

`HttpServletResponse`으로 데이터 송신

```
1 PlatformData data = ...;
2
3 // HttpServletResponse를 이용하여 HttpPlatformResponse 생성
4 HttpPlatformResponse res = new HttpPlatformResponse(response);
5 res.setData(data);
6
7 // 데이터 송신
8 res.sendData();
```

echo.jsp 예제

```
1 <%@ page import="com.nexacro.java.xapi.tx.*" %>
2 <%@ page import="com.nexacro.java.xapi.data.*" %>
3
4 <%@ page contentType="text/xml; charset=UTF-8" %>
5
6 <%
7   out.clearBuffer();
8
```

```

9  HttpPlatformRequest req = new HttpPlatformRequest(request.getInputStream());
10 req.receiveData();
11 PlatformData data = req.getData();
12
13 HttpPlatformResponse res = new HttpPlatformResponse(response);
14 res.setData(data);
15 res.sendData();
16 %>

```

## 3.5 Client 상의 HTTP 데이터 통신

PlatformHttpClient은 X-API를 이용하여 작성된 JSP 등의 서비스와 통신을 수행하며, 통신하는 과정은 다음과 같습니다.

1. PlatformHttpClient 생성
2. 데이터 전송 : sendData
3. 데이터 수신 : receiveData
4. 종료 : close

PlatformHttpClient을 이용한 서버와의 통신

```

1  PlatformData reqData = ...;
2
3  // Client 생성
4  String url = "http://host/context/service.jsp";
5  PlatformHttpClient client = new PlatformHttpClient(url);
6
7  // 데이터 송신
8  client.sendData(reqData);
9
10 // 데이터 수신
11 PlatformData resData = client.receiveData();
12
13 // 종료
14 client.close();

```

## 3.6 데이터 송수신 형식

송수신 형식은 **데이터 송수신**에서 언급되었듯이 데이터가 송수신되기 위해 객체에서 특정 형식의 데이터(stream)으로 변환되는 형식을 의미하며, 기 구현된 형식은 다음과 같습니다.

상수값	설 명
PlatformType.CONTENT_TYPE_XML	Platform에서 정의된 XML 형식
PlatformType.CONTENT_TYPE_BINARY	Platform에서 정의된 바이너리 형식
PlatformType.CONTENT_TYPE_SSV	Platform에서 정의된 SSV 형식
PlatformType.CONTENT_TYPE_JSON	Platform에서 정의된 JSON 형식

XML 형식으로 데이터를 수신하고, Binary 형식으로 데이터를 송신하는 예제

```

1  <%@ page import="com.nexacro.java.xapi.data.*" %>
2  <%@ page import="com.nexacro.java.xapi.tx.*" %>
3
4  <%
5  // XML 형식으로 데이터 수신
6  HttpPlatformRequest req = new HttpPlatformRequest(request, PlatformType.CONTENT_TYPE_XML);
7  req.receiveData();
8
9  PlatformData data = req.getData();
10
11 // Binary 형식으로 데이터 송신
12 HttpPlatformResponse res = new HttpPlatformResponse(response, PlatformType.CONTENT_TYPE_
13 BINARY);
14 res.setData(data);
15
16 res.sendData();
17 %>
```

만약에 사용자에게 의해 정의된 송수신 형식이 아닌 기 제공되는 송수신 형식인 경우에는 다음과 같이 HttpPlatformRequest에 별도의 송수신 형식을 지정하지 않아도, 내부에서 자동으로 송수신 형식을 판단하여 처리합니다.

송수신 형식을 자동으로 판단하는 HttpPlatformRequest

```

1  <%@ page import="com.nexacro.java.xapi.data.*" %>
2  <%@ page import="com.nexacro.java.xapi.tx.*" %>
3
4  <%@ page contentType="text/xml; charset=UTF-8" %>
5
```

```

6  <%
7  out.clearBuffer();
8
9  // 송수신 형식을 자동으로 판단하여 데이터 수신
10 HttpPlatformRequest req = new HttpPlatformRequest(request.getInputStream());
11 req.receiveData();
12
13 PlatformData data = req.getData();
14
15 // HttpPlatformRequest에 의해 판단된 송수신 형식으로 데이터 송신
16 HttpPlatformResponse res = new HttpPlatformResponse(response.getOutputStream(), req);
17 res.setData(data);
18
19 res.sendData();
20 %>

```

## 3.7 데이터 프로토콜 형식

프로토콜 형식은 데이터 송수신에서 언급되었듯이 데이터의 압축, 암호화 등을 수행하는 것을 의미하며, 기 구현된 형식은 다음과 같습니다.

상수값	설 명
PlatformType.PROTOCOL_TYPE_ZLIB	ZLIB 방식으로 압축

ZLIB 방식으로 압축하여 데이터 송신

```

1  <%@ page import="com.nexacro.java.xapi.data.*" %>
2  <%@ page import="com.nexacro.java.xapi.tx.*" %>
3
4  <%
5  // 데이터 수신
6  HttpPlatformRequest req = new HttpPlatformRequest(request.getInputStream());
7  req.receiveData();
8
9  PlatformData data = req.getData();
10
11 // ZLIB 방식으로 압축하여 데이터 송신
12 HttpPlatformResponse res = new HttpPlatformResponse(response, PlatformType.CONTENT_TYPE_

```



```

BINARY);
13 res.addProtocolType(PlatformType.PROTOCOL_TYPE_ZLIB);
14 res.setData(data);
15
16 res.sendData();
17 %>

```

## 3.8 데이터 분할 송신

일반적으로 데이터는 DataSet에 저장하고, HttpPlatformResponse를 이용하여 넥사크로로 데이터를 전달합니다. 그러나, 데이터 건수가 많은 대용량 데이터의 경우 모든 데이터를 DataSet에 저장해야하므로, 메모리를 많이 사용하여 시스템에 부담을 줄 수 있습니다.

이런 문제를 대처하기 위해 데이터를 여러번 나누어 송신하는 기능을 제공합니다. 데이터는 나누어 송신하지만, 다수의 연결(connection)이 생성되는것이 아니라, 하나의 연결(connection)로 모든 데이터를 송신함을 유의합니다.

데이터 분할 송신은 다음의 순서로 진행되어야 합니다.

1. 객체 초기화 - HttpPartPlatformResponse 생성
2. 송신 시작 - HttpPartPlatformResponse.start() 호출 (생략 가능)
3. Variable 송신 - HttpPartPlatformResponse.sendVariable(Variable) 호출
4. Variable 송신 반복
5. DataSet 송신 - HttpPartPlatformResponse.sendDataSet(DataSet) 호출
6. DataSet 송신 반복
7. 송신 종료 - HttpPartPlatformResponse.end() 호출 (반드시 호출 필요)

Variable 송신과 DataSet 송신은 각각 생략이 가능하며, DataSet 송신 후에 Variable 송신하는 경우에는 예외가 발생합니다.

데이터 분할 송신

```

1 // HttpPartPlatformResponse 생성
2 HttpPartPlatformResponse res = new HttpPartPlatformResponse(response);
3
4 // "company" Variable 송신
5 Variable companyVar = Variable.createVariable("company", "Amazon.com, Inc.");
6 res.sendVariable(companyVar);
7
8 // "url" Variable 송신
9 Variable urlVar = Variable.createVariable("url", "http://www.amazon.com/");

```

```

10 res.sendVariable(urlVar);
11
12 // "2011BestBooks" DataSet 생성
13 DataSet bestBooksDs = new DataSet("2011BestBooks");
14 bestBooksDs.addColumn("title", DataTypes.STRING, 64);
15 bestBooksDs.addColumn("author", DataTypes.STRING, 64);
16 bestBooksDs.addColumn("publisher", DataTypes.STRING, 64);
17 bestBooksDs.addColumn("price", DataTypes.INT, 16);
18
19 // "2011BestBooks" DataSet의 데이터 추가
20 String[][] bestBooks = {
21     { "Lost in Shangri-La", "Mitchell Zuckoff", "Harper", "27" }
22     // ...
23 };
24
25 for (int i = 0; i < bestBooks.length; i++) {
26     int row = bestBooksDs.newRow();
27     bestBooksDs.set(row, "title", bestBooks[i][0]);
28     bestBooksDs.set(row, "author", bestBooks[i][1]);
29     bestBooksDs.set(row, "publisher", bestBooks[i][2]);
30     bestBooksDs.set(row, "price", Float.parseFloat(bestBooks[i][3]));
31 }
32
33 // "2011BestBooks" DataSet 송신
34 // 송신 후의 DataSet 데이터는 자동으로 삭제됨
35 res.sendDataSet(bestBooksDs);
36
37 // "fictionBooks" DataSet 생성
38 DataSet fictionBooksDs = new DataSet("fictionBooks");
39 fictionBooksDs.addColumn("title", DataTypes.STRING, 64);
40 fictionBooksDs.addColumn("author", DataTypes.STRING, 64);
41 fictionBooksDs.addColumn("publisher", DataTypes.STRING, 64);
42 fictionBooksDs.addColumn("price", DataTypes.INT, 16);
43
44 // "fictionBooks" DataSet의 "comic" 데이터 추가
45 String[][] comicBooks = {
46     { "The Slackers Guide to U.S. History", "John Pfeiffer", "Adams Media", "13" }
47     // ...
48 };
49
50 for (int i = 0; i < comicBooks.length; i++) {

```

```
51 // ...
52 }
53
54 // "fictionBooks" DataSet의 "comic" 데이터 송신
55 res.sendDataSet(fictionBooksDs);
56
57 // "fictionBooks" DataSet의 "drama" 데이터 추가
58 String[][] dramaBooks = {
59     { "Megan's Way", "Melissa Foster", "Outskirts Press, Inc.", "15" }
60     // ...
61 };
62
63 for (int i = 0; i < dramaBooks.length; i++) {
64     // ...
65 }
66
67 // "fictionBooks" DataSet의 "drama" 데이터 송신
68 res.sendDataSet(fictionBooksDs);
69
70 // "fictionBooks" DataSet의 "essays" 데이터 추가
71 String[][] essaysBooks = {
72     { "Dracula", "Bram Stoker", "Bedrick. Blackie", "8" }
73     // ...
74 };
75
76 for (int i = 0; i < essaysBooks.length; i++) {
77     // ...
78 }
79
80 // "fictionBooks" DataSet의 "essays" 데이터 송신
81 res.sendDataSet(fictionBooksDs);
82
83 // HttpPartPlatformResponse 종료
84 res.end();
```

## 3.9 HTTP GET 방식의 데이터

로는 데이터를 HTTP GET 방식으로 전달하고, 이를 수신받기를 원하는 경우도 있다. 대부분의 경우 간단한 데이터일 것입니다.

이를 위해서 `HttpPlatformRequest`는 다음과 같은 2가지 속성을 지원합니다.

속성명	데이터 형식	유효한 값	기본값	설 명
<code>http.getparameter.register</code>	String	true 또는 false	false	데 이 터 수 신 시 HT TP G ET 데 이 터 의 등 록 여 부
<code>http.getparameter.asvariable</code>	String	true 또는 false	false	HT TP G ET 데 이 터 등 록 시 Va ria bl e 형 식 으

속성명	데이터 형식	유효한 값	기본값	설 명
				로 의 변 환 여 부, fal se 인 경 우 HT TP G ET 데 이 터 는 Da ta Se t 형 식 으 로 변 환

#### HTTP GET 방식의 데이터 수신

```

1  // HttpPlatformRequest 생성
2  HttpPlatformRequest req = new HttpPlatformRequest(request.getInputStream());
3  // HTTP GET 데이터의 등록 설정
4  req.setProperty("http.getparameter.register", "true");
5  // Variable 형식으로의 변환 설정
6  req.setProperty("http.getparameter.asvariable", "true");
7
8  // 데이터 수신
9  req.receiveData();
10

```

```

11 PlatformData data = req.getData();
12
13 // HttpPlatformResponse 생성
14 HttpPlatformResponse res = new HttpPlatformResponse(response);
15 res.setData(data);
16
17 // 데이터 송신
18 res.sendData();

```

## 3.10 파일 업로드

파일 업로드는 Client에서 Server (X-API)로 파일을 전송하는 것을 의미합니다. X-API는 파일 송수신을 위한 API가 아니기 때문에, 가급적 타 제품 또는 타 패키지를 이용하여 파일을 송수신할 것을 권장합니다. 특히, X-API에서 XML 형식으로 파일을 송수신하는 경우 메모리를 많이 사용하므로, 성능에 영향을 미칠 수 있음을 유의합니다.

만약에 송신되는 또는 수신받은 데이터의 데이터 형식(dataType)을 변경하기 원하는 경우 DataTypeChanger를 이용하여 변경할 수 있습니다. 이 DataTypeChanger를 이용하여 수신받은 byte 배열 또는 String 데이터를 파일로 저장할 수 있습니다.

### 파일 업로드 과정

1. Client에서 파일 데이터를 byte 배열 형식으로 DataSet에 저장
2. Client에서 Server로 데이터 송신
3. Server에서 byte 배열 형식으로 송신된 데이터의 데이터 형식(dataType)을 DataTypes.FILE 데이터 형식(dataType)으로 변환하는 DataTypeChanger 등록
4. Server에서 Client로부터 데이터 수신
5. Server에서 데이터를 수신하는 과정에 DataTypes.FILE 데이터 형식(dataType)으로 변환되는 데이터는 자동으로 임시 파일로 저장
6. Server에서 저장된 임시 파일 조작

파일 업로드 jsp 예제

```

1 <%@ page import="java.io.*" %>
2
3 <%@ page import="com.nexacro.java.xapi.tx.*" %>
4 <%@ page import="com.nexacro.java.xapi.data.*" %>
5
6 <%@ page contentType="text/xml; charset=UTF-8" %>

```

```

7
8 <%
9 // 버퍼(buffer) 초기화
10 out.clearBuffer();
11
12 // HttpPlatformRequest 생성
13 HttpPlatformRequest req = new HttpPlatformRequest(request.getInputStream());
14 // byte 배열 데이터를 파일로 저장하기 위한 DataTypeChanger 설정
15 req.setDataTypeChanger(new UploadDataTypeChanger());
16
17 // 데이터 수신
18 req.receiveData();
19
20 // 수신된 데이터 참조
21 PlatformData reqData = req.getData();
22
23 // 임시로 저장된 파일들을 업로드될 위치로 이동
24 copyFiles(reqData);
25
26 // 송신 데이터 생성
27 PlatformData resData = new PlatformData();
28 VariableList resVl = resData.getVariableList();
29
30 // 오류코드 설정
31 resVl.add("ERROR_CODE", "200");
32
33 // HttpPlatformResponse 생성
34 HttpPlatformResponse res = new HttpPlatformResponse(response.getOutputStream(), req);
35 // 송신 데이터 설정
36 res.setData(resData);
37
38 // 데이터 송신
39 res.sendData();
40 %>
41
42 <%!
43 // 임시로 저장된 파일들을 업로드될 위치로 이동
44 void copyFiles(PlatformData data) {
45     // 파일들이 업로드될 위치
46     String dir = "C:\\upload";
47     // 파일 데이터가 저장된 DataSet 참조

```

```

48  DataSet ds = data.getDataSet("resources");
49  // DataSet의 행(row)의 갯수 참조
50  int count = (ds == null) ? 0 : ds.getRowCount();
51
52  // DataSet의 행(row)의 갯수, 즉 업로드된 파일의 갯수만큼 순환
53  for (int i = 0; i < count; i++) {
54      // 파일명 참조
55      String name = ds.getString(i, "name");
56      // 파일크기 참조
57      int size = ds.getInt(i, "size");
58      // 파일의 변경시간 참조
59      long lastWriteTime = ds.getLong(i, "lastWriteTime");
60      // 임시로 저장된 파일의 경로 참조
61      String filename = ds.getString(i, "content");
62
63      // 임시로 저장된 File
64      File file = new File(filename);
65      // 업로드될 위치로 이동할 File
66      File dest = new File(dir, name);
67
68      // 파일 이동
69      file.renameTo(dest);
70  }
71 }
72
73 // 수신 받은 DataSet 열(column)의 데이터 형식(dataType)을 변경하는 DataTypeChanger
74 class UploadDataTypeChanger implements DataTypeChanger {
75
76     // byte 배열의 데이터가 저장된 DataSet 열(column)의 데이터 형식(dataType)을 DataTypes.
    FILE 데이터 형식(dataType)으로 변경
77     // 수신 받는 데이터의 데이터 형식(dataType)을 DataTypes.FILE 데이터 형식(dataType)으로
    변경하는 경우
78     // 데이터는 자동으로 임시 파일로 저장되고, DataSet에는 저장된 임시 파일의 경로가
    저장된다.
79     public int getDataType(String dsName, String columnName, int dataType) {
80         // "resources" DataSet의 "content" 열(column)인 경우 DataTypes.FILE 데이터 형식(
    dataType) 반환
81         if ("resources".equals(dsName) && "content".equals(columnName)) {
82             return DataTypes.FILE;
83         }
84

```



```

85     // 이외에는 원 데이터 형식(dataType) 반환
86     return dataType;
87 }
88 }
89 %>

```

## 3.11 파일 다운로드

파일 다운로드 Server (X-API)에서 Client로 파일을 전송하는 것을 의미합니다. X-API는 파일 송수신을 위한 API가 아니기 때문에, 가급적 타 제품 또는 타 패키지를 이용하여 파일을 송수신할 것을 권장합니다. 특히, X-API에서 XML 형식으로 파일을 송수신하는 경우 메모리를 많이 사용하므로, 성능에 영향을 미칠 수 있음을 유의합니다.

파일의 데이터를 Client로 송신을 원하는 경우에는 다음의 2가지 경우가 가능합니다.

- DataSet의 열(column)을 DataTypes.BLOB 형식으로 추가하고, 파일의 데이터를 byte 배열 형식으로 읽고, DataSet에 설정한다.
- DataSet의 열(column)을 DataTypes.FILE 형식으로 추가하고, 다운로드할 파일의 경로를 DataSet에 설정한다. 데이터 송신시 DataTypes.FILE 형식은 자동으로 DataTypes.BLOB 형식으로 변환되며, 파일 경로에 위치한 파일의 데이터를 송신한다.

다음은 DataTypes.FILE 데이터 형식(dataType)을 이용한 파일 다운로드 예제입니다.

파일 다운로드 jsp 예제

```

1  <%@ page import="java.io.*" %>
2
3  <%@ page import="com.nexacro.java.xapi.tx.*" %>
4  <%@ page import="com.nexacro.java.xapi.data.*" %>
5
6  <%@ page contentType="text/xml; charset=UTF-8" %>
7
8  <%
9  // 버퍼(buffer) 초기화
10 out.clearBuffer();
11
12 // HttpPlatformRequest 생성
13 HttpPlatformRequest req = new HttpPlatformRequest(request.getInputStream());
14
15 // 데이터 수신

```

```

16 req.receiveData();
17
18 // 수신 데이터 참조
19 PlatformData reqData = req.getData();
20
21 // 파일 데이터가 포함된 송신 데이터 생성
22 PlatformData resData = createData();
23
24 // HttpPlatformResponse 생성
25 HttpPlatformResponse res = new HttpPlatformResponse(response.getOutputStream(), req);
26 // 송신 데이터 설정
27 res.setData(resData);
28
29 // 데이터 송신
30 res.sendData();
31 %>
32
33 <%!
34 // 파일 데이터가 포함된 송신 데이터 생성
35 PlatformData createData() {
36     // PlatformData 생성
37     PlatformData data = new PlatformData();
38
39     // 파일 데이터를 저장할 DataSet 생성
40     DataSet ds = new DataSet("resources");
41
42     // 파일명의 열(column) 추가
43     ds.addColumn("name", DataTypes.STRING, 128);
44     // 파일크기의 열(column) 추가
45     ds.addColumn("size", DataTypes.INT);
46     // 파일 변경시간의 열(column) 추가
47     ds.addColumn("lastWriteTime", DataTypes.LONG);
48     // 파일 데이터의 열(column) 추가
49     // 데이터 형식(dataType)을 DataTypes.FILE 데이터 형식(dataType)으로 추가하는 경우
50     // 데이터 송신시 데이터 형식(dataType)은 자동으로 DataTypes.BLOB 데이터 형식(dataType)으로 변환되며,
51     // 데이터는 설정된 파일 경로의 파일 데이터를 전송한다.
52     ds.addColumn("content", DataTypes.FILE);
53
54     // DataSet의 행(row) 데이터 추가
55     addRow(ds, "C:\\download\\data_structure.gif");

```

```

56  addRow(ds, "C:\\download\\serialize_flow.gif");
57
58  // PlatformData의 DataSet 추가
59  data.addDataSet(ds);
60
61  // 생성된 PlatformData 반환
62  return data;
63 }
64
65 // DataSet의 행(row) 데이터 추가
66 void addRow(DataSet ds, String filename) {
67     // DataSet의 행(row) 추가
68     int row = ds.newRow();
69     // 다운로드될 File 생성
70     File file = new File(filename);
71
72     // 파일명 설정
73     ds.set(row, "name", file.getName());
74     // 파일크기 설정
75     ds.set(row, "size", file.length());
76     // 파일의 변경시간 설정
77     ds.set(row, "lastWriteTime", file.lastModified());
78     // 파일의 경로 설정
79     ds.set(row, "content", file.getPath());
80 }
81 %>

```

## 3.12 Stream을 이용한 데이터 통신

때로는 Socket과 File 등과 같이 InputStream과 OutputStream을 이용하여 데이터를 통신할 필요성이 발생합니다. 이런 경우 PlatformRequest와 PlatformResponse를 이용하여 데이터를 주고 받을 수 있습니다. 물론, 단 방향만 수행할 수도 있습니다.

PlatformRequest와 PlatformResponse를 이용한 데이터 통신

```

1  InputStream in = ...;
2
3  // PlatformRequest 생성

```

```

4 PlatformRequest req = new PlatformRequest(in);
5
6 // 데이터 수신
7 req.receiveData();
8 PlatformData reqData = req.getData();
9
10 OutputStream out = ...;
11
12 // PlatformResponse 생성
13 PlatformData resData = ...;
14 PlatformResponse res = new PlatformResponse(out);
15 res.setData(resData);
16
17 // 데이터 송신
18 res.sendData();

```

참고로, 다음과 같이 PlatformRequest와 PlatformResponse를 이용하여 PlatformHttpClient와 동일한 기능을 수행할 수도 있습니다.

PlatformRequest와 PlatformResponse를 이용한 서버와의 통신

```

1 // 연결 생성
2 String loc = "http://host/context/service.jsp";
3 URL url = new URL(loc);
4
5 HttpURLConnection conn = (HttpURLConnection) url.openConnection();
6 conn.setRequestProperty("Content-Type", "text/xml; charset=UTF-8");
7 conn.setRequestMethod("POST");
8 conn.setDoOutput(true);
9 conn.setDoInput(true);
10
11 // 데이터 송신
12 PlatformData sendingData = ...;
13 OutputStream out = conn.getOutputStream();
14 PlatformResponse res = new PlatformResponse(out, PlatformType.CONTENT_TYPE_XML, "UTF-8");
15 res.setData(sendingData);
16 res.sendData();
17 out.close();
18
19 // 데이터 수신
20 InputStream in = conn.getInputStream();
21 PlatformRequest req = new PlatformRequest(in);

```

```

22 req.receiveData();
23 PlatformData receivedData = req.getData();
24 in.close();
25
26 // 연결 종료
27 conn.disconnect();

```

## 3.13 파일로부터의 데이터 적재와 저장

파일의 데이터를 읽거나 쓰기 위해서는 다음과 같은 방법이 있습니다.

- PlatformRequest와 PlatformResponse를 이용하여 직접 읽거나 쓰는 방법
- PlatformFileClient를 이용하여 데이터를 읽거나 쓰는 방법

파일의 데이터 읽기와 쓰기

```

1  // 데이터가 저장된 파일
2  String sourceFilename = ...;
3  InputStream source = new FileInputStream(sourceFilename);
4
5  // 파일로부터 데이터 읽기
6  PlatformRequest req = new PlatformRequest(source, PlatformType.CONTENT_TYPE_XML);
7  req.receiveData();
8
9  source.close();
10
11 PlatformData data = req.getData();
12
13 // 데이터를 저장할 파일
14 String targetFilename = ...;
15 OutputStream target = new FileOutputStream(targetFilename);
16
17 // 데이터를 파일로 쓰기
18 PlatformResponse res = new PlatformResponse(target, PlatformType.CONTENT_TYPE_BINARY);
19 res.setData(data);
20
21 res.sendData();
22

```

```
23 target.close();
```

PlatformFileClient를 이용한 데이터 읽기와 쓰기

```
1  // 데이터가 저장된 파일
2  String sourceFilename = ...;
3  // 데이터를 저장할 파일
4  String targetFilename = ...;
5
6  // PlatformFileClient 생성
7  PlatformFileClient client = new PlatformFileClient(sourceFilename, targetFilename);
8  // 데이터가 저장된 형식
9  client.setSourceContentType(PlatformType.CONTENT_TYPE_XML);
10 // 데이터를 저장할 형식
11 client.setTargetContentType(PlatformType.CONTENT_TYPE_BINARY);
12
13 // 파일로부터 데이터 읽기
14 PlatformData data = client.receiveData();
15
16 // 데이터를 파일로 쓰기
17 client.sendData(data);
18
19 // 종료
20 client.close();
```

## 3.14 PlatformData를 XML 문자열로 변환

PlatformData를 XML 문자열로 변환하기 위해서는 PlatformResponse를 이용하여 가능합니다. 즉, XML 문자열을 저장할 버퍼를 생성하여 PlatformResponse으로 전달하고, PlatformResponse는 XML 방식으로 PlatformData를 출력하면 됩니다.

PlatformData를 XML 문자열로 변환

```
1 PlatformData data = ...;
2
3 // XML 문자열을 저장할 버퍼
4 Writer out = new CharArrayWriter();
5
6 // CONTENT_TYPE_XML 형식으로 PlatformResponse 생성
```

```

7 PlatformResponse res = new PlatformResponse(out, PlatformType.CONTENT_TYPE_XML);
8 res.setData(data);
9
10 // XML 문자열 저장
11 res.sendData();
12
13 out.close();
14
15 // 저장된 XML 문자열
16 String xml = out.toString();

```

## 3.15 추가, 변경, 삭제된 데이터 송신

DataSet은 데이터가 추가, 변경, 삭제된 경우 변경된 상태와 변경 이전의 원본 데이터를 저장합니다. 자세한 정보는 [DataSet의 원본 데이터와 변경된 데이터](#)를 참조합니다.

PlatformResponse 또는 HttpPlatformResponse는 위와 같은 데이터를 저장 방식(saveType)에 따라 구분하여 전송합니다.

상수값	설 명
DataSet.SAVE_TYPE_NONE	미설정
DataSet.SAVE_TYPE_ALL	현재의 데이터와 추가, 변경, 삭제된 모든 데이터 저장 또는 전송
DataSet.SAVE_TYPE_NORMAL	현재의 데이터만 저장 또는 전송
DataSet.SAVE_TYPE_UPDATED	추가, 변경된 데이터만 저장 또는 전송
DataSet.SAVE_TYPE_DELETED	삭제된 데이터만 저장 또는 전송
DataSet.SAVE_TYPE_CHANGED	추가, 변경, 삭제된 데이터만 저장 또는 전송

저장 방식(saveType)은 PlatformData와 DataSet이 가지고 있으며, DataSet의 저장 방식이 우선으로 적용되고, DataSet의 저장 방식이 DataSet.SAVE\_TYPE\_NONE인 경우 PlatformData의 저장 방식이 적용됩니다.

만약에 PlatformData의 저장 방식도 DataSet.SAVE\_TYPE\_NONE인 경우 기본값 DataSet.SAVE\_TYPE\_NORMAL이 적용됩니다.

## 3.16 StreamLog를 이용한 송수신 데이터(stream) 저장

PlatformRequest의 StreamLog는 클라이언트로부터 수신받은 데이터(stream)를 특정 위치에 저장하는 역할을 수행합니다.

이것은 서버에서 데이터 수신중에 오류가 발생하거나, 수신받은 데이터가 의심스러운 경우 파일로 저장하여 정확한 확인을 위한 것입니다.

주의할 점은 StreamLog가 활성화된 경우 메모리를 많이 차지할 수 있으므로, 반드시 필요한 경우에만 사용합니다.

수신받은 데이터(stream) 저장

```

1 // HttpServletRequest를 이용하여 HttpPlatformRequest 생성
2 HttpPlatformRequest req = new HttpPlatformRequest(request.getInputStream());
3
4 // 수신받은 데이터(stream) 로그의 활성화
5 req.setStreamLogEnabled(true);
6 // 수신받은 데이터(stream) 로그의 저장 위치
7 req.setStreamLogDir("/home/log");
8
9 // 데이터 수신
10 // 예외가 발생한 경우 자동적으로 로그 저장 위치에 수신받은 데이터(stream)가 저장됨
11 req.receiveData();
12
13 // 예외가 발생하지 않더라도 필요한 경우 강제로 로그 저장
14 req.storeStreamLog();
15
16 PlatformData data = req.getData();

```

## 3.17 localhost 테스트

local PC 상에서 "localhost" 또는 "127.0.0.1"으로 시작하는 URL으로 요청하는 경우 라이선스 파일 없이 동작이 가능합니다.

라이선스 파일 없이 동작하기 위해서는 다음과 같이 작성하여야 합니다.

1. HttpServletRequest를 이용하여 HttpPlatformRequest를 생성한다.
2. HttpServletResponse와 이전 과정에서 생성된 HttpPlatformRequest를 이용하여 HttpPlatformResponse를



생성한다.

3. local PC 상에서 "localhost" 또는 "127.0.0.1" 으로 시작되는 URL으로 요청한다.

localhost 테스트 방법

```

1  <%@ page import="com.nexacro.java.xapi.tx.*" %>
2  <%@ page import="com.nexacro.java.xapi.data.*" %>
3
4  <%@ page contentType="text/xml; charset=UTF-8" %>
5
6  <%
7  out.clearBuffer();
8
9  HttpPlatformRequest req = new HttpPlatformRequest(request.getInputStream());
10 req.receiveData();
11 PlatformData data = req.getData();
12
13 HttpPlatformResponse res = new HttpPlatformResponse(response.getOutputStream(), req);
14 res.setData(data);
15 res.sendData();
16 %>

```

## 3.18 X-API의 내부 로그 출력하기

X-API의 내부 로깅(logging)은 Apache의 Commons Logging을 이용하여 출력합니다. Apache Commons Logging에 대한 자세한 사항은 아래 링크를 참조하기 바랍니다. 따라서, Apache Commons Logging의 방침에 따라 로깅 설정을 하여 로그를 출력하고, 출력된 로그를 분석하여 개발 또는 운영시에 참조합니다.



<http://commons.apache.org/logging/>  
<http://logging.apache.org/log4j/>

다음은 Apache Log4j를 이용하여 X-API의 내부 로그를 파일로 출력하는 예제입니다.

1. 클래스 경로에 X-API와 같이 Apache Log4j의 jar(log4j-x.x.x.jar)를 복사한다.
2. log4j.properties를 작성하고, 클래스 경로에 복사한다.
3. WAS(Web Application Server)를 재구동시킨다.
4. X-API를 이용하는 서비스를 호출한다.

5. Apache Log4j에서 설정된 위치에 로그가 파일로 출력되었을 것이다.

#### log4j.properties 예제

```

1 log4j.logger.com.nexacro.java.xapi.data=DEBUG, file
2 log4j.logger.com.nexacro.java.xapi.tx=DEBUG, file
3
4 log4j.appender.file=org.apache.log4j.FileAppender
5 log4j.appender.file.File=xapi.log
6 log4j.appender.file.Append=false
7 log4j.appender.file.layout=org.apache.log4j.PatternLayout
8 log4j.appender.file.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n

```

Apache Tomcat 환경인 경우 다음과 같이 설정할 수 있습니다.

1. Apache Commons Logging의 jar(commons-logging-x.x.x.jar)를 \$CATALINA\_HOME/common/lib으로 복사한다.
2. Apache Log4j의 jar(log4j-x.x.x.jar)를 \$CATALINA\_HOME/common/lib으로 복사한다.
3. log4j.properties를 작성하고, \$CATALINA\_HOME/common/classes으로 복사한다.
4. Apache Tomcat을 재구동시킨다.
5. X-API를 이용하는 서비스를 호출한다.
6. Apache Log4j에서 설정된 위치에 로그가 파일로 출력되었을 것이다.

#### Apache Tomcat의 log4j.properties 예제

```

1 log4j.rootLogger=INFO, tomcat
2 log4j.logger.com.nexacro.java.xapi.tx=DEBUG, xapi
3 log4j.logger.com.nexacro.java.xapi.data=DEBUG, xapi
4
5 log4j.appender.stdout=org.apache.log4j.ConsoleAppender
6 log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
7 log4j.appender.stdout.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n
8
9 log4j.appender.tomcat=org.apache.log4j.FileAppender
10 log4j.appender.tomcat.File=${catalina.home}/logs/tomcat.log
11 log4j.appender.tomcat.Append=false
12 log4j.appender.tomcat.layout=org.apache.log4j.PatternLayout
13 log4j.appender.tomcat.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n
14
15 log4j.appender.xapi=org.apache.log4j.FileAppender
16 log4j.appender.xapi.File=${catalina.home}/logs/xapi.log
17 log4j.appender.xapi.Append=false
18 log4j.appender.xapi.layout=org.apache.log4j.PatternLayout

```

```
19 log4j.appender.xapi.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n
```

## 3.19 데이터에 포함된 null 문자(0x00) 오류

넥사크로의 NRE와 데이터 통신을 수행하는 경우 넥사크로 NRE와 X-API의 null 문자(0x00) 처리 방식의 차이로 인해 오동작이 발생할 수도 있습니다. 넥사크로 NRE에서는 null 문자를 문자열의 종료로 인식하지만, X-API에서는 null 문자를 포함할 수 있습니다. 따라서, X-API에서 null 문자가 포함된 데이터를 넥사크로 NRE로 송신한 경우 넥사크로 NRE에서는 null 문자까지만 표현하고, 나머지 데이터는 무시될 것입니다.

## 3.20 JSP에서 데이터 전송시의 java.lang.IllegalStateException 예외

JSP에서 HttpPlatformResponse를 이용하여 데이터 전송시 동작은 성공적으로 수행되지만, java.lang.IllegalStateException 예외가 발생하게 된다. 이것은 Web Application Server 별로 동작이 다릅니다.

이 예외가 발생하는 원인은 HttpPlatformResponse에서 Jakarta.servlet.http.HttpServletResponse(또는 javax.servlet.http.HttpServletResponse)의 OutputStream을 이용하기 때문입니다. 즉, HttpPlatformResponse에서 이미 HttpServletResponse.getOutputStream() 메소드가 호출되었고, JSP 자체에서도 이를 참조하기 때문에 Web Application Server에서 이 예외를 발생시키는 것으로 예상됩니다.

Web Application Server	예외 발생 여부	예외 메시지 (버전에 따라 다를 수 있음)
IBM WebSphere	발생	java.lang.IllegalStateException: OutputStream already obtained
Oracle WebLogic	무시	
Tmax JEUS	무시	
Apache Tomcat	발생	java.lang.IllegalStateException: getOutputStream() has already been called for this response

XML 또는 CSV 형식으로 데이터를 전송하는 경우에는 Jakarta.servlet.http.HttpServletResponse(또는 javax.servlet.http.HttpServletResponse)의 Writer를 이용하기 때문에 위와 같은 예외가 발생하지 않습니다. SSV 또는 Binary 형식으로 데이터를 전송하는 경우에만 예외가 발생합니다.

이 예외에 대한 대처 방안은 다음과 같습니다.

- JSP 수행에는 지장이 없으므로 무시한다. 테스트 장비에서는 가능할 것이다.
- SSV 또는 Binary 형식으로 전송하는 경우에만 예외가 발생하므로, XML 형식으로만 전송한다.
- JSP 상단에서 `out.clear()`와 `out = pageContext.pushBody()`를 호출하여 예외 발생을 방지한다.
- 근본적인 원인은 JSP에서 Binary 형식으로 전송하기 때문이므로, JSP 대신에 Servlet으로 작성한다.

java.lang.IllegalStateException 예외 처리

```
1  <%@ page import="com.nexacro.java.xapi.tx.*" %>
2  <%@ page import="com.nexacro.java.xapi.data.*" %>
3
4  <%
5    out.clear();
6    out = pageContext.pushBody();
7
8    // ...
9
10   HttpPlatformResponse res = new HttpPlatformResponse(response, PlatformType.CONTENT_TYPE_
    BINARY);
11   res.setData(data);
12   res.sendData();
13 %>
```

**파트 II.**

---

**넥사크로 X-API (C#)**

## 4.

# 소개 및 구성

X-API는 넥사크로 클라이언트와의 통신을 위한 API입니다. X-API의 주목적은 넥사크로 클라이언트와의 통신을 제공하고, 데이터 송수신, 또는 송수신된 데이터의 조작을 단순화함으로써, 개발자는 비즈니스 개발에 초점을 맞추도록 하는 것입니다.

클라이언트와 서버간에 송수신하는 데이터는 key-value으로 구성된 단일 데이터도 존재하고, DB의 Table과 유사한 2차원 데이터도 존재합니다. 이러한 데이터를 조작 또는 송수신하기 위한 것입니다.

X-API의 주요 기능은 아래와 같습니다.

- com.nexacro.dotnet.xapi.data : X-API의 데이터 구조를 정의합니다.
- com.nexacro.dotnet.xapi.tx : X-API의 데이터 통신을 수행합니다.

## 4.1 구성

X-API DLL 파일은 아래와 같이 구성되어 있습니다. .NET Framwork 4.5 이상이 필요합니다.

파일명	설명	필수여부
NexacroXAP-dotnet.dll	X-API	O
NLog.dll	X-API 내부 로깅	O
ICSharpCode.SharpZipLib.dll	X-API Zip library	O

## 4.2 라이선스

X-API의 라이선스 파일명은 "nexacro\_server\_license.xml" 이며, XML 형식으로 구성되어 있습니다. 라이선스 발급은 필수이며 라이선스가 없을 경우 정상 작동하지 않습니다. 라이선스 발급은 고객센터를 통해 발급하여 사용

할 수 있습니다.

라이센스 파일은 NexacroXAP-dotnet.dll 파일과 동일한 디렉토리에 배치합니다.

## 5.

---

# com.nexacro.dotnet.xapi.data (C#)

X-API의 데이터 구조를 정의합니다.

클라이언트와 서버간의 송수신하는 데이터는 단일 데이터도 존재하고, DB의 Table과 유사한 2차원적인 데이터도 존재합니다. 이러한 데이터를 송수신 또는 조작하기 위한 데이터 구조를 정의합니다. 주요 클래스는 PlatformData, DataSet와 Variable 등입니다.

## 5.1 시작하기

다음은 X-API의 데이터를 참조하는 간단한 예제입니다.

데이터 참조

```
1 PlatformData department = ...;
2
3 // VariableList 참조
4 VariableList varList = department.GetVariableList();
5
6 // VariableList으로부터 값 참조
7 String name = varList.GetString("name");
8 String location = varList.GetString("location");
9 int number = varList.GetInt("number");
10 // ...
11
12 // DataSet 참조
13 DataSet employees = department.GetDataSet("employees");
14
15 // DataSet의 행(row)수만큼 순환
16 for (int i = 0; i < employees.GetRowCount(); i++) {
17     // DataSet의 데이터 참조
```



```

18     int id = employees.GetInt(i, "id");
19     String firstName = employees.GetString(i, "firstName");
20     String lastName = employees.GetString(i, "lastName");
21     boolean manager = employees.GetBoolean(i, "manager");
22     // ...
23 }

```

다음은 X-API의 데이터를 생성하는 간단한 예제입니다.

#### 데이터 생성

```

1  // PlatformData 생성
2  PlatformData department = new PlatformData();
3
4  // VariableList 참조
5  VariableList varList = department.GetVariableList();
6
7  // VariableList에 값 추가
8  varList.Add("name", "R&D Center");
9  varList.Add("location", "222 Jamsil-Dong, Songpa-Ku, Seoul");
10 varList.Add("number", 99);
11
12 // DataSet 생성
13 DataSet employees = new DataSet("employees");
14
15 // 열(column) 추가
16 employees.AddColumn("id", DataTypes.INT);
17 employees.AddColumn("firstName", DataTypes.STRING, 16);
18 employees.AddColumn("lastName", DataTypes.STRING, 8);
19 employees.AddColumn("manager", DataTypes.BOOLEAN);
20
21 // 행(row) 추가
22 int row = employees.NewRow();
23
24 // 추가된 행(row)의 데이터 설정
25 employees.Set(row, "id", 0);
26 employees.Set(row, "firstName", "John");
27 employees.Set(row, "lastName", "Jones");
28 employees.Set(row, "manager", false);
29
30 // 행(row) 추가
31 row = employees.NewRow();

```

```

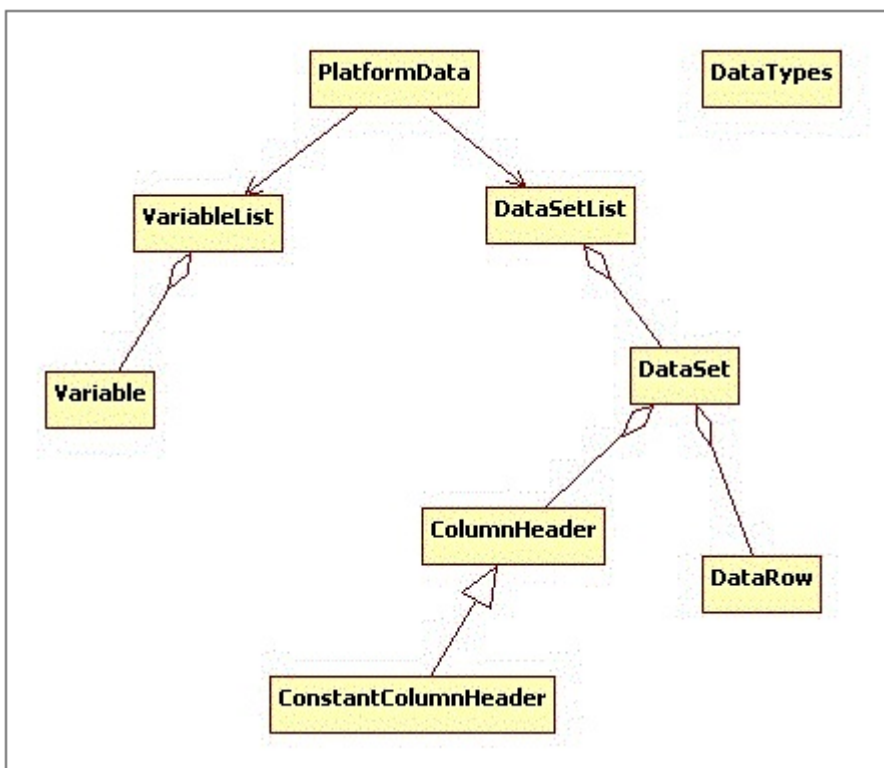
32
33 // 추가된 행(row)의 데이터 설정
34 employees.Set(row, "id", 1);
35 employees.Set(row, "firstName", "Tom");
36 employees.Set(row, "lastName", "Glover");
37 employees.Set(row, "manager", true);
38
39 // DataSet을 PlatformData에 추가
40 department.AddDataSet(employees);

```

## 5.2 데이터 구조

데이터는 크게 단일 데이터와 2차원 데이터로 구분합니다. 단일 데이터는 데이터를 구분할 수 있는 식별자(name)와 값(value)을 가지고 있으며, 이는 VariableList에 저장됩니다. 2차원 데이터를 저장하는 DataSet은 열(column)과 행(row)으로 구성되어 있으며, DataSetList를 통해 저장 또는 참조됩니다.

VariableList와 DataSetList를 가지고 있는 PlatformData는 데이터 구조의 최상위에 위치하고 있으며, 데이터 이동과 데이터 송수신의 기본 단위로 사용되어집니다.



## 5.3 Variable의 단일 데이터 조작

Variable은 데이터를 저장하는 변수를 의미하며, 이는 식별자(name)와 값(value)으로 구성된다. 값(value)은 데이터의 형식(type)에 따라 변환된 후 저장됩니다.

Variable 생성과 데이터 설정은 3가지 방식을 지원합니다.

- Variable(name, type, value)의 생성자 호출
- Variable(name) 또는 Variable(name, type)의 생성자 호출 후 set(value) 메소드 호출
- Variable.createVariable(name, value)의 정적 메소드 호출을 통한 생성

VariableList의 Variable 추가는 Variable 생성 후 add(var) 메소드를 통하여 추가합니다. 또한, Variable을 생성하지 않고 add(name, value) 메소드를 통하여 직접 값으로 추가할 수 있습니다.

단일 데이터(Variable) 추가

```

1  // PlatformData 생성
2  PlatformData department = new PlatformData();
3
4  // VariableList 참조
5  VariableList varList = department.GetVariableList();
6
7  // VariableList에 값을 직접 추가
8  varList.Add("name", "R&D Center");
9
10 // Variable을 생성한 후 VariableList에 값을 추가
11 Variable location = new Variable("location");
12 location.Set("222 Jamsil-Dong, Songpa-Ku, Seoul");
13 varList.Add(location);
14
15 // VariableList에 int 형의 값을 직접 추가
16 varList.Add("number", 99);

```

## 5.4 DataSet의 2차원 데이터 참조

DataSet은 열(column)과 행(row)으로 구성되며, 2차원 데이터를 저장합니다. 구조는 DB의 Table과 유사하며, 열(column)에 대한 정보는 ColumnHeader에 의해 저장되고, 데이터는 내부 클래스인 DataRow에 의해 행(row) 단위

로 저장됩니다.

DataSet에 저장된 데이터는 행(row)의 위치(index)와 열(column)의 이름(name) 또는 위치(index)로 참조하며, Variable과 동일한 방식으로 getObject(rowIndex, columnIndex)과 getString(rowIndex, columnIndex)등의 필요한 데이터 형식에 따른 메소드를 이용하여 값을 참조할 수 있습니다.

마찬가지로 주의할 점은 원 데이터의 형식과 다른 데이터 형식으로 반환을 요청한 경우 데이터의 변형이 발생할 수 있습니다.

식별자(name)를 이용한 DataSet의 데이터 참조

```
1 PlatformData department = ...;
2
3 // DataSet을 식별자(name)를 이용하여 참조
4 DataSet employees = department.GetDataSet("employees");
5
6 // DataSet의 행(row)수만큼 순환
7 for (int i = 0; i < employees.GetRowCount(); i++) {
8     // DataSet의 데이터를 식별자(name)를 통하여 참조
9     object name = employees.GetObject(i, "name");
10    string jobTitle = employees.GetString(i, "jobTitle");
11    int number = employees.GetInt(i, "number");
12    bool manager = employees.GetBoolean(i, "manager");
13    // ...
14 }
```

위치(index)를 이용한 DataSet의 데이터 참조

```
1 PlatformData department = ...;
2
3 // DataSet을 위치(index)를 이용하여 참조
4 DataSet employees = department.GetDataSet(0);
5
6 // DataSet의 행(row)수만큼 순환
7 for (int i = 0; i < employees.GetRowCount(); i++) {
8     // DataSet의 데이터를 열(column)의 위치(index)를 통하여 참조
9     object name = employees.GetObject(i, 0);
10    string jobTitle = employees.GetString(i, 1);
11    int number = employees.GetInt(i, 2);
12    bool manager = employees.GetBoolean(i, 3);
13    // ...
14 }
```

## 5.5 DataSet 생성

DataSet 생성은 다음과 같은 과정으로 이루어집니다.

1. DataSet 생성
2. 열(column) 추가
3. 행(row) 추가
4. 데이터 설정

DataSet 생성과 데이터 추가

```

1 PlatformData department = new PlatformData();
2 // DataSet 생성
3 DataSet employees = new DataSet("employees");
4
5 // 열(column) 추가
6 employees.AddColumn("name", DataTypes.STRING, 8);
7 employees.AddColumn("jobTitle", DataTypes.STRING, 16);
8 employees.AddColumn("number", DataTypes.INT);
9 employees.AddColumn("manager", DataTypes.BOOLEAN);
10
11 // 행(row) 추가
12 int row = employees.NewRow();
13
14 // 데이터 설정
15 employees.Set(row, "name", "John Jones");
16 employees.Set(row, "jobTitle", "developer");
17 employees.Set(row, "number", 1234);
18 employees.Set(row, "manager", false);
19
20 // 행(row) 추가
21 row = employees.NewRow();
22
23 // 데이터 설정
24 employees.Set(row, "name", "Tom Glover");
25 employees.Set(row, "jobTitle", "manager");
26 employees.Set(row, "number", 9876);
27 employees.Set(row, "manager", true);
28
29 // DataSet을 PlatformData에 추가
30 department.AddDataSet(employees);

```

## 5.6 ColumnHeader의 속성 참조

DataSet의 열(column)에 대한 정보는 ColumnHeader에 의해 저장되며, 열(column)에 대한 정보는 다음과 같습니다.

속성명	변수명	데이터 형식	유효한 값
식별자	name	String	null과 ""를 제외한 DataSet 내에서 유일한 문자열
열(column)의 형식	type	int	일반적인 열(TYPE_NORMAL)과 상수값을 가진 열(TYPE_CONSTANT)
데이터 형식	dataType	int	DataTypes에 정의된 상수 참조
데이터 크기	dataSize	int	정수값
값	value	Object	ConstantColumnHeader 내에서만 유효

ColumnHeader의 속성 참조

```

1 PlatformData department = ...;
2
3 // DataSet 참조
4 DataSet employees = department.GetDataSet("employees");
5
6 // DataSet의 열(column)수만큼 순환
7 for (int i = 0; i < employees.GetColumnCount(); i++) {
8     // DataSet으로부터 ColumnHeader 참조
9     ColumnHeader columnHeader = employees.GetColumn(i);
10
11     // 열(column)의 속성 참조
12     string name = columnHeader.GetName();
13     int type = columnHeader.GetType();
14     int dataType = columnHeader.GetDataType();
15     int dataSize = columnHeader.GetDataSize();
16     bool isConstant = columnHeader.IsConstant();
17
18     // 상수값을 가진 ColumnHeader인 경우

```

```

19  object value = null;
20  if (isConstant) {
21      value = ((ConstantColumnHeader) columnHeader).GetValue();
22  }
23  // ...
24  }

```

## 5.7 ColumnHeader를 이용한 DataSet의 열(column) 추가

DataSet에 열(column)을 추가하는 방법은 AddColumn(name, dataType, dataSize)를 통하거나, 직접 ColumnHeader를 생성하여 추가할 수 있습니다.

ColumnHeader를 이용한 DataSet의 열(column) 추가

```

1  // DataSet 생성
2  DataSet employees = new DataSet("employees");
3
4  // DataSet에 열(column) 추가
5  employees.AddColumn(new ColumnHeader("name", DataTypes.STRING, 8));
6  employees.AddColumn(new ColumnHeader("jobTitle", DataTypes.STRING, 16));
7  employees.AddColumn(new ColumnHeader("number", DataTypes.INT));
8  employees.AddColumn(new ColumnHeader("manager", DataTypes.BOOLEAN));
9
10 // 행(row) 추가
11 int row = employees.NewRow();
12
13 // 추가된 행(row)의 데이터 설정
14 employees.Set(row, "name", "John Jones");
15 employees.Set(row, "jobTitle", "developer");
16 employees.Set(row, "number", 1234);
17 employees.Set(row, "manager", false);
18
19 // 행(row) 추가
20 row = employees.NewRow();
21
22 // 추가된 행(row)의 데이터 설정

```

```

23 employees.Set(row, "name", "Tom Glover");
24 employees.Set(row, "jobTitle", "manager");
25 employees.Set(row, "number", 9876);
26 employees.Set(row, "manager", true);

```

## 5.8 ColumnHeader를 이용한 DataSet의 데이터 참조

때로는 DataSet의 열(column)에 대한 정보를 참조해야 하는 경우도 있습니다. 예를 들어, 각각의 열(column)에 대한 정보를 알지 못하는 경우나, DataSet의 데이터를 공통적으로 처리하여야 하는 경우일 것입니다.

DataSet의 getColumn(index)를 호출하여 열(column)의 갯수만큼 ColumnHeader를 참조하고, ColumnHeader 으로부터 식별자(name), 데이터 형식(dataType), 데이터 크기(dataSize) 등을 참조하여, 이에 따라 원하는 동작을 수행하면 될 것입니다.

ColumnHeader를 이용한 DataSet의 데이터 참조

```

1 PlatformData department = ...;
2
3 // DataSet을 식별자(id)를 이용하여 참조
4 DataSet employees = department.GetDataSet("employees");
5
6 // DataSet의 행(row)수만큼 순환
7 for (int i = 0; i < employees.GetRowCount(); i++) {
8     // DataSet의 열(column)수만큼 순환
9     for (int j = 0; j < employees.GetColumnCount(); j++) {
10         // DataSet으로부터 ColumnHeader 참조
11         ColumnHeader columnHeader = employees.GetColumn(j);
12         // 열(column)의 식별자(name) 참조
13         string name = columnHeader.GetName();
14         // 데이터의 형식(dataType)에 따른 구분
15         switch (columnHeader.GetDataType()) {
16             case DataTypes.STRING:
17                 string str = employees.GetString(i, name);
18                 // ...
19                 break;
20             case DataTypes.INT:
21                 int n = employees.GetInt(i, name);
22                 // ...
23                 break;

```



```

24     case DataTypes.BOOLEAN:
25         bool bool = employees.GetBoolean(i, name);
26         // ...
27         break;
28     default:
29         object obj = employees.GetObject(i, name);
30         // ...
31         break;
32     }
33 }
34 }

```

## 5.9 ConstantColumnHeader가 가지는 열(column)의 상수값

ConstantColumnHeader는 상수값을 가진 열(column)을 의미합니다.

즉, DataSet에 addConstantColumn(name, value)을 호출하여 열(column)을 추가하거나, ConstantColumnHeader를 생성하여 추가한다면, 해당 열(column)의 값은 행(row)의 위치(index)와 관계없이 일정한 상수값을 가지게 됩니다.

DataSet의 상수값을 가진 열(column) 추가

```

1  // DataSet 생성
2  DataSet employees = new DataSet("employees");
3
4  // DataSet에 일반 열(column) 추가
5  employees.AddColumn("name", DataTypes.STRING, 8);
6  employees.AddColumn("jobTitle", DataTypes.STRING, 16);
7
8  // DataSet에 상수값을 가진 열(column) 추가
9  employees.AddConstantColumn("city", "Seoul");
10 employees.AddColumn(new ConstantColumnHeader("company", "Tobesoft"));

```

## 5.10 DataSet의 원본 데이터와 변경된 데이터

DataSet은 데이터가 추가, 변경, 삭제된 경우 변경된 상태와 변경 이전의 원본 데이터를 저장합니다. 데이터가 변경되는 경우에는 원본 데이터를 별도로 저장하고, 현재 데이터를 변경하며, 삭제되는 경우에는 현재 데이터에서는 삭제되지만, 별도의 삭제된 데이터에 저장됩니다. 변경된 상태는 행(row) 단위로 저장되며, DataSet의 `getRowType(index)`를 호출하여 현재의 상태를 확인할 수 있습니다.

상수값	설 명
<code>DataSet.ROW_TYPE_NORMAL</code>	일반적인 행(row)
<code>DataSet.ROW_TYPE_INSERTED</code>	추가된 행(row)
<code>DataSet.ROW_TYPE_UPDATED</code>	변경된 행(row), 원본 데이터 존재할 수 있음
<code>DataSet.ROW_TYPE_DELETED</code>	삭제된 행(row), 다른 데이터와는 별도로 저장됨

저장 여부는 DataSet의 `startStoreDataChanges()`를 호출하여 활성화시키고, `stopStoreDataChanges()`를 통하여 저장을 중지하며, `startStoreDataChanges()`를 호출하는 시점의 데이터를 기준 데이터로 설정됩니다.

`startStoreDataChanges()`가 호출되면 이전에 저장된 원본 또는 삭제된 데이터는 삭제되므로, 데이터 유지가 필요한 경우 `startStoreDataChanges(true)`를 호출하여야 합니다. 반대로 `stopStoreDataChanges()`가 호출되면 이전에 저장된 원본 또는 삭제된 데이터는 보존되므로, 보존을 원하지 않는 경우 `stopStoreDataChanges(false)`를 호출합니다.

또한, 각각의 상태에 따른 데이터는 다음 메소드를 통하여 참조할 수 있습니다.

- 현재 데이터 : `getObject(rowIndex, columnIndex)` 등
- 변경 이전의 원본 데이터 : `getSavedData(rowIndex, columnIndex)` 등
- 삭제된 데이터 : `getRemovedData(rowIndex, columnIndex)` 등

주의할 점은 DataSet의 기본 설정은 변경되는 상태와 데이터를 저장하는 것입니다. 즉, DataSet의 생성과 동시에 `startStoreDataChanges()`가 자동적으로 호출되어 있는 것입니다.

이것이 의미하는 것은 사용자가 `startStoreDataChanges()`를 별도로 호출되지 않는 이상 DataSet에 저장되는 모든 데이터의 상태는 `ROW_TYPE_INSERTED`일 것입니다. 예를 들어, DataSet을 생성한 후에 데이터를 추가하고, 추가된 데이터를 다시 변경하더라도 데이터의 상태는 여전히 `ROW_TYPE_INSERTED`입니다. 이유는 DataSet의 기준 데이터는 생성한 직후가 되기 때문에, 즉 데이터가 없는 상태를 기준으로 본다면 데이터는 여전히 추가된 상태인 것입니다. 마찬가지로, 데이터를 추가한 후에 삭제하더라도 데이터의 상태는 `ROW_TYPE_DELETED`가 아니고, 데이터가 없는 상태를 기준으로 본다면 어떤 변경도 없었던 것입니다.

따라서, DataSet의 추가, 변경, 삭제된 상태와 데이터가 필요한 경우 적절한 시점에 `startStoreDataChanges()`가 호출되어야 합니다. 물론 DataSet을 생성하지 않고, 통신을 통해 전달받고, 이 시점을 기준 데이터로 본다면 굳이 호출하지 않아도 됩니다.

추가, 변경, 삭제된 DataSet의 행(row)

```

1 PlatformData department = ...;
2
3 // DataSet을 식별자(id)를 이용하여 참조
4 DataSet employees = department.GetDataSet("employees");
5
6 // 변경 정보 저장 시작
7 employees.StartStoreDataChanges();
8
9 // DataSet의 데이터 추가, 변경, 삭제 수행
10 ...
11 // 변경 정보 저장 중지
12 employees.StopStoreDataChanges();
13
14 // DataSet의 행(row)수만큼 순환
15 for (int i = 0; i < employees.GetRowCount(); i++) {
16     // 행(row)의 상태 참조
17     int rowType = employees.GetRowType(i);
18     if (rowType == DataSet.ROW_TYPE_NORMAL) {
19         // 일반적인 행(row)인 경우
20         object name = employees.GetObject(i, "name");
21         // ...
22     } else if (rowType == DataSet.ROW_TYPE_INSERTED) {
23         // 추가된 행(row)인 경우
24         object name = employees.GetObject(i, "name");
25         // ...
26     } else if (rowType == DataSet.ROW_TYPE_UPDATED) {
27         // 변경된 행(row)인 경우
28         object name = employees.GetObject(i, "name");
29         object savedName = employees.GetSavedData(i, "name");
30         // ...
31     } else {
32         // 발생 않음
33     }
34 }
35 for (int i = 0; i < employees.GetRemovedRowCount(); i++) {
36     // 삭제된 행(row)인 경우
37     object removedName = employees.GetRemovedData(i, "name");
38     // ...
39 }

```

## 6.

---

# com.nexacro.dotnet.xapi.tx (C#)

X-API의 데이터 통신을 수행합니다.

넥사크로 또는 PlatformData를 이용하는 모든 클라이언트와 데이터 송수신을 수행합니다. 데이터 통신의 대부분의 경우 HTTP 상에서 수행되며, XML 등의 형식으로 변환된 후 송수신됩니다. 주요 클래스는 HttpPlatformRequest, HttpPlatformResponse 등입니다.

## 6.1 시작하기

다음은 X-API를 이용하여 데이터를 송수신하는 간단한 예제입니다.

X-API를 이용한 예제

```
1  using Com.Nexacro.Dotnet.Xapi.Data;
2  using Com.Nexacro.Dotnet.Xapi.Tx;
3  using System;
4  public partial class XAPI_TEST : System.Web.UI.Page
5  {
6      protected void Page_Load(object sender, EventArgs e)
7      {
8          // HttpRequest를 이용하여 PlatformRequest 생성
9          PlatformRequest req = new PlatformRequest(Request.InputStream);
10
11         // 데이터 수신
12         req.ReceiveData();
13
14         // 수신받은 데이터 획득
15         PlatformData reqData = req.GetData();
16         VariableList reqVarList = reqData.GetVariableList();
17     }
```

```

18     // 부서명 획득
19     string name = reqVarList.GetString("name");
20
21     // 송신할 데이터 생성
22     PlatformData resData = new PlatformData();
23     VariableList resVarList = resData.GetVariableList();
24
25     // 부서별 인원을 저장할 DataSet 생성
26     DataSet employees = new DataSet("employees");
27
28     // DataSet에 열(column) 추가
29     employees.AddColumn(new ColumnHeader("name", DataTypes.STRING, 8));
30     employees.AddColumn(new ColumnHeader("jobTitle", DataTypes.STRING));
31     employees.AddColumn(new ColumnHeader("number", DataTypes.INT));
32     employees.AddColumn(new ColumnHeader("manager", DataTypes.BOOLEAN));
33
34     // 부서별 인원 데이터 추가
35     if ("R&D Center".Equals(name)) {
36         // 행(row) 추가
37         int row = employees.NewRow();
38         // 추가된 행(row)의 데이터 설정
39         employees.Set(row, "name", "John Jones");
40         employees.Set(row, "jobTitle", "developer");
41         employees.Set(row, "number", 1234);
42         employees.Set(row, "manager", false);
43         // ...
44         // 정상 수행
45         resData.AddDataSet(employees);
46         resVarList.Add("ERROR_CODE", 200);
47     } else if ("Quality Assurance".Equals(name)) {
48         // 행(row) 추가
49         int row = employees.NewRow();
50         // 추가된 행(row)의 데이터 설정
51         employees.Set(row, "name", "Tom Glover");
52         employees.Set(row, "jobTitle", "manager");
53         employees.Set(row, "number", 9876);
54         employees.Set(row, "manager", true);
55         // ...
56         // 정상 수행
57         resData.AddDataSet(employees);
58         resVarList.Add("ERROR_CODE", 200);

```

```

59         } else {
60             // 오류 발생
61             resVarList.Add("ERROR_CODE", 500);
62         }
63         // HttpServletResponse를 이용하여 PlatformResponse 생성
64         PlatformResponse res = new PlatformResponse(Response.OutputStream);
65         res.SetData(resData);
66         // 데이터 송신
67         res.SendData();
68     }
69 }

```

## 6.2 데이터 송수신

넥사크로와의 데이터 통신은 대부분의 경우 HTTP 상에서 수행되며, 특정 형식으로 변환된 후 송수신됩니다.

송수신 형식(contentType)은 데이터가 송수신되기 위해 객체에서 특정 형식의 데이터(stream)으로 변환되는 것을 의미하며, 프로토콜 형식(protocolType)은 데이터의 압축, 암호화 등을 수행하는 것을 의미합니다.

송수신 형식(contentType)과 프로토콜 형식(protocolType) 등 데이터 변환에 대한 주요 인터페이스는 다음과 같습니다.

인터페이스명	설 명
DataSerializer	PlatformData를 특정 형식의 데이터(stream)으로 변환
DataDeserializer	특정 형식의 데이터(stream)를 PlatformData로 변환
ProtocolEncoder	데이터(stream)의 압축, 암호화 등을 수행
ProtocolDecoder	압축, 암호화 등이 적용된 데이터(stream)를 압축 해제, 복호화 등을 수행

## 6.3 데이터 송수신의 내부 흐름

### 데이터 송신 흐름

1. Server에서 데이터를 PlatformData에 저장한 후 송신함
2. DataSerializer에 의해 PlatformData가 특정 형식의 데이터(stream)로 변환됨

3. ProtocolEncoder에 의해 압축, 암호화 등이 적용됨
4. HTTP 상으로 데이터(stream)가 송신됨

## 데이터 수신 흐름

1. HTTP 상에서 압축, 암호화 등이 적용된 데이터(stream)를 수신받음
2. ProtocolDecoder에 의해 압축 해제, 복호화 등이 수행됨
3. DataDeserializer에 의해 특정 형식의 데이터(stream)가 PlatformData로 변환됨
4. Client에서 데이터가 저장된 PlatformData를 수신받음

## 6.4 Server 상의 HTTP 데이터 통신

넥사크로와 HTTP 상에서 데이터 통신을 하기 위해서는 PlatformRequest와 PlatformResponse를 이용하여 cs 파일을 작성합니다.

PlatformRequest와 PlatformResponse는 System.Web.HttpRequest와 System.Web.HttpResponse를 이용하여 HTTP 통신을 수행하며, PlatformRequest는 넥사크로로 데이터(stream)를 수신받은 후 PlatformData으로 변환하고, 그와는 반대로 PlatformResponse는 PlatformData를 데이터(stream)으로 변환한 후 넥사크로로 송신합니다.

HttpServletRequest으로부터 데이터 수신

```
1 // System.Web.HttpRequest를 이용하여 PlatformRequest 생성
2 PlatformRequest req = new PlatformRequest(Request.InputStream);
3 // 데이터 수신
4 req.ReceiveData();
5 PlatformData data = req.GetData();
```

HttpServletResponse으로 데이터 송신

```
1 PlatformData data = ...;
2 // System.Web.HttpResponse를 이용하여 PlatformResponse 생성
3 PlatformResponse res = new PlatformResponse(Response.OutputStream);
4 res.SetData(data);
5 // 데이터 송신
6 res.SendData();
```

echo.cs 예제

```

1  using Com.Nexacro.Dotnet.Xapi.Data;
2  using Com.Nexacro.Dotnet.Xapi.Tx;
3  using System;
4  public partial class XAPI_TEST : System.Web.UI.Page
5  {
6      protected void Page_Load(object sender, EventArgs e)
7      {
8          PlatformRequest req = new PlatformRequest(Request.InputStream);
9          req.ReceiveData();
10         PlatformData reqData = req.GetData();
11         PlatformResponse res = new PlatformResponse(Response.OutputStream);
12         res.SetData(data);
13         res.SendData();
14     }
15 }

```

## 6.5 데이터 송수신 형식

송수신 형식은 [데이터 송수신](#)에서 언급되었듯이 데이터가 송수신되기 위해 객체에서 특정 형식의 데이터(stream)으로 변환되는 형식을 의미하며, 기 구현된 형식은 다음과 같습니다.

상수값	설 명
PlatformType.CONTENT_TYPE_XML	Platform에서 정의된 XML 형식
PlatformType.CONTENT_TYPE_BINARY	Platform에서 정의된 바이너리 형식
PlatformType.CONTENT_TYPE_SSV	Platform에서 정의된 SSV 형식

XML 형식으로 데이터를 수신하고, Binary 형식으로 데이터를 송신하는 예제

```

1  // XML 형식으로 데이터 수신
2  PlatformRequest req = new PlatformRequest(Request.InputStream, PlatformType.CONTENT_TYPE_XML);
3  req.ReceiveData();
4  PlatformData data = req.GetData();
5
6  // Binary 형식으로 데이터 송신
7  PlatformResponse res = new PlatformResponse(Response.OutputStream, PlatformType.CONTENT_TYPE_BINARY);

```



```

8  res.SetData(data);
9
10 res.SendData();

```

만약에 사용자에게 의해 정의된 송수신 형식이 아닌 기 제공되는 송수신 형식인 경우에는 다음과 같이 PlatformRequest에 별도의 송수신 형식을 지정하지 않아도, 내부에서 자동으로 송수신 형식을 판단하여 처리합니다.

송수신 형식을 자동으로 판단하여 데이터 수신

```

1  // 송수신 형식을 자동으로 판단하여 데이터 수신
2  PlatformRequest req = new PlatformRequest(Request.InputStream);
3  req.ReceiveData();
4
5  PlatformData data = req.GetData();

```

## 6.6 데이터 프로토콜 형식

프로토콜 형식은 데이터 송수신에서 언급되었듯이 데이터의 압축, 암호화 등을 수행하는 것을 의미하며, 기 구현된 형식은 다음과 같습니다.

상수값	설 명
PlatformType.PROTOCOL_TYPE_ZLIB	ZLIB 방식으로 압축

ZLIB 방식으로 압축하여 데이터 송신

```

1  // 데이터 수신
2  PlatformRequest req = new PlatformRequest(Request.InputStream);
3  req.ReceiveData();
4  PlatformData data = req.GetData();
5
6  // ZLIB 방식으로 압축하여 데이터 송신
7  PlatformResponse res = new PlatformResponse(Response.OutputStream, PlatformType.CONTENT_
TYPE_BINARY);
8  res.AddProtocolType(PlatformType.PROTOCOL_TYPE_ZLIB);
9  res.SetData(data);
10
11 res.SendData();

```

## 6.7 추가, 변경, 삭제된 데이터 송신

DataSet은 데이터가 추가, 변경, 삭제된 경우 변경된 상태와 변경 이전의 원본 데이터를 저장합니다. 자세한 정보는 [DataSet의 원본 데이터와 변경된 데이터](#)를 참조합니다.

PlatformResponse는 위와 같은 데이터를 저장 방식(saveType)에 따라 구분하여 전송합니다.

상수값	설 명
DataSet.SAVE_TYPE_NONE	미설정
DataSet.SAVE_TYPE_ALL	현재의 데이터와 추가, 변경, 삭제된 모든 데이터 저장 또는 전송
DataSet.SAVE_TYPE_NORMAL	현재의 데이터만 저장 또는 전송
DataSet.SAVE_TYPE_UPDATED	추가, 변경된 데이터만 저장 또는 전송
DataSet.SAVE_TYPE_DELETED	삭제된 데이터만 저장 또는 전송
DataSet.SAVE_TYPE_CHANGED	추가, 변경, 삭제된 데이터만 저장 또는 전송

저장 방식(saveType)은 PlatformData와 DataSet이 가지고 있으며, DataSet의 저장 방식이 우선으로 적용되고, DataSet의 저장 방식이 DataSet.SAVE\_TYPE\_NONE인 경우 PlatformData의 저장 방식이 적용됩니다.

만약에 PlatformData의 저장 방식도 DataSet.SAVE\_TYPE\_NONE인 경우 기본값 DataSet.SAVE\_TYPE\_NORMAL이 적용됩니다.

## 7.

# NLog config

## 7.1 Web.config에 직접 설정

Web.config

```
1 <configSections>
2   <section name="nlog" type="NLog.Config.ConfigSectionHandler, NLog"/>
3 </configSections>
4 <nlog throwConfigExceptions="true" xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
5   <targets>
6     <target name="LogFile"
7       xsi:type="File"
8       layout="${longdate} [{uppercase:${level:padding=-5}}] ${message} ${exception:
format=tostring}"
9       fileName="${basedir}Logs\\${var:runtime}\\${date:format=yyyyMMdd}.log"
10      encoding="UTF-8"
11      archiveFileName="${basedir:processDir=true}Logs\\archives\\${var:runtime}\\archive
.{#}.log"
12      archiveEvery="Day"
13      archiveNumbering="Rolling"
14      maxArchiveFiles="7"
15      header="[Start Logging]"
16      footer="[End Logging]${newline}"/>
17   </targets>
18   <rules>
19     <logger name="*" minlevel="Info" writeTo="LogFile"/>
20   </rules>
21 </nlog>
```

## App.config나 web.config 파일에 nlog를 embedded 할 경우 section을 지정

```
<configSections>
  <section name="nlog" type="NLog.Config.ConfigSectionHandler, NLog">
</configSections>
```

## 로그 생성 중 예외가 발생했을 경우 출력 옵션

```
<nlog throwConfigExceptions="true" ..... >
```

## 변수 설정

```
<variable name="logDir" value="${basedir}/logs/">
```

## target 지정 시 출력 layout을 CSV, JSON, XML 혹은 File 형태로 지정 가능

```
<targets>
<target name="LogFile"
xsi:type="File"
  layout="${longdate} [{uppercase:${level:padding=-5}}] ${message} ${exception:format=
tostring}"
  fileName="${basedir}Logs\${var:runtime}\${date:format=yyyyMMdd}.log"
  encoding="UTF-8"
  archiveFileName="${basedir:processDir=true}Logs\archives\${var:runtime}\archive.{#}.log
"
  archiveEvery="Day"
  archiveNumbering="Rolling"
  maxArchiveFiles="7"
  header="[Start Logging]"
  footer="[End Logging]${newline}">
</targets>
```

## Rule 설정

```
<rules>  
  <logger name="*" minlevel="Info" writeTo="LogFile">  
<logger name="Com.Nexacro.Dotnet.Xapi.Tx.*" minlevel="Info" writeTo="LogFile">  
</rules>
```

## 7.2 환경 변수 및 XML 설정 참고



Layout 환경 변수

<https://nlog-project.org/config/?tab=layout-renderers>

XML 설정

<https://github.com/nlog/NLog/wiki/Configuration-file#variables>

## 파트 III.

---

nexacro-xeni

## 8.

### 설치

nexacro-xeni는 서버모듈로 제공되며 클라이언트에서 파일 형태로 데이터 처리를 할 때 필요한 기능을 제공합니다. 파일에 포함된 데이터를 그리드 컴포넌트로 가져오거나 그리드에 연결된 데이터를 파일로 내보내는 기능을 처리합니다.

nexacro-xeni는 아래와 같은 파일 형태와 기능을 지원합니다.

파일 형태	확장자	기능
Microsoft Excel 97-2003	xls	export / import
Microsoft Excel 2007/2010	xlsx	export / import
CSV format file	csv	import
HancomOffice Hancell 2010	cell	export
HancomOffice Hancell 2014	cell	export / import

### 8.1 설치

nexacro-xeni는 자바 기반 서버모듈로 제공되며 버전에 따라 아래와 같이 지원 환경 및 기능이 달라집니다.

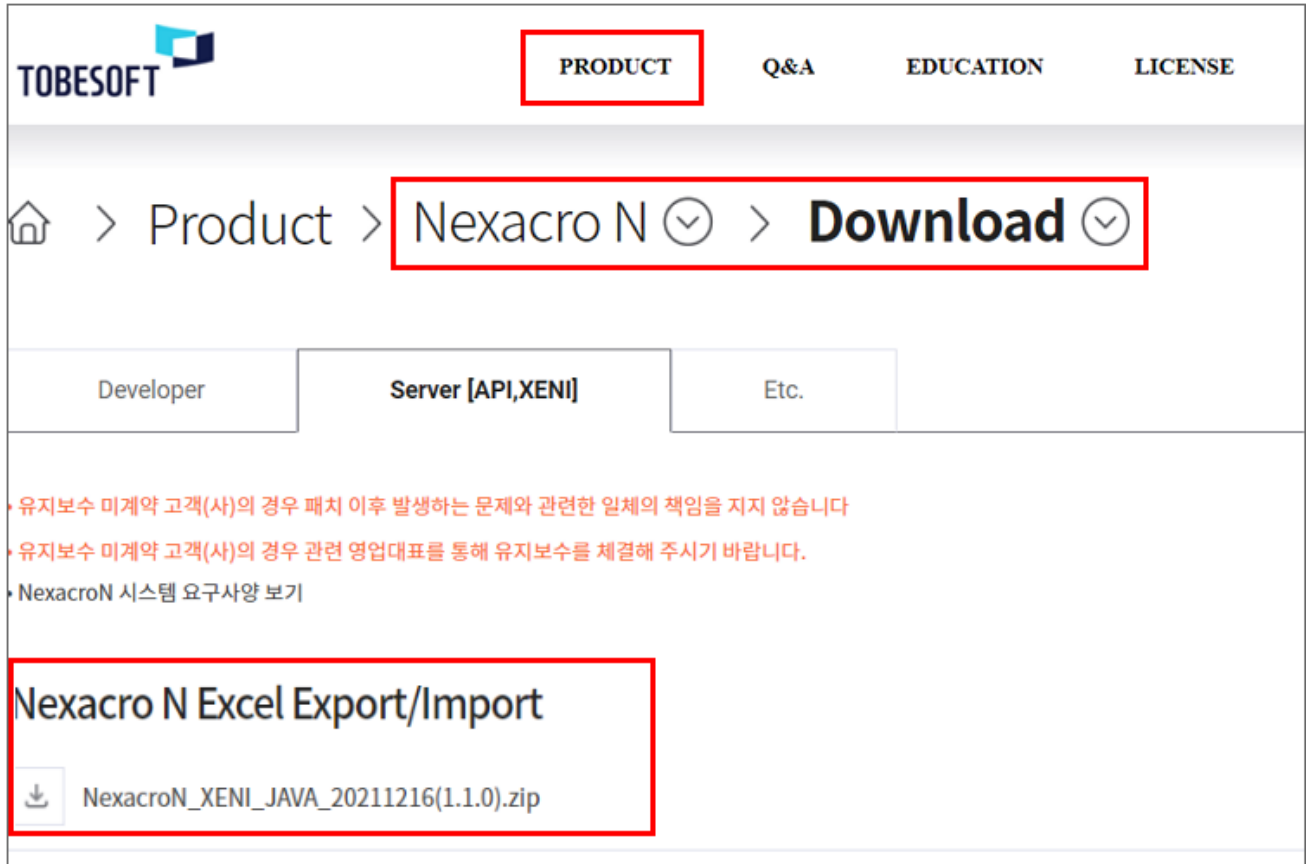
nexacro-xeni	JDK version	POI 라이브러리 버전	유지보수
1.x.x	1.8 이상	POI 4.1.2	신규, 변경 포함



1.4.1 이후 버전은 Jakarta EE 스펙으로 구현된 WAS에서 사용할 수 있는 nexacro-xeni를 제공합니다. 파일명에 "\_jakarta\_"가 포함된 파일을 내려받아 사용하세요.  
<https://jakarta.ee/compatibility/>

기술지원 사이트 메뉴 PRODUCT > Nexacro N > Download > Server [API, XENI]에서 war 파일이 포함된 압축 파일을 내려받을 수 있습니다.

[http://support.tobesoft.co.kr/Support/?menu=Download\\_N](http://support.tobesoft.co.kr/Support/?menu=Download_N)



제공된 war(Web application ARchive) 파일을 직접 배치하거나 war 파일의 압축을 풀어 필요한 파일을 WAS의 / WEB-INF/lib 디렉터리 또는 정의된 클래스 경로에 복사해 사용할 수 있습니다.



war 파일을 직접 배치하지 않고 압축을 풀어 복사하는 경우에는 라이브러리 버전 차이로 문제가 발생할 수 있습니다.



nexacro-xeni는 Apache POI 라이브러리를 사용합니다. 자세한 Apache POI 설명은 아래 링크를 참고해주세요.

<https://poi.apache.org/>



nexacro-xeni가 동작하려면 X-API가 설치되어 있어야 합니다. [설치](#) 항목을 참고해주세요.

기존 컨텍스트에 파일을 복사한 경우에는 아래 내용을 web.xml 파일에 추가해주어야 합니다.



```

<servlet>
  <servlet-name>XExportImport</servlet-name>
  <servlet-class>com.nexacro.java.xeni.services.GridExportImportServlet</servlet-class>
</servlet>
<servlet>
  <servlet-name>XImport</servlet-name>
  <servlet-class>com.nexacro.java.xeni.services.GridExportImportServlet</servlet-class><class>
</servlet>

<servlet-mapping>
  <servlet-name>XExportImport</servlet-name>
  <url-pattern>/XExportImport</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>XImport</servlet-name>
  <url-pattern>/XImport</url-pattern>
</servlet-mapping>

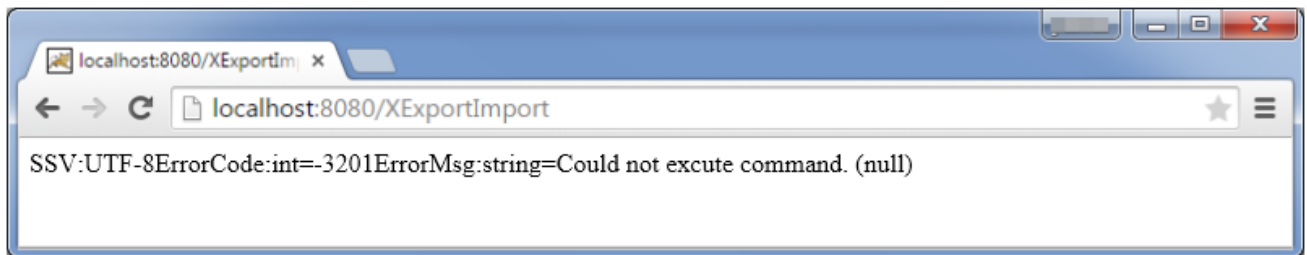
<context-param>
  <param-name>export-path</param-name>
  <param-value>/export</param-value>
</context-param>
<context-param>
  <param-name>import-path</param-name>
  <param-value>/import</param-value>
</context-param>
<context-param>
  <param-name>monitor-enabled</param-name>
  <param-value>true</param-value>
</context-param>
<context-param>
  <param-name>monitor-cycle-time</param-name>
  <param-value>30</param-value>
</context-param>
<context-param>
  <param-name>file-storage-time</param-name>
  <param-value>10</param-value>
</context-param>

```

## 8.2 설치 확인

정상적으로 배치 또는 복사되었다면 아래 URL을 웹브라우저에서 열어 메시지가 정상적으로 나타나는지 확인합니다.

<http://127.0.0.1:8080/XExportImport>



URL은 설치 경로에 따라 달라질 수 있습니다. war 파일을 직접 배치한 경우 URL은 아래와 같습니다.  
<http://127.0.0.1:8080/nexacro-xeni-java/XExportImport>

## 8.3 주요 설정

서블릿 컨텍스트에 파라미터로 설정된 주요 항목은 아래와 같습니다.

이름	기본값	설명
export-path	/export	내려받을 파일을 임시로 저장하는 경로를 지정합니다.
import-path	/import	그리드 컴포넌트로 가져올 파일을 임시로 저장하는 경로를 지정합니다. monitor-enabled 설정값이 true인 경우에는 데이터 처리 후 바로 삭제합니다.
monitor-enabled	true	파일 매니저 실행 여부를 지정합니다. false로 지정한 경우에는 파일 매니저가 동작하지 않고 임시 파일을 삭제하지 않습니다.
monitor-cycle-time	30	임시 저장된 내려받을 파일을 삭제하는 주기를 '분' 단위로 설정합니다. 해당 주기로 파일 매니저를 실행해 삭제할 파일을 체크하고 삭제 처리합니다.
file-stroage-time	10	임시로 저장한 내려받을 파일을 보관하는 시간을 '분' 단위로 설정합니다. 파일 매니저 실행 시 생성 후 지정된 시간이 지난 파일은 삭제합니다.



자세한 내용은 배포되는 war 파일에 포함된 readme.txt 파일을 참고하세요.

## 8.4 예제

간단한 데이터 처리 예제를 설명합니다. 세부적인 속성이나 메소드, 이벤트는 도움말에서 아래 항목을 참고하세요.

Objects > Misc. Objects > ExcelExportObject

### 8.4.1 Export

그리드 컴포넌트에 바인딩된 데이터를 서버로 전송해 파일로 생성하고 생성된 파일을 내려받습니다. 아래 예제에서는 Grid00 컴포넌트에 바인딩된 데이터를 파일로 생성하고 내려받습니다.

```
this.Button00_onclick = function(obj:Button, e:nexacro.ClickEventInfo)
{
    this.exportObj = new ExcelExportObject();
    var ret = this.exportObj.addExportItem(nexacro.ExportItemTypes.GRID,
        this.Grid00, "Sheet1!A1");
    this.exportObj.set_exporturl("http://localhost:8080/XExportImport");
    this.exportObj.exportData();
}
```

### 8.4.2 Import

사용자가 가지고 있는 파일을 서버로 전송해 데이터셋으로 변환합니다. 사용자는 변환된 데이터셋을 내려받아 업무를 처리합니다.

```
this.Button00_onclick = function(obj:Button, e:nexacro.ClickEventInfo)
{
    this.importObj = new ExcelImportObject("Import00", this);
    this.importObj.addEventHandler("onsuccess", this.Import_onsuccess, this);
    this.importObj.set_importurl("http://localhost:8080/XExportImport");
}
```

```
    this.importObj.importData("", "output=ds", "Dataset00=ds");  
}  
  
this.Import_onsuccess = function(obj,e)  
{  
    this.Grid00.createFormat();  
}
```

## 9.

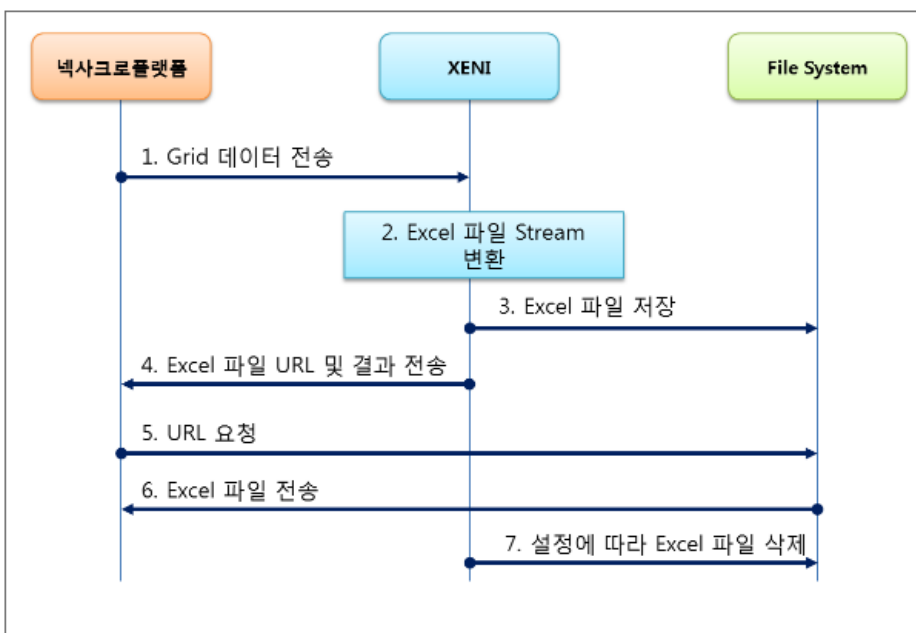
# export 기능

export 처리 과정과 실행 샘플을 설명합니다.

## 9.1 Export 처리

1. 애플리케이션은 export를 위한 data(command, export data, style)를 일정 크기로 분할하고, 분할된 data(chunked data)를 nexacro-xeni 서버에 순차적으로 전송합니다.
2. nexacro-xeni 서버는 전송 받은 데이터를 command에 따라 저장될 파일 stream으로 구성합니다.
3. 지정된 경로에 임의의 폴더를 생성하고 stream을 지정된 이름의 파일로 생성합니다.
4. 생성된 파일의 url을 애플리케이션에 전송해 파일을 내려받을 수 있도록 합니다.
5. url을 요청합니다.
6. Excel 파일을 전송합니다.
7. 생성된 파일은 설정에 따라 관리/삭제됩니다.

그림 9-1 nexacro-xeni에서의 Excel Export 실행 단계

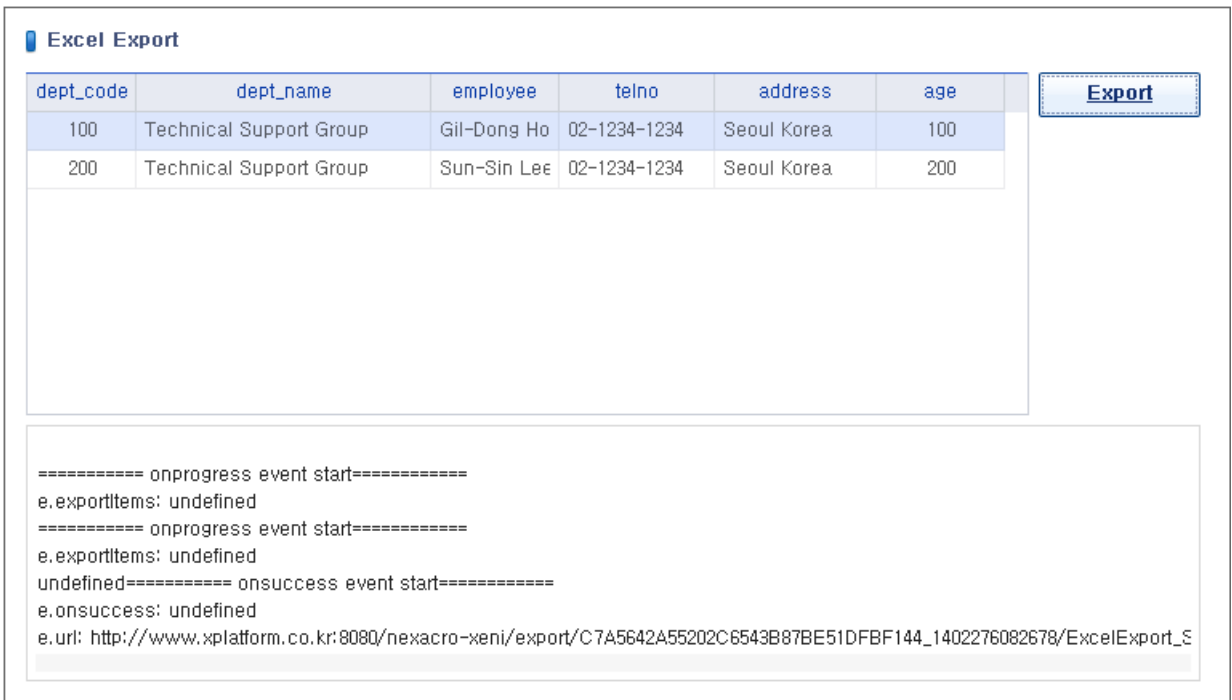


## 9.2 실행 샘플

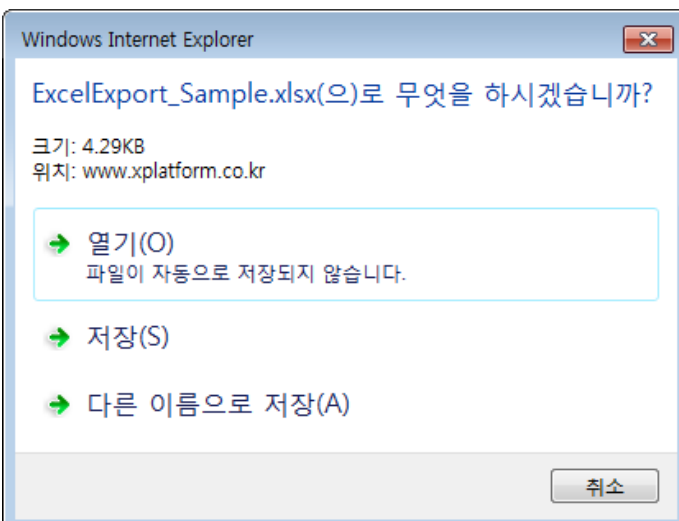
nexacro platform grid component 의 data 를 Excel 파일로 export 하는 방법을 설명 합니다.

### 9.2.1 nexacro platform 화면

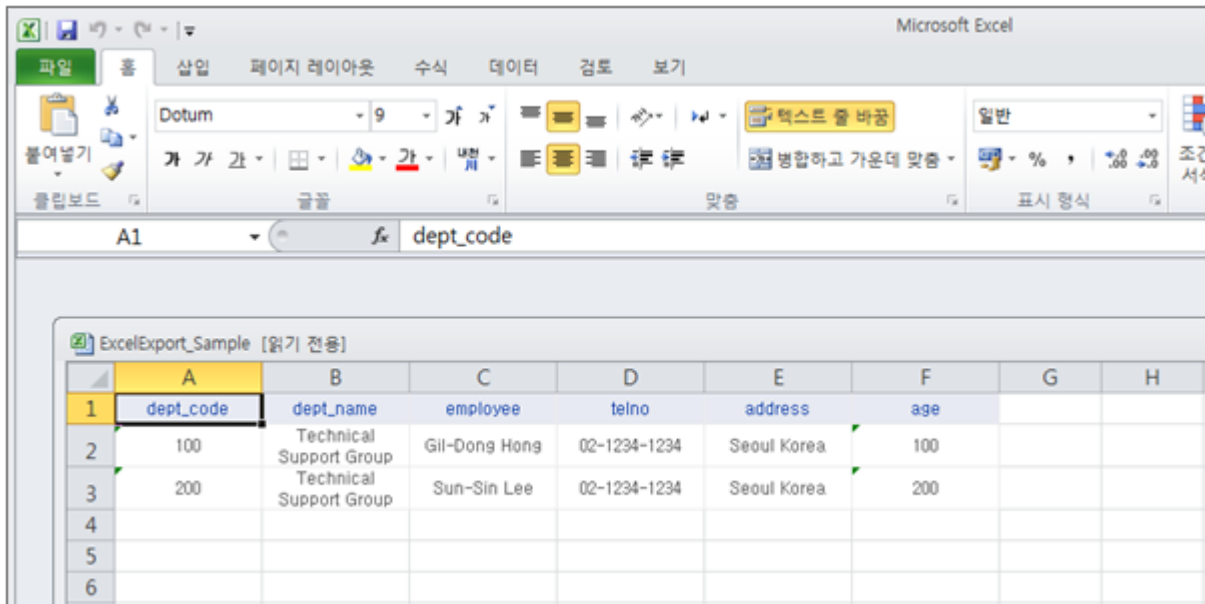
- ① 아래 샘플 그림에서 export 버튼을 클릭하면 grid 의 내용이 nexacro-xeni 로 분할 전송 됩니다.



- ② nexacro-xeni 에서 작업이 완료되면 아래와 같은 대화 상자가 표시 됩니다.



- 3 다운로드 된 파일을 확인 합니다.



## 9.2.2 nexacro platform 소스

```

1  this.Button00_onclick = function(obj:Button, e:nexacro.ClickEventInfo)
2  {
3      this.url = "http://127.0.0.1:8080/nexacro-xeni/XExportImport ";
4
5      this.exportObj = new ExcelExportObject();
6      this.exportObj.addEventHandler("onprogress", this.ExcelExportObject00_onprogress, this
7  );
8      this.exportObj.addEventHandler("onsuccess", this.ExcelExportObject00_onsuccess, this);
9      this.exportObj.addEventHandler("onerror", this.ExcelExportObject00_onerror, this);
10
11     var ret = this.exportObj.addExportItem(nexacro.ExportItemTypes.GRID, this.gd_excel, "
12     Sheet1!A1");
13     this.exportObj.set_exportmessageprocess("%d [ %d / %d ]");
14     this.exportObj.set_exportuitype("exportprogress");
15     this.exportObj.set_exporteventtype("itemrecord");
16     this.exportObj.set_exporttype(nexacro.ExportTypes.EXCEL2007);
17     this.exportObj.set_exportfilename("ExcelExport_Sample");
18     this.exportObj.set_exporturl(this.url);
19     this.exportObj.exportData();
20 }

```

## 9.2.3 ExcelExportObject Event

export 처리 과정과 결과를 이벤트를 통해 확인 할 수 있습니다.

Event Name	Description
onerror	Export 수행 중 오류가 발생 되었을 때 발생 되는 이벤트
onprogress	Export 수행 중 진행 상태 별로 발생하는 이벤트
onsuccess	Export 작업이 완료 되었을 때 발생 되는 이벤트

## 9.3 오류 대응

### 9.3.1 파일 대화 상자가 열리지 않거나 파일이 깨져서 표시되는 경우

Export 작업 완료 후 파일 대화 상자가 열리지 않고 브라우저 상에 파일 내용이 깨진 상태로 표시 될 경우 web.xml 에 mime-mapping 을 추가한 후 WAS 를 재 시작 합니다.

```

43     <param-value>30</param-value> ↵
44 </context-param> ↵
45 ↵
46 <context-param> ↵
47     <param-name>file-storage-time</param-name> ↵
48     <param-value>10</param-value> ↵
49 </context-param> ↵
50 ↵
51 <mime-mapping> ↵
52     <extension>xls</extension> ↵
53     <mime-type>application/octet-stream</mime-type> ↵
54 </mime-mapping> ↵
55 <mime-mapping> ↵
56     <extension>xlsx</extension> ↵
57     <mime-type>application/octet-stream</mime-type> ↵
58 </mime-mapping> ↵
59 </web-app> ↵

```



# 10.

## import 기능

import 처리 과정과 실행 샘플을 설명합니다.

### 10.1 Import 처리

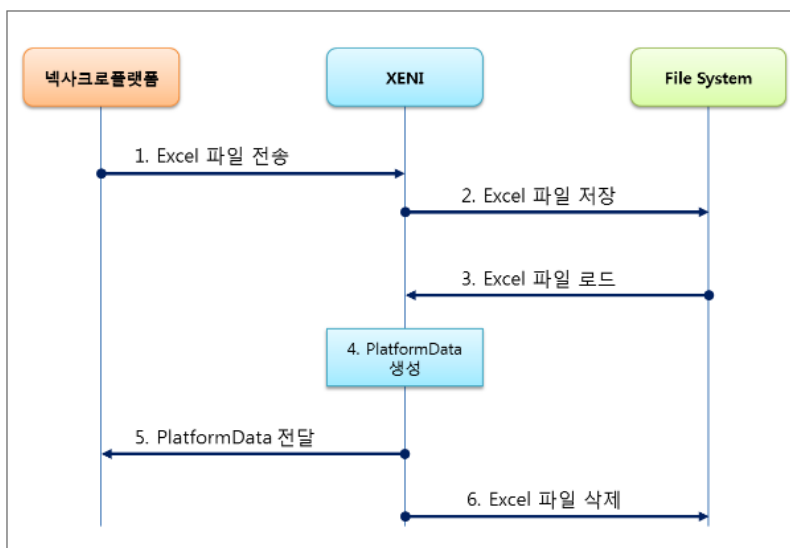
1. 애플리케이션에서 import할 파일과 import command를 전송받습니다.
2. 지정된 경로에 임시 폴더를 생성하고 전송받은 파일을 저장합니다.
3. 저장된 파일을 읽어 들인 후 command에 따라 data를 추출합니다.
4. dataset (PlatformData)으로 변환합니다.
5. 데이터를 애플리케이션에 전송합니다.

Excel의 경우 command에 따라 sheet목록을 dataset으로 변환하거나 여러 개의 sheet data를 복수의 dataset으로 변환해 전송할 수 있습니다.

nexacro-xeni에서 생성되는 dataset의 datatype은 모두 string으로 바인딩 되는 dataset에서 layout과 data type을 지정해 주어야 합니다.

6. 생성된 파일은 작업이 완료된 후 바로 삭제됩니다.

그림 10-1 nexacro-xeni에서의 Excel Import 실행 단계





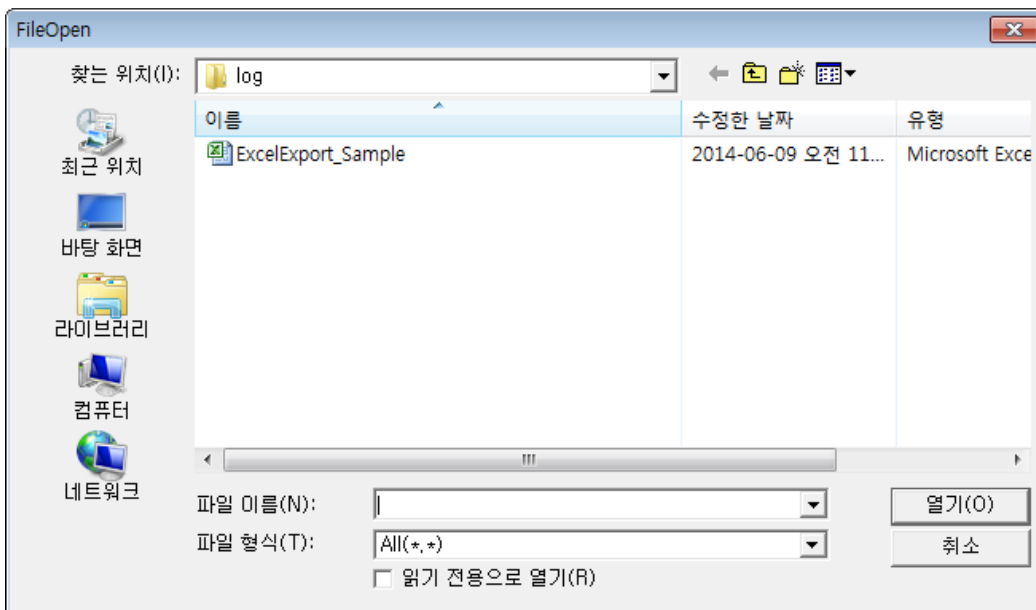
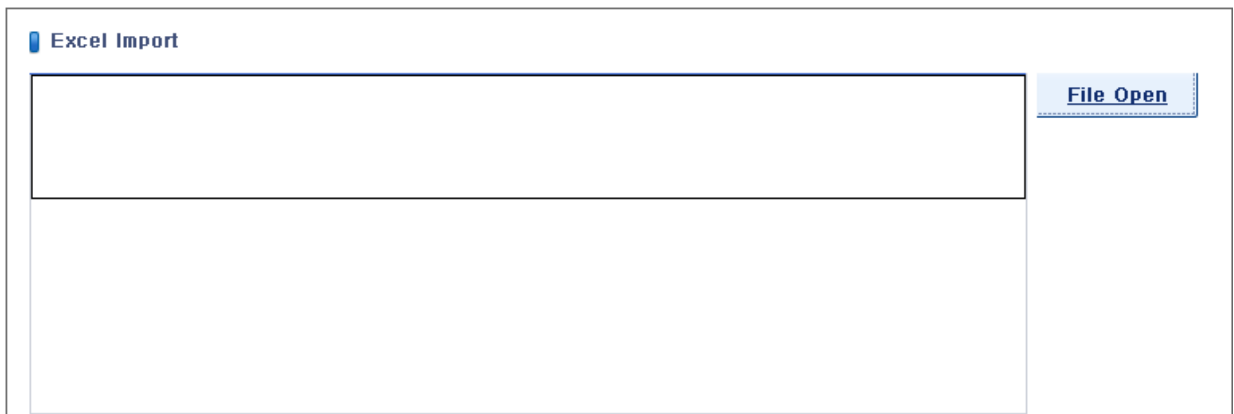
import 시 server mode로 동작할 경우 서버에 이미 저장되어 있는 파일을 사용하며 해당 파일은 작업 완료 후에도 삭제되지 않습니다. import file mode는 nexacro help를 참고하십시오.

## 10.2 실행 샘플

excel 파일을 nexacro-xeni 로 upload 하여 data 를 추출, Grid 컴포넌트에 표시하는 방법을 설명합니다.

### 10.2.1 nexacro platform 화면

- 1 아래 그림에서 File Open 버튼을 클릭하면 파일 선택 대화 상자가 표시됩니다.



- 2 선택된 파일이 nexacro-xeni에 upload 되어 import된 결과가 grid에 표시 됩니다.

Excel Import

dept_code	dept_name	employee	telno	address	age
100	Technical Support Group	Gil-Dong Ho	02-1234-1234	Seoul Korea	100
200	Technical Support Group	Sun-Sin Lee	02-1234-1234	Seoul Korea	200

File Open

```

importurl set :http://www.xplatform.co.kr:8080/nexacro-xeni/XImport
importData() Method
undefined===== onsuccess event start=====
e.eventid: onsuccess
e.fromobject: [object ExcelImportObject]
e.fromreferenceobject: [object ExcelImportObject]
e.url: F:/Was/apache-tomcat-6.0.35/webapps/Next/nexacro-xeni/import/F401A16C0FFCC6BC8C4B7B831207DD88_140228226304E

```

## 10.2.2 nexacro platform 소스

ExcelImportObject.importData에 대한 설명은 nexacro platform help를 참조해 주십시오.

```

1  this.Button00_onclick = function(obj:Button, e:nexacro.ClickEventInfo)
2  {
3      this.url = "http://127.0.0.1:8080/nexacro-xeni/XExportImport";
4
5
6      this.importObj = new ExcelImportObject("Import00",this);
7      this.importObj.set_importtype(nexacro.ImportTypes.EXCEL);
8      this.importObj.addEventHandler("onsuccess", this.Import00_onsuccess, this);
9      this.importObj.addEventHandler("onerror", this.Import00_onerror, this);
10     this.importObj.set_importurl(this.url);
11     this.importObj.importData("", "Body=Sheet1!A1:F6;output=ds", "ds_excel=ds");
12 }

```

## 10.2.3 ExcelImportObject Event

Import 처리 결과를 이벤트를 통해 확인 할 수 있습니다.

Event Name	Description
onerror	Import 수행 중 오류가 발생했을 때 발생하는 이벤트
onsuccess	Import 작업이 완료됐을 때 발생하는 이벤트

# 11.

## nexacro-xeni 확장 인터페이스

nexacro-xeni는 사용자가 Excel 파일 Export/Import 기능을 확장할 수 있게 확장 인터페이스를 지원합니다.

이번 장에서는 확장 인터페이스를 사용해 Excel 파일 Export/Import 기능을 확장하는 방법을 설명합니다.

### 11.1 개요

제공된 확장 인터페이스 구현을 통해 넥사크로플랫폼에서 전달받은 Grid 컴포넌트 데이터를 Excel 파일로 생성하기 전에 사용자 정의 기능을 추가할 수 있습니다. 또한, 넥사크로플랫폼에서 전달받은 Excel 파일을 Grid 컴포넌트 데이터로 변환하기 전에 추가적인 작업을 처리할 수 있습니다.

nexacro-xeni 확장 인터페이스를 이용해 아래와 같은 추가 기능을 구현할 수 있습니다. nexacro-xeni의 기본 실행 단계를 벗어나지 않는 범위에서 원하는 기능을 구현할 수 있습니다.

- DRM이 적용된 Excel 파일의 DRM 해제
- 생성된 Excel 파일에 DRM 적용
- Excel 파일을 데이터베이스에 저장하고 데이터베이스에서 읽어오는 기능 (파일 시스템 사용하지 않음)
- 기타 추가적인 사용자 로직

제품 버전에 따라 사용할 수 있는 nexacro-xeni 확장 인터페이스는 아래와 같습니다.

제품 버전	nexacro-xeni 확장 인터페이스
넥사크로플랫폼 14	XeniExtendBase
넥사크로플랫폼 17.1, N	XeniExcelDataStorageBase

nexacro-xeni 확장 인터페이스는 다음의 메소드를 제공합니다.

Type	Method
InputStream	loadTargetStream
String	saveImportStream
int	saveExportStream

Type	Method
Dataset	saveExportStream

해당 메소드는 확장 인터페이스를 상속받아서 구현할 수 있습니다. 구현된 인터페이스를 사용하기 위해서는 사용자가 구현한 Class를 해당 properties 파일에 등록하여 사용합니다.



Excel 파일을 PlatformData로 변환하거나 PlatformData를 Excel 파일로 변환하는 기능은 nexacro-x eni 내부에서 구현되어 있으며 사용자가 해당 기능을 변경하는 것은 허용되지 않습니다.

## 11.2 메소드

### 11.2.1 (InputStream) loadTargetStream

```
public InputStream loadTargetStream(String filepath)
```

Export/Import 기능을 처리하기 위해 저장된 Excel 파일을 스트림으로 반환하는 인터페이스입니다.

#### DRM 기능을 적용했을 경우

Import 시 넥사크로플랫폼에서 전달받아 임시로 저장된 Excel 파일의 DRM을 해제하고 스트림으로 반환하도록 구현합니다. 또는 DRM이 적용된 Excel 파일에 Export를 추가하는 경우에도 사용됩니다(여러 Grid 컴포넌트를 한 번에 Export 하는 경우).

아래는 DRM이 적용되지 않은 Excel 파일을 읽어 들이는 샘플입니다.

```
public InputStream loadTargetStream(String filepath) throws Exception {
    File file = new File(filepath);
    return new FileInputStream(file);
}
```

### 11.2.2 (String) saveImportStream

```
public String saveImportStream(VariableList varlist,
    InputStream in,
    String filepath)
```

Import 시 넥사크로플랫폼에서 전달받은 Excel 파일(InputStream)을 지정된 디렉터리(filepath)나 데이터베이스에 저장하고 filepath나 데이터베이스의 Key값을 반환하는 인터페이스입니다.

### DRM이 적용된 Excel 파일인 경우

DRM 해제 후 저장하도록 구현하거나 DRM이 적용된 상태로 저장 후 loadTargetStream에서 DRM을 해제할 수 있습니다.

인터페이스의 구현은 아래의 기본적인 Excel Import 로직을 포함하여 자유롭게 확장할 수 있습니다.

- 지정된 디렉터리(filepath) 확인하여 없으면 생성
- 지정된 디렉터리(filepath)에 FileOutputStream 파일 생성
- 전달된 Excel 파일(InputStream in)을 읽어서 FileOutputStream에 저장
- FileOutputStream 닫기

```
public String saveImportStream(VariableList varlist, InputStream in, String filepath) throws
Exception {
    int nIdx = filepath.lastIndexOf("/");
    String sPath = filepath.substring(0, nIdx);

    File file = new File(sPath);
    if (file.exists() == false) {
        file.mkdirs();
    }

    // write input stream to file
    OutputStream out = new FileOutputStream(filepath);

    byte[] buf = new byte[1024];
    int length = 0;
    while ((length = in.read(buf)) > 0) {
        out.write(buf, 0, length);
    }

    out.flush();
    out.close();
    in.close();
}
```

```
return null;
}
```

### 11.2.3 (int) saveExportStream

```
public int saveExportStream(VariableList varlist,
    DataSet dscmd,
    ByteArrayOutputStream out,
    String filepath,
    String fileurl,
    HttpServletResponse response)
```

Excel 스트림으로 변환된 Grid 컴포넌트 데이터를 Excel 파일로 저장하고 해당 파일을 다운로드할 수 있는 URL을 포함한 Export 결과를 넥사크로플랫폼으로 전달하는 인터페이스입니다.

생성되는 Excel 파일에 DRM을 적용할 수 있습니다.

아래의 기본적인 Excel Export 로직을 포함하여 자유롭게 확장할 수 있습니다.

- 파라미터로 전달된 filepath(Excel 파일명)의 디렉터리가 없으면 생성
- 생성된 Excel 파일에 파라미터로 전달받은 Excel로 변환된 Stream(ByteArrayOutputStream out)을 저장
- 넥사크로플랫폼으로 다운로드 URL 및 파일명이 포함된 PlatformData 전달
- 생성된 Excel 파일은 환경설정에 의해서 자동으로 삭제

```
public int saveExportStream(VariableList varlist, DataSet dscmd, ByteArrayOutputStream out,
    String filepath, String fileurl, HttpServletResponse response) throws Exception {
```

```
    int nIdx = filepath.lastIndexOf("/");
    String sPath = filepath.substring(0, nIdx);
```

```
    File file = new File(sPath);
    if (file.exists() == false) {
        file.mkdirs();
    }
```

```
    FileOutputStream fout = new FileOutputStream(filepath);
    fout.write(out.toByteArray());
```

```
    fout.close();
    out.close();
```



```

DataSet dsRes = CommUtil.getDatasetExportResponse(dscmd);

PlatformData resData = new PlatformData();
VariableList varList = resData.getVariableList();

varList.add("ErrorCode", 0);
varList.add("ErrorMsg", "SUCCESS");

dsRes.set(0, "url", fileurl);
resData.addDataSet(dsRes);

HttpPlatformResponse platformRes = new HttpPlatformResponse(response, PlatformType.CONTENT_
TYPE_SSV, "UTF-8");
platformRes.setData(resData);
platformRes.sendData();

return 0;
}

```

## 11.2.4 (Dataset) saveExportStream

```

public Dataset saveExportStream(VariableList varlist,
    DataSet dscmd,
    ByteArrayOutputStream out,
    String filepath,
    String fileurl)

```

구현이 필요하지 않은 메소드입니다. 빈 메소드로 선언 합니다.

## 11.3 사용 설정

- ① 확장 인터페이스를 구현한 후, 클래스 파일을 아래의 경로에 복사합니다.  
확장된 클래스명은 변경할 수 있습니다.

nexacro-xeni > WEB-INF > classes

② 개발된 클래스의 사용을 위해서는 xeni.properties 파일을 아래의 경로중 하나에 생성합니다.

- nexacro-xeni > WEB-INF > classes
- nexacro-xeni > WEB-INF > lib
- CLASSPATH에 정의된 경로

③ xeni.properties 파일에 개발된 클래스명을 nexacro-xeni의 확장 인터페이스로 등록합니다.

제품 버전	확장 인터페이스 등록
넥사크로플랫폼 14	xeni.exportimport.extend={사용자패키지명}.{사용자클래스명} 예) xeni.exportimport.extend=com.nexacro.xeni.util.XeniExtendUseFile
넥사크로플랫폼 17.1, N	xeni.exportimport.storage={사용자패키지명}.{사용자클래스명} 예) xeni.exportimport.storage=com.extend.userExtendClass

## 12.

# nexacro-xeni 확장 인터페이스 작성 예 - DRM

nexacro-xeni 확장 인터페이스로 DRM 솔루션 적용 시에는 DRM 솔루션 제공 업체에서 가이드하는 방법으로 구현해야 합니다. DRM 솔루션에 따라 다양한 메소드 및 방법을 제공하고 있습니다. 이 문서에서는 일반적인 사용 시나리오를 설명합니다.

Excel 파일에 DRM을 적용하는 방법은 아래와 같은 두 가지 방법이 있습니다.

- Client에서 DRM을 적용, 해제
- Server에서 DRM을 적용, 해제

이번 장은 Server에서 DRM을 적용하는 방법만 설명합니다.



이번 장은 넥사크로플랫폼 N 버전을 기준으로 설명합니다.

넥사크로플랫폼 14, 17.1 관련 설명은 [nexacro-xeni 확장 인터페이스](#)를 참고하세요.

## 12.1 DRM이 적용된 Excel 파일 Import/Export 시나리오

DRM이 적용된 Excel 파일의 Import는 아래와 같은 순서로 처리됩니다.

1. 넥사크로플랫폼에서 서버로 파일 업로드
2. 서버에 업로드된 Excel 파일을 DRM 솔루션에서 제공한 메소드를 이용하여 해제
3. PlatformData로 변환해 넥사크로플랫폼으로 전달

Grid 컴포넌트에 표현된 데이터를 Excel 파일로 Export 하는 것은 아래와 같은 순서로 처리됩니다.

1. Grid 컴포넌트에 표현된 데이터를 서버로 업로드
2. 서버에서 Excel 파일 생성
3. DRM 솔루션에서 제공한 메소드를 이용해 DRM이 적용된 Excel 파일로 변환

4. Excel 파일을 내려받을 수 있는 URL을 넥사크로플랫폼으로 전달

## 12.1.1 확장 인터페이스 상속

XeniExcelDataStorageBase 인터페이스를 상속받은 XeniDrmSample 클래스를 생성합니다. 각 참조할 패키지와 DRM 솔루션에서 제공하는 패키지도 import 합니다.

```
package com.nexacro.user.drm;

import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import jakarta.servlet.http.HttpServletResponse;
// import javax.servlet.http.HttpServletResponse;

import com.nexacro.java.xapi.data.DataSet;
import com.nexacro.java.xapi.data.PlatformData;
import com.nexacro.java.xapi.data.VariableList;
import com.nexacro.java.xapi.tx.HttpPlatformResponse;
import com.nexacro.java.xapi.tx.PlatformType;

public class XeniDrmSample implements XeniExcelDataStorageBase {
```

## 12.1.2 메소드 구현

Excel 파일 Import/Export 시나리오에 따라 메소드를 구현합니다.

### (InputStream) loadTargetStream

loadTargetStream 메소드는 다른 구현 없이 지정된 경로의 Excel 파일을 열고 파일 스트림을 전달하도록 구현합니다.

```
public InputStream loadTargetStream(String filepath) throws Exception {
    File file = new File(filepath);
    return new FileInputStream(file);
}
```

## (String) saveImportStream

넥사크로플랫폼에서 전달받은 Excel 데이터를 설정한 위치에 파일로 저장하거나 데이터베이스에 저장하는 기능을 합니다.

saveImportStream 메소드는 파라미터로 전달받은 경로를 확인합니다. 지정된 경로가 존재하지 않을 경우 새로 생성합니다.

```
int nIdx = filepath.lastIndexOf("/");
String sPath = filepath.substring(0, nIdx);
String fileName = filepath.substring(nIdx + 1);
String srcFile = sPath + "/__temp_" + fileName;

File file = new File(sPath);
if(file.exists() == false) {
    file.mkdirs();
}
```

파라미터로 전달받은 DRM이 적용된 Excel 파일 스트림을 지정된 경로에 프리픽스 "\_\_temp\_"를 붙여서 임시 파일을 생성하고 저장합니다.

```
OutputStream out = new FileOutputStream(srcFile);

byte[] buf = new byte[1024];
int length = 0;
while((length = in.read(buf)) > 0) {
    out.write(buf, 0, length);
}

out.flush();
out.close();
in.close();
```

지정된 경로에 저장된 임시 파일(DRM 적용 파일)을 DRM 솔루션이 제공한 DRM 해제 메소드를 이용하여 해제합니다. DRM 메소드의 필수 파라미터와 실행 결과에 대한 Exception 처리는 제공된 DRM 솔루션의 가이드에 따릅니다.

```
boolean isSuccess = DrmUtil.extractDRM(srcFile, filepath);
```

DRM의 해제 메소드가 정상적으로 처리되었으면 DRM이 적용된 임시 파일은 삭제하고 반환합니다.

```
File delFile = new File(srcFile);
if(delFile.exists()) {
    file.delete();
}
```

이 이후의 DRM이 해제된 Excel 파일을 PlatformData로 변환하는 작업은 nexacro-xeni 내부에서 처리하며 따로 구현할 필요는 없습니다. 또한 변경하도록 허용되지 않습니다.

## (int) saveExportStream

이 메소드는 넥사크로플랫폼에서 전달받은 Grid 컴포넌트 데이터를 가지고 nexacro-xeni 내부에서 생성한 Excel 파일 스트림을 Excel 파일로 저장하고 넥사크로플랫폼에서 다운로드 할 수 있는 URL을 전달하는 기능을 구현합니다.

파라미터로 전달받은 경로의 존재 여부를 체크합니다. 경로가 존재하지 않으면 생성합니다.

Grid 데이터를 Excel 파일로 변환하는 기능은 nexacro-xeni 내부에서 처리함으로 여기에서는 구현할 필요가 없습니다. 파라미터 "ByteArrayOutputStream out"로 전달받습니다.

```
int nIdx = filepath.lastIndexOf("/");
String sPath = filepath.substring(0, nIdx);
String fileName = filepath.substring(nIdx + 1);
String srcFile = sPath + "/__temp_" + fileName;

File file = new File(sPath);
if(file.exists() == false) {
    file.mkdirs();
}
```

nexacro-xeni에서 변환된 Excel 파일 스트림(DRM 적용전)을 지정된 경로에 임시 파일명(프리픽스 "\_\_temp\_")으로 생성하고 저장합니다.

```
FileOutputStream fout = new FileOutputStream(srcFile);
fout.write(out.toByteArray());
```

```
fout.close();
out.close();
```

DRM이 적용되지 않은 임시 파일을 DRM 솔루션이 제공한 메소드를 이용하여 DRM이 적용된 파일로 생성합니다. 상세한 구현은 DRM 솔루션에서 제공하는 가이드를 따릅니다.

```
String id = varlist.getString("id");
String name = varlist.getString("name");
String code = varlist.getString("code");
String dept = varlist.getString("dept");
boolean isSuccess = DrmUtil.packagingDRM(srcFile, filepath, id, name, code, dept);
```

DRM 적용이 성공했으면 사용된 임시 파일은 삭제합니다.

```
File delFile = new File(srcFile);
if(delFile.exists()) {
    file.delete();
}
```

DRM이 적용된 Excel 파일의 정보(URL) 및 변환 결과를 PlatformData에 담아 넥사크로플랫폼으로 전달합니다.

```
DataSet dsRes = CommUtil.getDatasetExportResponse(dscmd);

PlatformData resData = new PlatformData();
VariableList varList = resData.getVariableList();

varList.add("ErrorCode", 0);
varList.add("ErrorMsg", "SUCCESS");

dsRes.set(0, "url", fileurl);
resData.addDataSet(dsRes);

HttpPlatformResponse platformRes = new HttpPlatformResponse(response, PlatformType.CONTENT_TYPE_SSV, "UTF-8");
platformRes.setData(resData);
platformRes.sendData();

return 0;
```



넥사크로플랫폼으로 전달해야 할 정보가 있으면 이 위치에 추가할 수 있습니다.

## (DataSet) saveExportStream

구현하지 않아도 되지만 빈 메소드로 선언해야 하는 메소드입니다.

```
public DataSet saveExportStream(VariableList varlist, DataSet dscmd, ByteArrayOutputStream out,
String filepath, String fileurl) throws Exception {
    return null;
}
```



# 13.

## web.xml

nexacro-xeni/WEB-INF/web.xml 의 설정 항목 입니다.

### 13.1 Export 관련 설정

#### 13.1.1 export file path

현재 context(nexacro-xeni) 아래 Export 된 파일이 저장 될 경로를 지정 합니다. export 시 지정한 폴더가 존재 하지 않는 경우 자동 생성 합니다. 임시로 저장된 export 파일은 생성된 지 일정 시간이 지나면 자동 삭제 됩니다. 기본 설정은 '/export' 입니다.

```
<context-param>
  <param-name>export-path</param-name>
  <param-value>/export</param-value>
</context-param>
```

#### 13.1.2 관리 실행 주기

export 시에 사용된 임시 파일 관리와 chunked data 관리를 위한 monitor 실행 주기를 설정 합니다. 단위는 '분'이며 기본 설정은 '30'분 입니다.

```
<context-param>
  <param-name>monitor-cycle-time</param-name>
  <param-value>30</param-value>
</context-param>
```



monitor 실행 주기 설정 시 초 단위 설정을 지원합니다.  
실행 주기를 30초로 설정할 때는 아래와 같은 형식으로 값을 설정합니다.  
<param-value>30/sec</param-value>

파일 관리가 설정된 경우 export 시에 사용된 모든 파일을 검사하여 일정 시간이 경과한 파일은 삭제 합니다. export 시에 nexacro platform 으로 부터 전송 받은 chunked data 가 일정 시간 이상 메모리에 남아 있을 경우 오류에 의한 작업 중지로 판단하고 삭제 합니다.



chunked data 관리는 [파일 관리 실행](#)에 관계 없이 주기마다 이루어 집니다.

### 13.1.3 파일 저장 시간

export 시에 사용된 임시 파일과 chunked data 의 저장 시간을 설정 합니다. 단위는 ‘분’이며 기본 설정은 ‘10’분 입니다.

```
<context-param>
  <param-name>file-storage-time</param-name>
  <param-value>10</param-value>
</context-param>
```



임시 파일 저장 시간 설정 시 초 단위 설정을 지원합니다.  
저장 시간을 30초로 설정할 때는 아래와 같은 형식으로 값을 설정합니다.  
<param-value>30/sec</param-value>

## 13.2 Import 관련 설정

### 13.2.1 import file path

import 시에 임시로 저장 될 파일의 경로를 지정 합니다. import 시 지정한 폴더가 존재 하지 않는 경우 자동 생성 합니다. 임시로 사용된 파일은 import 완료 직후 자동 삭제 됩니다. 기본 설정은 ‘/import’ 입니다.

```
<context-param>
  <param-name>import-path</param-name>
  <param-value>/import</param-value>
</context-param>
```

## 13.2.2 파일명 지정

import 시 임시로 저장될 파일명을 'import\_temp'로 고정합니다.

```
<context-param>
  <param-name>import-temp-name</param-name>
  <param-value>true</param-value>
</context-param>
```

## 13.3 기타 설정

### 13.3.1 파일 관리 실행

임시 파일 관리 여부를 설정 합니다. 이 값이 'false'로 지정될 경우 export/import 시에 사용된 임시 파일을 삭제 하지 않습니다. 기본 설정은 'true' 입니다.

```
<context-param>
  <param-name>monitor-enabled</param-name>
  <param-value>true</param-value>
</context-param>
```

### 13.3.2 텍스트 한정자

텍스트 파일 처리 시 텍스트 한정자 사용 여부를 설정합니다. 텍스트 한정자는 " (quotation mark)를 사용합니다.

```
<context-param>  
  <param-name>csv-quote</param-name>  
  <param-value>true</param-value>  
</context-param>
```

# 14.

## xeni.properties

옵션을 설정할 수 있는 파일로 필수 파일은 아닙니다. 아래 위치 중 한 곳에 xeni.properties 파일이 있을 경우 옵션이 적용됩니다.

- nexacro-xeni/WEB-INF/classes
- nexacro-xeni/WEB-INF/lib
- CLASSPATH

현재 사용 가능한 옵션들에 대해 설명합니다.

### 14.1 xeni.exportimport.storage

Export/Import 데이터 처리에 대해 사용자가 확장한 class를 지정 할 수 있는 옵션입니다. Export/Import 시에 사용되는 데이터는 임시 파일로 저장되어 처리되는 것이 기본 동작이지만 file 외의 저장소(ex. Database) 등을 사용하고 자 할 때 XeniExcelDataStorageBase interface를 구현하고 해당 class명을 이 옵션에 지정합니다.

```
public interface XeniExcelDataStorageBase {  
    InputStream loadTargetStream(String filepath) throws Exception;  
  
    String saveImportStream(VariableList varlist, InputStream in, String filepath) throws  
Exception;  
  
    int saveExportStream(VariableList varlist,  
        DataSet dscmd,  
        ByteArrayOutputStream out,  
        String filepath,  
        String fileurl,  
        HttpServletResponse response) throws Exception;  
}
```



xeni.exportimport.storage는 넥사크로플랫폼 N 버전을 기준으로 설명한 내용입니다.  
넥사프로플랫폼 14, 17.1 관련 설명과 확장 인터페이스 관련 설명은 [nexacro-xeni 확장 인터페이스](#)를  
참고하세요.

## 14.2 xeni.multipart.proc

Spring과 같은 framework 사용 시 multipart request를 처리하기 위해 사용자가 확장한 class를 지정할 수 있는 옵션입니다. Spring framework는 nexacro uiadapter를 통해 기능을 지원합니다.

```
public interface XeniMultipartProcBase {  
    XeniMultipartReqData getImportData(HttpServletRequest req) throws Exception;  
}
```

## 파트 IV.

---

### App Builder

# 15.

## App Builder 설치

---

App Builder 관련 문서

- 서버 설정/개발 가이드 > App Builder
  - [App Builder 설치](#)
  - [서버 환경 설정](#)
  - [디플로이 서버 설정](#)
  - [운영체제별 서명 설정](#)
  - [사용자 라이브러리 설정](#)
  - [넥사크로 라이브러리 설정](#)
  - [사용자 설정](#)
- 앱 배포 가이드 > App Builder
  - [App 생성](#)
  - [App 목록](#)
- 개발도구 가이드 > Build App 단계에서 설치 파일 만들기
  - [Build App 단계에서 설치 파일 만들기](#)

### 15.1 App Builder 설치 전 필요한 작업

App Builder를 서버에 설치하기 전에 필요한 작업입니다. App Builder를 설치하기 위한 공통 작업과 배포 대상에 따라 추가 작업이 필요합니다.

#### 15.1.1 App Builder 설치 공통

JDK 설치 후 Tomcat(WAS)을 설치합니다.





본문에서는 JDK 17, Tomcat 9 버전을 사용했습니다.

## 15.1.2 Android

Android SDK를 설치합니다. 안드로이드 스튜디오 설치 후 SDK Manager를 실행하거나 명령행 도구를 내려받아 SDK를 설치할 수 있습니다.



본문에서는 Android 14(API 34) 버전을 사용했습니다.

## 15.1.3 iOS, macOS

Xcode를 설치하고 필요한 SDK를 설치합니다. Xcode는 macOS 이외의 운영체제에는 설치할 수 없으므로 macOS 운영체제를 사용할 수 있는 장비가 필요합니다.



본문에서는 Xcode(Build 15C500b), iOS SDK(17.2), OS X SDK(14.2) 버전을 사용했습니다.

## 15.2 App Builder 설치

App Builder는 WAR(Web application ARchive) 파일로 제공됩니다. 제공된 WAR 파일(appbuilder.war)을 각 WAS 환경에 따라 배포합니다.



본문에서는 Tomcat 9 버전을 사용했습니다. 해당 버전 기준으로 설명합니다.

### ① WAR 파일 배포

[webapps] 폴더에 appbuilder.war 파일을 복사해놓고 Tomcat 서버를 시작합니다. webapps 경로에 appbuilder 폴더가 생성된 것을 확인하고 Tomcat 서버를 중지합니다.

### ② 관리 콘솔 설정

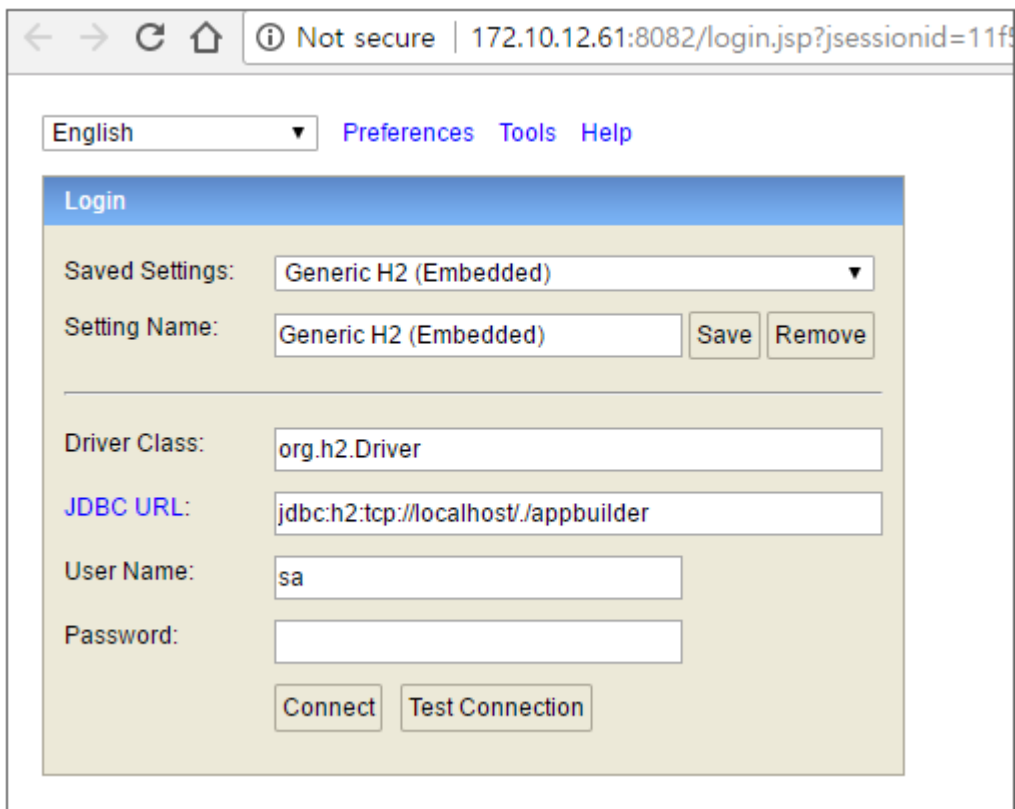
[webapps\appbuilder\ Nexacro\_ui] 폴더에서 environment.xml.js 파일을 텍스트 편집기로 열고 "HOST:

PORT"로 표기된 부분을 사용할 URL로 수정합니다.

```
nexacro._addService("svcUrl", "JSP", "http://HOST:PORT/appbuilder", "none", null, "", "0",
"0");
nexacro._addService("svcUrl", "JSP", "http://127.0.0.1:8080/appbuilder", "none", null, "",
"0", "0");
```

### 3 H2 Database 시작

[webapps/appbuilder/WEB-INF/package/h2/bin] 폴더에서 runH2DB 배치 파일을 실행하면 H2 Database가 시작되고 관리 콘솔이 표시됩니다. 관리 콘솔창은 사용하지 않습니다. 열린 웹브라우저 창을 닫습니다.



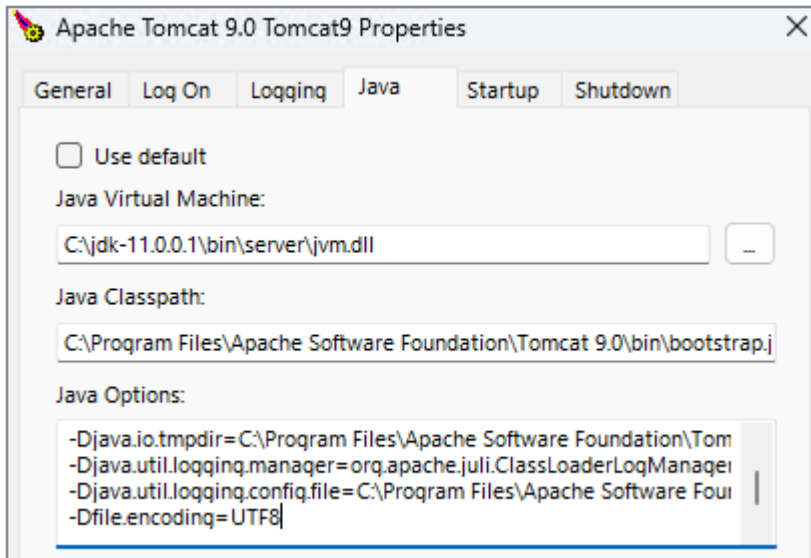
H2 Database를 처음 시작했을 때 해당 폴더에 필요한 파일을 생성하는데 윈도우 운영체제를 사용하는 경우에는 Tomcat 설치 경로에 따라 관리자 권한이 없으면 오류가 발생하는 경우가 있습니다. 오류가 발생하는 경우에는 관리자 권한으로 배치 파일을 실행해야 합니다.

### 4 인코딩 설정

일부 환경에서 한국어가 정상적으로 표기되지 않을 수 있습니다. 그런 경우에는 인코딩 설정을 추가합니다. [bin] 폴더 아래 catalina.bat 파일을 열어 아래와 같이 수정합니다.

```
set "JAVA_OPTS=%JAVA_OPTS% %LOGGING_CONFIG% -Dfile.encoding=UTF8"
```

Tomcat 버전에 따라 catalina.bat 파일을 사용하지 않는 경우가 있습니다. 그런 경우에는 [bin] 폴더 아래에서 실행파일인 Tomcat7w.exe를 실행하면 Tomcat 속성을 설정할 수 있는 창이 나타납니다. 해당 창에서 [Java > Java Options] 항목에 "-Dfile.encoding=UTF8" 항목을 추가합니다.



## 5 서버 시작

[webapps] 폴더에 복사해놓은 appbuilder.war 파일을 삭제하고 Tomcat 서버를 다시 시작합니다.

## 6 관리 콘솔 접근

웹 브라우저를 열어 아래 URL을 입력 후 관리 콘솔에 접근할 수 있는지 확인합니다. 입력할 URL은 pbuilder\_ui.xadl.js 파일에서 수정한 URL입니다.

```
http://xxx.xxx.xxx.xxx:8080/appbuilder
```

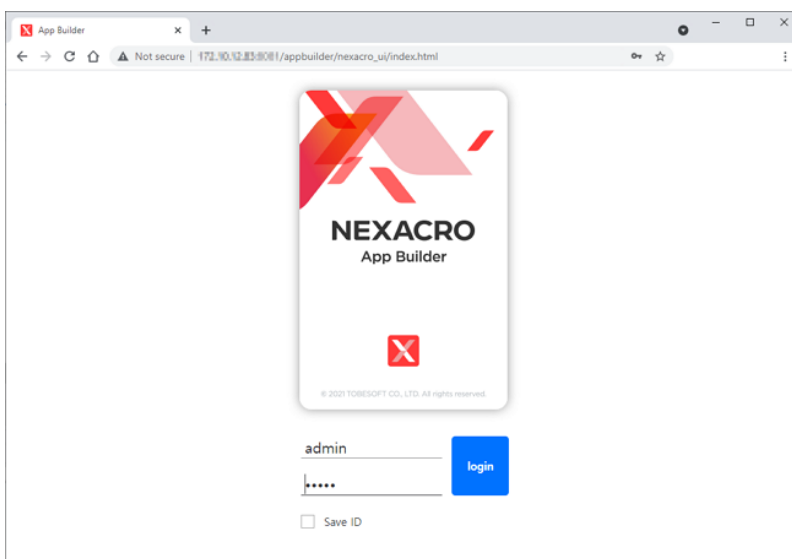
# 16.

## 서버 환경 설정

App Builder 관련 문서

- 서버 설정/개발 가이드 > App Builder
  - [App Builder 설치](#)
  - [서버 환경 설정](#)
  - [디플로이 서버 설정](#)
  - [운영체제별 서명 설정](#)
  - [사용자 라이브러리 설정](#)
  - [넥사크로 라이브러리 설정](#)
  - [사용자 설정](#)
- 앱 배포 가이드 > App Builder
  - [App 생성](#)
  - [App 목록](#)
- 개발도구 가이드 > Build App 단계에서 설치 파일 만들기
  - [Build App 단계에서 설치 파일 만들기](#)

App Builder 설치 이후 관리 콘솔에 접속해 App Builder 사용을 위한 설정을 진행합니다. 관리 콘솔에 접속하면 로그인 창이 표시됩니다. 최초 로그인 시 ID는 "admin", Password는 "admin"입니다. Password는 로그인 후 메뉴[User]에서 변경할 수 있습니다.



App Builder 환경 설정을 위해 관리 콘솔에서 메뉴[Configuration > Server Configuration] 항목을 선택합니다. 해당 메뉴는 4가지로 구분되어 있습니다. General 항목을 제외한 나머지 항목은 체크박스에서 사용 여부를 선택할 수 있습니다.

Server Configuration

User Management

1

General

Save

JDK Path

/Library/Java/JavaVirtualMachines/zulu-11.jdk/Contents/Home/

Work Path

/Users/tobesoft/appbuilder/output/010

Language

English

2

☒ iOS & macOS Configuration

Login Keychain Password

\*\*\*\*\*

Developer Directory

/Applications/Xcode.app/Contents/Developer

Xcode Version

Xcode 15.2

Xcode Build Version

15C500b

Name	FullName	Version	PlatformVersion
driverkit23.2	DriverKit23.2.sdk - DriverKit 23.2	23.2	23.2
iphones17.2	iPhoneOS17.2.sdk - iOS 17.2	17.2	17.2
iphonesimulator17.2	iPhoneSimulator17.2.sdk - Simulator - iC	17.2	17.2

3

☒ In-house Distribution

Host Name (IP)

macos.tobesoft.co.kr

Port

22

Login ID

tobesoft

Password

\*\*\*\*\*

Target URL

https://casao.tobesoft.co.kr/stat/ty--tobesoft/vcs/apps

Target Directory

/Users/tobesoft/public\_html/v24/apps

4

☒ Android Configuration

SDK Path

/Users/tobesoft/appbuilder/works/guest/appbuilder/android-sdk-macosx

ID	Name	Type	API Level	Revisions	Tag/ABIs
android-33	Android 13	Platform	33	2	no ABIs.

	항목	설명
1	General	JDK 경로와 생성된 앱 프로젝트를 관리할 경로를 지정합니다.
2	iOS & macOS Configuration	iOS, macOS SDK 정보를 표시합니다.
3	In-house Distribution	In-house 배포 설정 시 관련 정보를 지정합니다.
4	Android Configuration	Android SDK 설치 경로를 지정합니다.

## 16.1 General

General 항목은 기본적으로 작성해야 하는 항목입니다.

General	
1 JDK Path	/Library/Java/JavaVirtualMachines/zulu-11.jdk/Contents/Home/
2 Work Path	
3 Language	English

	항목	설명
1	JDK Path	JDK 설치 경로를 지정합니다.
2	Work Path	앱 프로젝트를 생성하고 업로드한 리소스 파일을 저장할 작업 경로를 지정합니다.
3	Language	UI 표시 언어를 설정합니다. 한국어, 영어, 중국어를 지원합니다.

## 16.2 iOS & macOS Configuration

iOS, macOS 앱을 빌드하기 위한 설정입니다.

Xcode와 관련 SDK가 정상적으로 설치된 경우라면 관련 설정을 자동으로 확인하고 설정값을 표시합니다. 설정값이 표시되지 않는 경우에는 Xcode나 SDK가 정상적으로 설치되었는지 확인합니다.

<input checked="" type="checkbox"/> iOS & macOS Configuration					
1	Login Keychain Password	.....			
2	Developer Directory	/Applications/Xcode.app/Contents/Developer			
3	Xcode Version	Xcode 15.2	4	Xcode Build Version	15C500b
5	Name	FullName	Version	PlatformVersion	
	driverkit23.2	DriverKit23.2.sdk - DriverKit 23.2	23.2	23.2	
	iphoneos17.2	iPhoneOS17.2.sdk - iOS 17.2	17.2	17.2	
	iphonesimulator17.2	iPhoneSimulator17.2.sdk - Simulator - it	17.2	17.2	

	항목	설명
1	Login Keychain Password	로그인 키체인 비밀번호를 입력합니다.
2	Developer Directory	Xcode developer directory를 표시합니다.
3	Xcode Version	설치된 Xcode 버전을 표시합니다.
4	Xcode Build Version	Xcode 빌드 버전을 표시합니다.
5	SDK Details	설치된 관련 SDK 상세 정보를 표시합니다.

## 16.3 In-house Distribution

In-house Distribution이란 Apple App Store를 사용하지 않고 자체적으로 설치파일을 배포할 수 있는 서버를 사용하는 것입니다. 해당 기능을 사용하기 위해서는 HTTPS 설정이 되어 있어야 하며 공인된 도메인으로 발급받은 서버 인증서가 갖춰져야 합니다.

In-house Distribution 항목이 설정된 경우에는 앱 빌드 시 .ipa 파일과 .plist 파일을 설정된 서버로 전송합니다. 빌드된 앱을 내려받을 때 설정한 Host 정보를 적용합니다.

☒ **In-house Distribution**

1

Host Name (IP)

cacao.nexacro.com

2

Port

22

3

Login ID

nexacro

4

Password

.....

5

Target URL

https://cacao.nexacro.com/~nexacro/appbuilder/test/

6

Target Directory

/home/nexacro/public\_html/appbuilder/test

	항목	설명
1	Host Name(IP)	서버 호스트명 또는 IP 정보를 입력합니다.
2	Port	파일 전송을 위한 Port 정보를 입력합니다.
3	Login ID	서버 접속 시 로그인 ID를 입력합니다.
4	Password	서버 접속 시 로그인 Password를 입력합니다.
5	Target URL	복사된 .ipa , .plist 파일을 참조하게 될 URL을 입력합니다.
6	Target Directory	복사된 .ipa , .plist 파일이 위치하게 경로를 입력합니다.



Target URL 항목에 입력된 URL과 실제 파일을 내려받기 위해 접속하는 URL은 다릅니다. Target URL 항목은 .ipa, .plist 파일이 실제 저장된 경로이며 파일을 내려받을때는 빌드 후 제공되는 URL에 접속해 설치 파일을 내려받습니다.



HTTPS 설정이 되어 있지 않거나 공인된 도메인으로 발급받은 서버 인증서가 없는 경우에는 "In-house Distribution" 기능을 통해 앱 설치를 할 수 없습니다. 이런 경우에는 "In-house Distribution" 기능을 사용하지 않고 .ipa 파일을 PC에 내려받은 후 Xcode 또는 애플 아이튠즈를 통해 앱을 설치해야 합니다.

## 16.4 Android Configuration

<input checked="" type="checkbox"/> <b>Android Configuration</b>					
<div> <div>1</div> <div>SDK Path</div> <div></div> </div>					
ID	Name	Type	API Level	Revisions	Tag/ABIs
android-33	Android 13	Platform	33	2	no ABIs.

	항목	설명
1	SDK Path	Android SDK가 설치된 경로를 지정합니다.



SDK Path 항목 설정 후 설치된 Android SDK 목록이 표시되지 않는 경우에는 아래 사항을 확인합니다.

- Android SDK가 정상적으로 설치되었는지 확인
- 설치된 Android SDK Path를 정확하게 지정했는지 확인
- Android SDK Manager 를 실행해 Android 13(API 33)이 설치되었는지 확인



# 17.

## 디플로이 서버 설정

App Builder 관련 문서

- 서버 설정/개발 가이드 > App Builder
  - [App Builder 설치](#)
  - [서버 환경 설정](#)
  - [디플로이 서버 설정](#)
  - [운영체제별 서명 설정](#)
  - [사용자 라이브러리 설정](#)
  - [넥사크로 라이브러리 설정](#)
  - [사용자 설정](#)
- 앱 배포 가이드 > App Builder
  - [App 생성](#)
  - [App 목록](#)
- 개발도구 가이드 > Build App 단계에서 설치 파일 만들기
  - [Build App 단계에서 설치 파일 만들기](#)

프로젝트 Update Type이 "Local"이 아닌 경우에는 필요한 리소스를 사용자가 내려받을 수 있도록 합니다. 앱 빌더를 사용하는 경우 앱 빌더 서비스 자체가 리소스를 제공하는 서버로 운영되는데, 별도 서버를 운영하고자 하는 경우 리소스를 업로드 하는 기능을 제공합니다.

Deploy Server 설정을 위해서는 관리 콘솔에서 메뉴[Settings > Deploy] 항목을 선택합니다.



리소스 업로드는 Secure File Transfer Protocol(SFTP) 통신으로 처리합니다. Deploy Server는 SFTP로 접속할 수 있는 서버만 지원합니다. 아래 Deploy Server 정보도 SFTP 기준으로 설정해야 합니다.

## 17.1 Deploy Server 설정하기

Nexacro Library
Signing
User Library
Deploy

5

Name

3

New

4

Save

	Name	Host	Port	Directory	Description	LastUploadTime
1	<input type="checkbox"/> Test Deploy Server1	casco.tobiasoft.co.kr	22	/home/tobiasoft/public_html	test deploy server	2020-05-06 16:11:42
	<input type="checkbox"/> Test Deploy Server2	192.168.1.100	22	/home/tobiasoft/public_html	test deploy	2020-04-28 14:04:32
	<input type="checkbox"/> Deploy Server3	casco.tobiasoft.co.kr	22	/home/tobiasoft/public_html	test	2020-04-28 15:22:53

2

Server Name

Test Deploy Server1

Use In-house Server Information

☐

Host Name

casco.tobiasoft.co.kr

Port

22

Login ID

tobiasoft

Password

\*\*\*\*\*

Plist URL

<https://casco.tobiasoft.co.kr/~tobiasoft/appbuilder/test/temp>

Directory


/home/tobiasoft/public\_html/appbuilder/test/temp

Private Key File

Description

test deploy server

	항목	설명
1	Deploy Server 목록	등록한 Deploy Server 목록을 표시합니다.
2	Deploy Server 상세	Deploy Server에 접속하고 리소스를 업로드 하기 위한 정보를 관리합니다.
3	추가	새로운 Deploy Server를 추가합니다.
4	저장	Deploy Server를 추가하거나 상세 항목 변경 후 저장합니다.
5	삭제	목록에서 선택한 Deploy Server 정보를 삭제합니다.

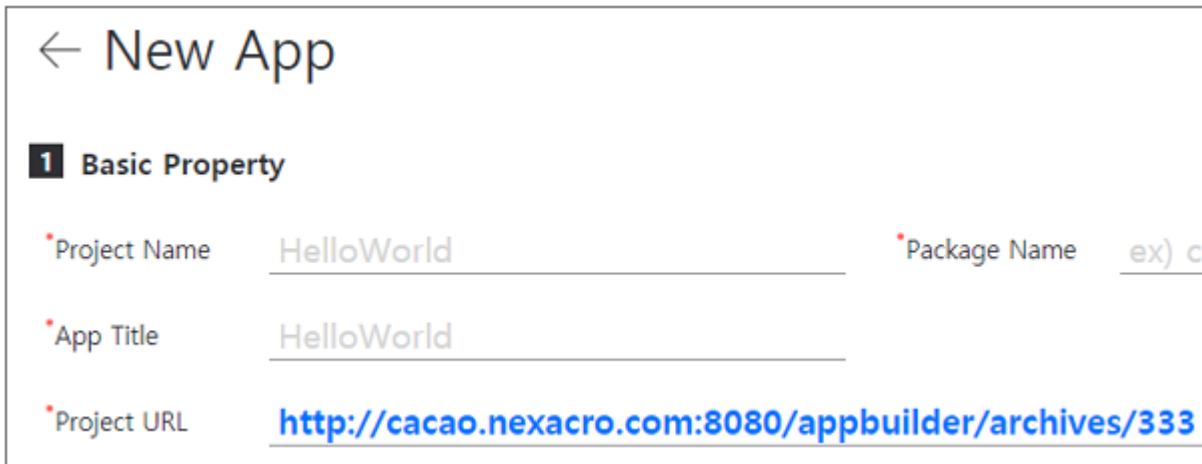
*Server Name	2	Test Deploy Server1	1	<input type="checkbox"/> Use In-house Server Information
*Host Name	3	cacao.nexacro.com	*Port	4 22
*Login ID	5	nexacro	*Password	6 .....
*Plist URL	7	<a href="https://cacao.nexacro.com/~nexacro/appbuilder/test/temp">https://cacao.nexacro.com/~nexacro/appbuilder/test/temp</a>		
*Directory	8	/home/nexacro/public_html/appbuilder/test/temp		
Private Key File	9			
Description	10	test deploy server		

	항목	설명
1	Use In-house Server Information	체크 시 In-house Distribution 설정에 입력한 정보를 가져옵니다.
2	Server Name	Deploy Server 이름
3	Host Name	접속할 SFTP HOST 정보 예) cacao.dummy.com
4	Port	접속할 Port 정보 예) 22
5	Login ID	로그인 ID 정보
6	Password	로그인 비밀번호 정보
7	Plist URL	리소스를 내려받는 URL 정보 메뉴 [BuildApp > App Info] 항목에서 Deploy 실행 시 nexacro Project URL 정보로 등록한 URL과 같은 도메인인지 확인하기 위한 용도입니다. 예) https://cacao.dummy.com/appbuilder/test
8	Directory	리소스 파일을 업로드할 서버 내 경로 정보 예) /home/dummy/public_html/appbuilder/test/temp
9	Private Key File	디플로이 서버에서 공개키 인증을 필요로 하는 경우 파일을 업로드합니다.
10	Description	참고 사항 기재

## 17.2 nexacro Project URL 변경하기

앱빌더를 사용하는 경우 기본 프로젝트 URL은 아래와 같은 형식으로 지정됩니다.

| http(s)://[server domain]/appbuilder/archives/\*\*\*



← New App

**1 Basic Property**

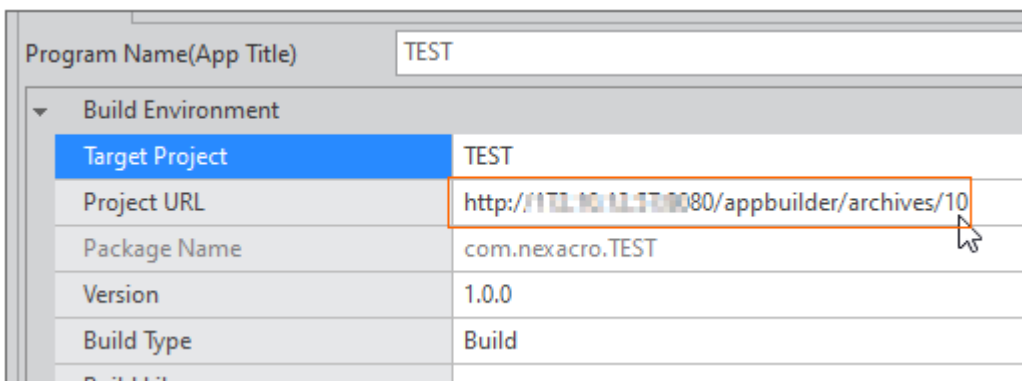
\*Project Name HelloWorld \*Package Name ex) c

\*App Title HelloWorld

\*Project URL <http://cacao.nexacro.com:8080/appbuilder/archives/333>

별도의 Deploy Server를 설정하고자 한다면 Nexacro Project URL 정보를 실제 운영할 서버 URL로 변경하고 앱 빌드 작업을 진행해야 합니다. 사용자가 앱 실행 시 해당 URL에서 리소스 파일을 확인합니다.

넥사크로 스튜디오에서는 앱을 빌드하는 경우에는 Build App 단계에서 Project URL 항목을 변경합니다.



Program Name(App Title) TEST

▼ Build Environment

Target Project	TEST
Project URL	<u>http://11.10.11.17:8080/appbuilder/archives/10</u>
Package Name	com.nexacro.TEST
Version	1.0.0
Build Type	Build
Build Library	

## 17.3 Deploy Server에 리소스 업로드하기

Deploy Server에 리소스를 업로드하는 작업은 앱 빌드가 성공한 이후 진행할 수 있습니다. 앱 빌드가 성공했다면 메뉴[BuildApp > App List] 목록에서 [Deploy] 버튼을 클릭하거나 [Basic Property] 항목에서 Options 영역을 확장하고 [Deploy] 버튼을 클릭해서 업로드할 수 있습니다.

1 Basic Property

\*Project Name

update\_copy

\*Package Name

\*App Title

defalutAppbuilder200428

\*Project URL

<http://111.111.111.111/appbuilder/archives/331>

\*Version

1.0.0

\*Owner

admin

\*Build Mode

☒ debug
 ☐ release

\*Access

☐ public
 ☐ share
 ☒ private
 

+

⌵

\*Detail Error Message Output

☐ false
 ☒ true

Deploy

Auto Deploy

☐

Deploy

History

\*Deploy Server

Test Deploy Server1 (cascadeAppBuilder)

⌵

+

\*Directory

/home/111.111.111.111/public\_html/appbuilder/test/temp

Options

⌵

ID

Authority

Deploy Server는 [Basic Property]에서 설정한 서버로 동작합니다. 서버를 설정하지 않으면 리소스를 업로드하지 않습니다. 업로드 처리 여부에 따라 결과 메시지를 표시합니다.

# 18.

## 운영체제별 서명 설정

App Builder 관련 문서

- 서버 설정/개발 가이드 > App Builder

[App Builder 설치](#)

[서버 환경 설정](#)

[디플로이 서버 설정](#)

[운영체제별 서명 설정](#)

[사용자 라이브러리 설정](#)

[넥사크로 라이브러리 설정](#)

[사용자 설정](#)

- 앱 배포 가이드 > App Builder

[App 생성](#)

[App 목록](#)

- 개발도구 가이드 > Build App 단계에서 설치 파일 만들기

[Build App 단계에서 설치 파일 만들기](#)

앱 빌드 시 사용할 Signing 설정을 위해 관리 콘솔에서 메뉴[Settings > Signing] 항목을 선택합니다. Signing 파일을 등록하거나 생성(Android의 경우)하고 등록된 내용을 확인할 수 있습니다.

Nexacro Library

Signing

User Library

Deploy

1

Name

Q

New

Save

2

	OS Type	Name	Owner	Access	Last Update Date
<input type="checkbox"/>	Android	android-signing	admin	public	2020-05-27 17:51:41
<input type="checkbox"/>	Android	test200424021	admin	private	2020-05-27 17:50:41
<input type="checkbox"/>	Android	android_signing_popup	admin	private	2020-04-28 15:28:01
<input type="checkbox"/>	Android	android-signing-0511	admin	public	2020-05-11 10:12:08
<input type="checkbox"/>	Android	addTest0511_2	admin	private	2020-05-11 18:18:40

3

OS Type

☒ Android

☐ IOS

☐ macOS

Name

android-signing

Owner

admin

Access

☒ public

☐ share

☐ private

+

Keystore Alias Name / Password

/

\*\*\*\*\*

Keystore File

☒ Generate

☐ Upload

android-signing.keystore

Keystore Password

\*\*\*\*\*

	ID	Authority	Role
--	----	-----------	------

	항목	설명
1	Name 검색	등록된 항목의 Name 정보를 검색합니다.
2	Signing 목록	등록된 항목의 목록을 표시합니다.
3	Details	추가할 항목의 세부 정보를 입력하거나 등록된 항목의 세부 정보를 표시합니다. Name, Owner, Access 항목은 배포할 대상과 상관없이 공통으로 적용되는 항목입니다.

Name

android-signing

Owner

admin

Access

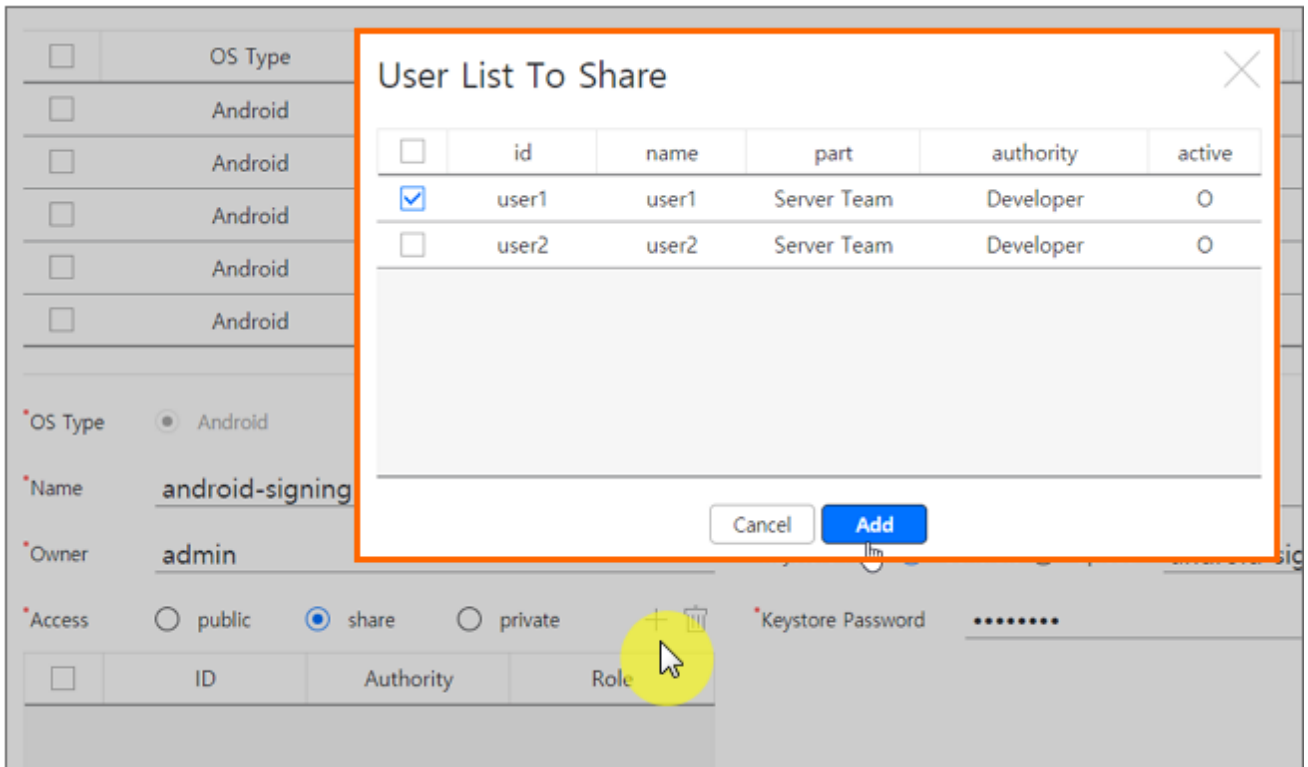
☒ public
☐ share
☐ private

+

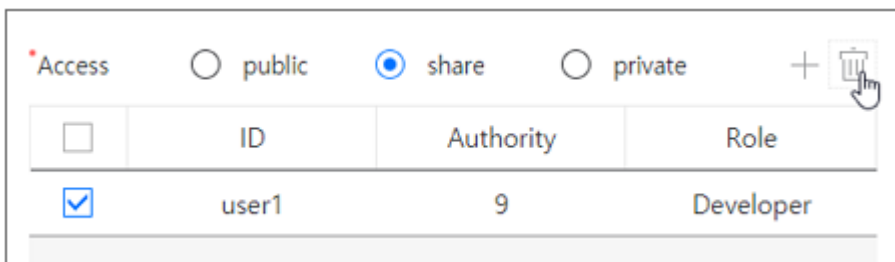
<input type="checkbox"/>	ID	Authority	Role

	항목	설명
1	Name	등록할 항목 이름을 입력합니다.
2	Owner	Keystore(또는 Signing)의 소유자를 입력합니다. (로그인 계정으로 자동 입력됩니다).
3	Access	Keystore(또는 Signing)의 접근 권한을 설정합니다. 권한 설정에 따라 사용자가 App 생성 시 사용할 수 있는 Signing 목록이 다르게 표시됩니다. - share: 지정된 사용자 공유 - public: 전체 사용자 공유 - private: Owner만 사용

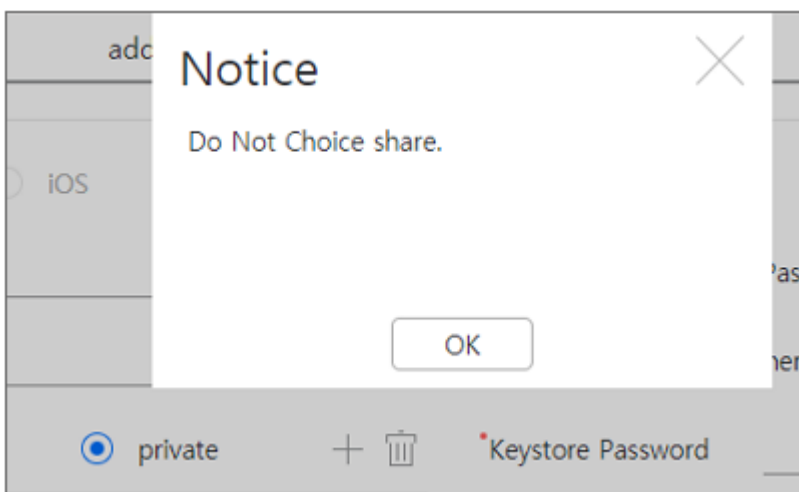
Keystore 접근 권한 설정 시 지정된 사용자 공유(share)를 선택한 경우에는 등록된 사용자 중 권한을 부여할 대상을 선택할 수 있습니다.



keystore 접근 권한이 부여된 사용자를 확인할 수 있으며 해당 사용자를 삭제하면 접근 권한이 해제됩니다.



public, private 항목을 선택한 경우에는 사용자를 선택 기능이 동작하지 않습니다.





## 18.1 Android

Keystore Alias Name / Password: **1** test\_alias **2** .....

Keystore File: **3** ☒ Generate ☐ Upload android-signing.keystore

Keystore Password: **4** ..... **Click & Download**

	항목	설명
1	Keystore Alias Name	Keystore Alias 이름을 입력합니다. Keystore Alias 이름은 6자 이상으로 입력해야 합니다.
2	Keystore Alias Password	Keystore Alias의 비밀번호를 입력합니다. Keystore Alias 비밀번호는 6자 이상으로 입력해야 합니다.
3	Keystore File	Keystore 파일을 생성하거나 가지고 있는 파일을 등록합니다. - Generate: App Builder에서 Keystore 파일을 생성합니다. - Upload: Keystore 파일을 등록하고 등록된 파일을 표시합니다. 등록된 파일은 링크 형태로 표시되며 링크를 클릭하면 해당 파일을 내려받을 수 있습니다.
4	Keystore Password	Keystore 비밀번호를 입력합니다.



Keystore에 대한 자세한 사항은 아래 URL을 참고하세요.

<https://developer.android.com/studio/publish/app-signing>

## 18.2 iOS

Certificate Name: **1** test

Certificate Password: **2** .....

Certificate File: **3** findmask\_0622.p12

Provisioning Profile: **4** testappstore.mobileprovision

Provisioning Type: **5** app-store

	항목	설명
1	Certificate Name	iOS Certificate 이름을 입력합니다.
2	Certificate Password	등록한 iOS Certificate 파일 비밀번호를 입력합니다.
3	Certificate File	iOS Certificate 파일을 등록합니다.
4	Provisioning Profile	Provisioning Profile 파일을 등록합니다.
5	Provisioning Type	Provisioning Profile 파일 유형을 표시합니다. 입력 항목은 아니고 Provisioning Profile 파일 속성에 따라 enterprise, app-store, ad-hoc, development 중 한 가지가 표시됩니다.



iOS Certificate 파일 발급 시 등록한 Package 이름과 앱 생성 시 입력한 Package 이름이 같아야 정상적으로 앱을 빌드할 수 있습니다.



wildcard app id(ex: com.nexacro.\*)로 in-house용 Provisioning profile을 생성할 수 없습니다. 개발용 인증서만 wildcard app id Provisioning profile을 생성할 수 있으며 배포용 Provisioning profile을 생성할 경우에는 반드시 Explicit app id(ex:com.nexacro.helloApp)을 선택하여 생성합니다.



Certificate 파일과 Provisioning Profile 파일에 대한 자세한 사항은 아래 URL을 참고하세요.  
<https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/MaintainingCertificates/MaintainingCertificates.html>

## 18.3 macOS

*Certificate Name	① Developer ID Application: TOBESOFT (RAH8A8
*Certificate Password	② .....
*Certificate File	③ developer-id-tobesoft.p12 

	항목	설명
1	Certificate Name	macOS Certificate 이름을 입력합니다.
2	Certificate Password	등록한 iOS Certificate 파일 비밀번호를 입력합니다.
3	Certificate File	macOS Certificate 파일을 등록합니다.



macOS Certificate 파일 발급 시 등록한 Package 이름과 앱 생성 시 입력한 Package 이름이 같아야 정상적으로 앱을 빌드할 수 있습니다.

# 19.

## 사용자 라이브러리 설정

App Builder 관련 문서

- 서버 설정/개발 가이드 > App Builder

[App Builder 설치](#)

[서버 환경 설정](#)

[디플로이 서버 설정](#)

[운영체제별 서명 설정](#)

[사용자 라이브러리 설정](#)

[넥사크로 라이브러리 설정](#)

[사용자 설정](#)

- 앱 배포 가이드 > App Builder

[App 생성](#)

[App 목록](#)

- 개발도구 가이드 > Build App 단계에서 설치 파일 만들기

[Build App 단계에서 설치 파일 만들기](#)

넥사크로는 모바일 운영체제에서 제공하는 기능을 넥사크로 스크립트 환경에서 사용할 수 있는 기능을 개발할 수 있습니다. 이렇게 개발된 기능을 DeviceAPI라고 하며 라이브러리 형태로 생성해 앱 빌드 시 적용할 수 있습니다.

User Library 설정을 위해 관리 콘솔에서 메뉴[Settings > User Library] 항목을 선택합니다. User Library 파일을 등록하고 등록된 내용을 확인할 수 있습니다.

Nexacro Library

Signing

User Library

Deploy

Name

New

Save

<input type="checkbox"/>	Name	Description	Version	Device Type	Android Class	Last Upload Time	Owner
<input type="checkbox"/>	pluginTest	pluginTest	1.0	<div>IOSlibHelloPlugin.a</div> <div>Androidlibhelloworldplugin.jar</div>	com.example.helloj	2020-04-13 16:36:00	admin
<input type="checkbox"/>	androidPlugin	com.example.helloj	1.2	<div>IOS</div> <div>Androidlibhelloworldplugin.jar</div>	com.example.helloj	2020-05-22 12:54:00	admin
<input type="checkbox"/>				<div>IOS</div> <div>test_ios.a</div>		2020-05-22 12:54:00	

Name

pluginTest

Version

1.0

Owner

admin

Description

pluginTest

☒ IOS

libHelloPlugin.a

☒ Android

libhelloworldplugin.jar

Android Class

com.example.helloplugin.HelloPlugin



## 20.

# 넥사크로 라이브러리 설정

App Builder 관련 문서

- 서버 설정/개발 가이드 > App Builder

[App Builder 설치](#)

[서버 환경 설정](#)

[디플로이 서버 설정](#)

[운영체제별 서명 설정](#)

[사용자 라이브러리 설정](#)

[넥사크로 라이브러리 설정](#)

[사용자 설정](#)

- 앱 배포 가이드 > App Builder

[App 생성](#)

[App 목록](#)

- 개발도구 가이드 > Build App 단계에서 설치 파일 만들기

[Build App 단계에서 설치 파일 만들기](#)

넥사크로 버전별 라이브러리 파일을 관리합니다. 앱 빌드 시 개발에 사용한 넥사크로 버전과 라이브러리 버전이 같지 않은 경우 실행 시 오류가 발생할 수 있습니다.

메뉴 [Settings > Nexacro Library]에서 새로운 라이브러리 파일을 등록하거나 목록을 확인할 수 있습니다.

Nexacro Library

Signing

User Library

Deploy

Name

New

Save

<input type="checkbox"/>	Name	Version	Owner	Date
<input type="checkbox"/>	210809(21.0.0.100)	21.0.0.100	admin	2021-08-09 11:42:38
<input type="checkbox"/>	210806(21.0.0.100)	21.0.0.100	admin	2021-08-06 13:27:02
<input type="checkbox"/>	210805(21.0.0.100)	21.0.0.100	admin	2021-08-06 11:21:16
<input type="checkbox"/>	nexacroN_20210824_1	21.0.0.100	admin	2021-08-06 11:17:26

General

\*Name

210809(21.0.0.100)

\*Version

21.0.0.100

\*Owner

admin

\*Nexacro Framework file

nexacrolib.zip

Version

21.0.0.1

Android

\*Android Library File

NexacroN\_Android\_20210809\_1.zip



관리자 권한을 가진 사용자가 등록한 라이브러리는 사용자 모두가 확인하고 앱 빌드 시 사용할 수 있습니다. 하지만 개별 사용자가 등록한 라이브러리는 해당 사용자와 관리자만 확인하고 사용할 수 있습니다.

라이브러리 파일은 제품과 함께 배포됩니다. 지정된 파일 그대로 업로드한 후 사용합니다.

General

1

Name

210809(21.0.0.100)

2

Version

21.0.0.100

3

Owner

admin

4

Nexacro Framework file

nexacrolib.zip

5

Version

21.0.0.1

6

Android

Android Library File

NexacroN\_Android\_20210809\_1.zip

Version

21.0.0.100

BuildProcessor

NRE for Android support(21.0.0.100 ~ )

iOS

iOS Library File

NexacroN\_iOS\_20210809\_1.zip

Version

21.0.0.100

BuildProcessor

NRE for iOS support(21.0.0.100 ~ )

macOS

macOS Library File

NexacroN\_macOS\_20210809\_1.zip

Version

21.0.0.100

BuildProcessor

NRE for macOS support(21.0.0.100 ~ )

7

Description

	항목	설명
1	Name	라이브러리 이름을 입력합니다.
2	Version	라이브러리 버전을 입력합니다.
3	Owner	소유자 정보를 표시합니다.
4	Nexacro Framework File	배포되는 nexacrolib.zip 파일을 업로드합니다. 업로드한 항목은 마우스로 클릭 시 파일을 내려받을 수 있습니다.
5	Version	업로드한 Nexacro Framework File 버전입니다. Nexacro Framework File에 포함된 JSON 파일에서 버전 정보를 확인하고 표시합니다.
6	Library File	라이브러리 파일을 업로드합니다. 예) Android : nexacroN_Android_2021xxxx_1.zip iOS : nexacroN_iOS_2021xxxx_1.zip macOS : nexacroN_macOS_2021xxxx_1.zip

	항목	설명
		업로드한 항목은 마우스로 클릭 시 파일을 내려받을 수 있습니다. <ul style="list-style-type: none"><li>• Version: Library File에 포함된 JSON 파일에서 버전 정보를 확인하고 표시합니다.</li><li>• BuildProcessor: 라이브러리 Version 정보에 따라 빌드프로세서를 선택해서 표시합니다.</li></ul>
7	Description	라이브러리에 대한 설명을 입력합니다.



# 21.

## 사용자 설정

App Builder 관련 문서

- 서버 설정/개발 가이드 > App Builder

[App Builder 설치](#)

[서버 환경 설정](#)

[디플로이 서버 설정](#)

[운영체제별 서명 설정](#)

[사용자 라이브러리 설정](#)

[넥사크로 라이브러리 설정](#)

[사용자 설정](#)

- 앱 배포 가이드 > App Builder

[App 생성](#)

[App 목록](#)

- 개발도구 가이드 > Build App 단계에서 설치 파일 만들기

[Build App 단계에서 설치 파일 만들기](#)

App Builder를 사용할 사용자를 관리합니다. 등록된 사용자 정보는 관리 콘솔에 접근하거나 넥사크로 스튜디오에서 Deploy 시 App Builder를 사용하는 경우 로그인 시 사용합니다.

메뉴 [Configuration> User Management]에서 새로운 라이브러리 파일을 등록하거나 목록을 확인할 수 있습니다.

Server Configuration

User Management

User ID

New

Save

<input type="checkbox"/>	ID	Name	Department	Email	Authority	Active
<input type="checkbox"/>	admin	Administrator			Admin	<input type="radio"/>
<input type="checkbox"/>	user1	user1	Server Team	user1@nexacro.com	Developer	<input type="radio"/>
<input type="checkbox"/>	user2	user2	Server Team	user2@nexacro.com	Developer	<input type="radio"/>

\*User ID

1

user1

\*User Name

2

user1

E-mail

3

user1@nexacro.com

\*Active

7

☐

\*Password

4

\*\*\*\*\*

Department

5

Server Team

\*Authority

6

Developer

	항목	설명
1	User ID	사용자 ID를 입력합니다.
2	User Name	사용자 이름을 입력합니다.
3	E-mail	사용자 이메일 주소를 입력합니다.
4	Password	사용자 비밀번호를 입력합니다.
5	Department	사용자 소속 부서를 입력합니다.
6	Authority	사용자 권한을 선택합니다. - Admin: 관리자 권한 - Developer: 개발자 권한 관리자 권한은 모든 메뉴와 전체 앱 목록을 확인할 수 있습니다..
7	Active	사용 여부를 선택합니다. X로 지정한 경우에는 로그인할 수 없습니다.