응용 개발 가이드

21.0.0.300

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 | 2024/06/12
주소 | (06083) 서울시 강남구 봉은사로 617 인탑스빌딩 2-5층
전화 | 02-2140-7700
홈페이지 | www.tobesoft.com

고객지원센터 | support.tobesoft.co.kr
제품기술문의 | 1588-7895 (오전 10시부터 오후 5시까지)
유지보수정책 | 유지보수기간과 범위는 제품 라이선스 계약에 따라 다릅니다.

변경 이력

버전	변경일	내용
21.0.0.300.3	2024-03-29	앱 캐시 설명 중 cachelevel 속성값에 대한 잘못된 설명을 삭제했습니다.
21.0.0.300.2	2023-11-30	마이크로소프트 LUIS 서비스가 2023년 4월부터 새로운 리소스 생성을 중단하며, 2025년 10월에는 서비스를 종료할 예정입니다. 이로 인해 LUIS 서비스 기반으로 한 음성인식모듈(VoiceRecognition.xmodule)의 추가 배포가 중단되며, 관련 콘텐츠인 "응용 개발 가이드 > 음성인식 모듈 사용하기" 게시를 중단합니다. 자세한 내용은 마이크로소프트 공식 문서에서 확인하실 수 있습니다.
21.0.0.300	2021-11-23	넥사크로 스튜디오의 New Protocol Wizard 내 "HTML5" 문구를 "Web Browser"로 수정했습니다.
21.0.0.100.1	2021-10-12	Dataset SSV Format 항목 내 SSV 약자에 대한 설명을 수정했습니다.
21.0.0.100	2021-08-23	

차례

저작권 및 면책조항 2

변경 이력 3

차례 4

파트 I. 고급 활용 9

1. 프로토콜 어댑터 개발하기 10

1.1 프로토콜 어댑터 모듈 만들고 앱에서 사용하기 10

1.1.1 프로토콜 어댑터 모듈 만들기 10

1.1.2 프로토콜 어댑터 모듈 편집하기 11

1.1.3 프로토콜 어댑터 모듈 배포하기 13

1.1.4 모듈 설치하고 프로토콜 서비스 등록하기 14

1.1.5 프로토콜 어댑터를 통해 서비스 호출하기 15

1.2 추가로 설정할 수 있는 인터페이스 함수 16

1.2.1 재정의할 수 있는 인터페이스 함수 17

1.2.2 메소드로 추가하는 함수 18

2. X-PUSH 앱 개발하기 20

2.1 프로젝트에서 사용할 XPush 오브젝트 등록하기 20

2.2 실시간 메시지를 Grid 컴포넌트에 표시하기 21

2.3 신뢰성 메시지 처리하기 24

2.3.1 신뢰성 메시지 수신을 위해 토픽 등록하기 24

2.3.2 신뢰성 메시지 수신하기 25

2.3.3 등록된 토픽 취소하기 26

3. 위젯 앱 개발하기 28

3.1 간단한 달력 위젯 만들기 28

3.2 바로 실행되는 위젯 만들기 31

3.3 트레이에서 위젯 제어하기 33

4. 윈도우 운영체제 NRE에서 단위 테스트 실행하기 38

4.1 WinAppDriver 연동 환경 설정하기 38
4.1.1 WinAppDriver 설치하기 38
4.1.2 개발 도구 설치하기 41
4.2 단위테스트 만들고 실행하기 43
4.2.1 넥사크로 앱 만들고 배포하기 43
4.2.2 단위테스트 프로젝트 만들기 44
4.2.3 단위테스트 실행하기 55
4.3 AccessibilityId 속성값 확인하기 56
4.3.1 컴포넌트 속성값 확인하기 56
4.3.2 컨트롤, 아이템 속성값 확인하기 56
4.3.3 Inspect로 속성값 확인하기 57
4.4 기능별 데모 단위 테스트 실행하기 60
4.5 지원 인터페이스 61
4.5.1 Session 62
4.5.2 Timeout 62
4.5.3 Frame 63
4.5.4 Window 63
4.5.5 Element 65
4.5.6 ScreenShot 71
4.6 관련 링크 정보 72
5. 안드로이드 운영체제 NRE에서 단위 테스트 실행하기 74
5.1 Appium 연동 환경 설정하기 74
5.1.1 Appium 설치하고 단말기 연결 확인하기 74
5.1.2 테스트 프로젝트 실행 환경 설정하고 테스트하기 78
5.2 단위 테스트 만들고 실행하기 81
5.3 지원 인터페이스 82

파트 II. 동작 원리 84

6. 프로젝트 및 파일 구조 85
6.1 프로젝트 생성 85
6.1.1 프로젝트 설정 파일 (XPRJ) 86
6.1.2 Application 설정 파일 (XADL) 86
6.1.3 Type Definition 87
6.1.4 Environment 88
6.1.5 AppVariables 89
6.2 리소스 89
6.2.1 initValue 89
6.2.2 테마 (XTHEME) 90

6.2.3 ImageResource 90
6.2.4 UserFont 91
6.3 기타 91
6.3.1 Form (XFDL) 91
6.3.2 스크립트 (XJS) 92
6.3.3 스타일 설정 파일 (XCSS) 93
7. 앱 구동 시나리오 94
7.1 bootstrap 94
7.2 Application 로딩 95
7.3 폼 로딩 96
8. 넥사크로플랫폼 스크립트 언어 97
8.1 유효범위(Scope) 97
8.1.1 로컬 (local) 97
8.1.2 this 98
8.1.3 Global 99
8.1.4 Expr 102
8.1.5 lookup 103
8.2 이벤트 핸들러 104
8.2.1 메소드 104
8.2.2 오브젝트 타입 105
8.3 Setter 106
8.3.1 set 메소드 106
8.3.2 동적인 속성 107
8.4 기타 변경 사항 107
8.4.1 nexacro 메소드 107
8.4.2 동작 방식 변경 108
8.4.3 오브젝트 명 생성 시 제약 109
8.4.4 변수명, 함수명 생성 시 제약 109
8.4.5 추가 모듈에서 window 오브젝트 사용 시 제약 110
9. Frame Tree 111
9.1 Application 111
9.1.1 Script 예시 화면 112
9.1.2 application에서 form 오브젝트 접근 112
9.1.3 form에서 상위 오브젝트 접근 113
9.2 Form 113
9.2.1 Script 예시 화면 115
9.2.2 component / object / bind 116
9.2.3 container component 116

9.2.4 form을 연결한 container component 117
9.2.5 parent 117

파트 III. 참고 118

10. 앱 캐시 119
10.1 캐시의 종류 119
10.2 캐시 적용 방법 121

11. Dataset XML Format 122
11.1 Dataset XML layout 122
11.1.1 XML 선언 123
11.1.2 XML 예 123
11.2 Dataset 요소 124
11.2.1 Root 124
11.2.2 Parameters 124
11.2.3 Parameters > Parameter 125
11.2.4 Dataset 126
11.2.5 Dataset > ColumnInfo 127
11.2.6 Dataset > ColumnInfo > ConstColumn 128
11.2.7 Dataset > ColumnInfo > Column 129
11.2.8 Dataset > Rows 130
11.2.9 Dataset > Rows > Row 130
11.2.10 Dataset > Rows > Row > Col 131
11.2.11 Dataset > Rows > Row > OrgRow 132
11.2.12 Dataset > Rows > Row > OrgRow > Col 132

12. Dataset SSV Format 134
12.1 Dataset SSV layout 134
12.1.1 Stream Header 135
12.1.2 Variables 135
12.1.3 Datasets 136
12.2 Dataset 형식 상세 136
12.2.1 Dataset Header 136
12.2.2 Const Column Infos 137
12.2.3 Column Infos 137
12.2.4 Records 138
12.2.5 Null Record 138
12.3 참고 138
12.3.1 Type 138
12.3.2 예제 139

13. Dataset JSON Format 141
13.1 Dataset JSON layout 141
13.1.1 version 141
13.1.2 Parameters 141
13.1.3 Datasets 142
13.2 Dataset 형식 상세 143
13.2.1 id 143
13.2.2 ColumnInfo > ConstColumn 143
13.2.3 ColumnInfo > Column 144
13.2.4 Rows 144
13.3 참고 145
13.3.1 type 145
13.3.2 JSON 예 146

파트 I.

고급 활용

1.

프로토콜 어댑터 개발하기

넥사크로에서 기본적으로 제공하는 데이터 통신 방식은 HTTP 프로토콜을 사용해 요청과 응답을 처리합니다. 이 과정에서 데이터 자체를 변환하지는 않습니다. 하지만 사용 환경에 따라 데이터를 암호화하거나 특정 형식의 데이터를 수신하고 변환하는 과정이 필요할 수 있습니다.

프로토콜 어댑터를 사용하면 간단한 설정만으로 데이터를 암호화하거나 데이터 구조를 변환해 원하는 동작을 구현할 수 있습니다.

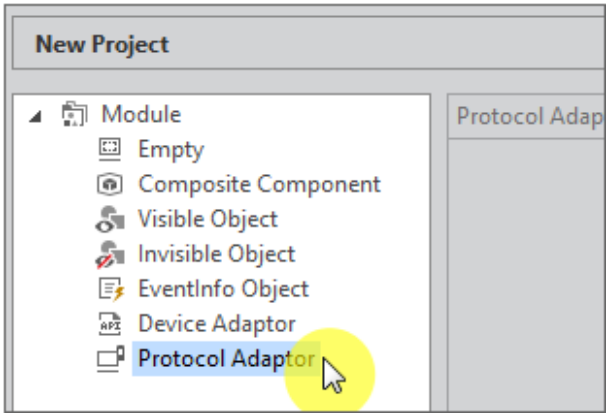
1.1 프로토콜 어댑터 모듈 만들고 앱에서 사용하기

프로토콜 어댑터를 사용하기 위해서는 넥사크로 모듈 디벨로퍼에서 모듈을 만들고 넥사크로 스튜디오에 작성된 모듈을 등록하고 프로토콜, 서비스를 추가해야 합니다.

1.1.1 프로토콜 어댑터 모듈 만들기

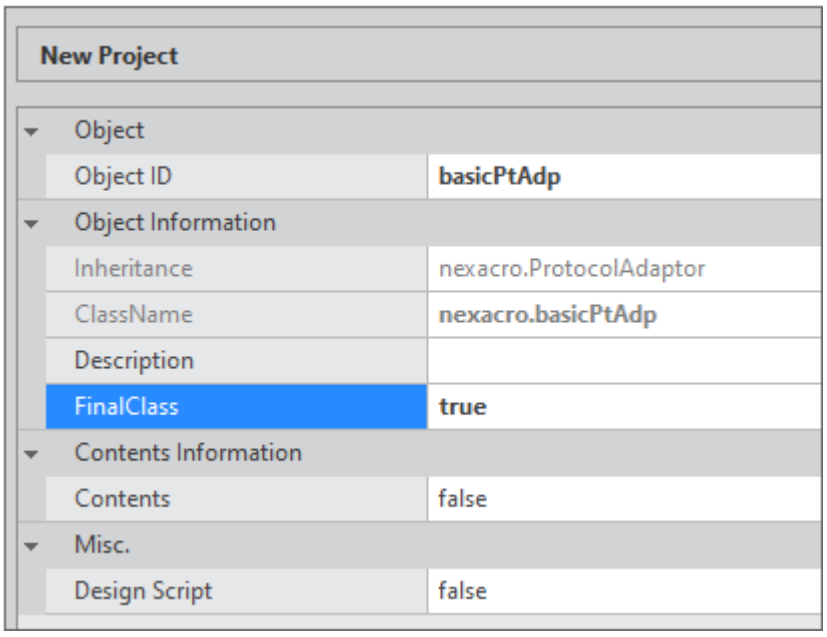
1. 넥사크로 모듈 디벨로퍼에서 새로운 프로젝트를 만듭니다. 메뉴에서 [File > New > Project]를 선택합니다.

2. Project Wizard가 열리면 Module 항목에서 "Protocol Adaptor"를 선택합니다.

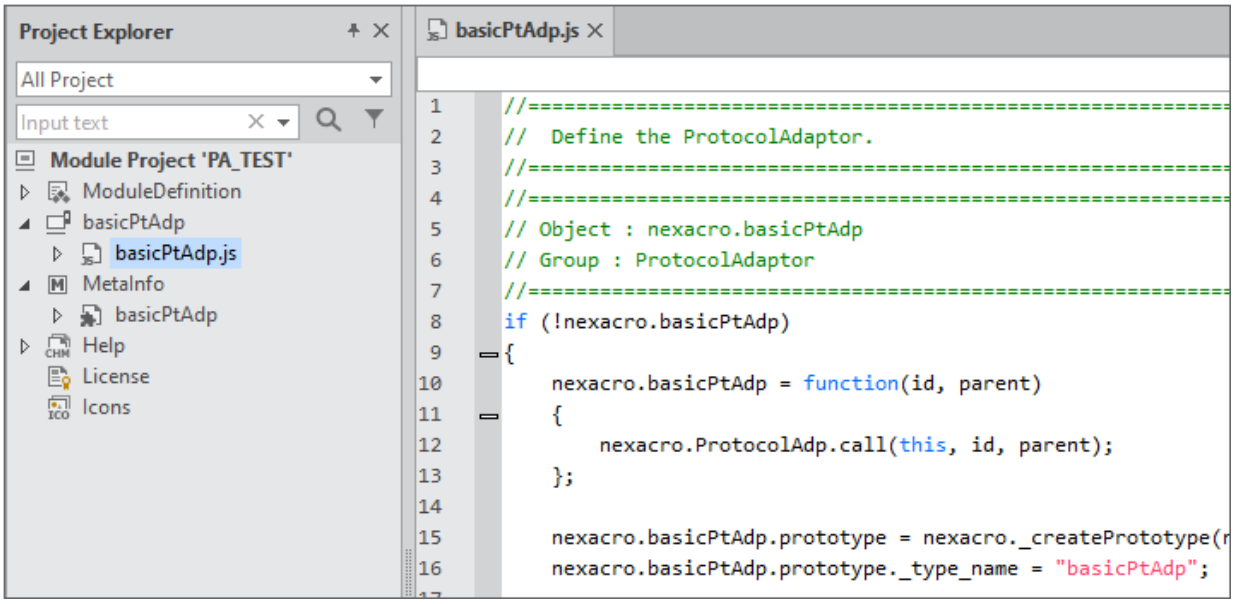


3 프로젝트 ID를 입력하고 [Next] 버튼을 클릭합니다.

4 Object ID를 입력하고 FinalClass 항목을 true로 변경합니다.



5 프로젝트가 생성되고 basicPtAdp.js 파일이 편집창에 표시됩니다.



1.1.2 프로토콜 어댑터 모듈 편집하기

프로토콜 어댑터 모듈을 새로 만들면 기본적으로 4개의 함수가 생성됩니다.

Function	설명
initialize	오브젝트가 생성되는 시점에 initialize 함수가 호출됩니다. 외부 모듈이 필요한 경우 initialize 함수 내에서 모듈을 생성하거나 초기화합니다.
finalize	오브젝트가 삭제되는 시점에 finalize 함수가 호출됩니다. 외부 모듈을 생성했거나 마무리가 필요한 경우 finalize 함수 내에서 생성된 모듈을 삭제하고 메모리에 남지 않도록 처리합니다.
encrypt	통신이 실행되기 전에 encrypt 함수가 호출됩니다. 전송할 데이터를 암호화하거나 원하는 형태로 변환할 수 있습니다.
decrypt	통신이 끝난 후에 decrypt 함수가 호출됩니다. 응답으로 수신한 데이터를 복호화하거나 원하는 형태로 변환할 수 있습니다.

initialize

```
nexacro.basicPtAdp.prototype.initialize = function()
{
    trace("basicPtAdp initialize");
};
```

외부 플러그인 모듈을 사용한 경우에는 아래와 같이 PluginElement를 사용해 초기화합니다.

```
nexacro.basicPtAdp.prototype.initialize = function()
{
    this._plugin = new nexacro.PluginElement();
    this._plugin.setElementClassId("Microsoft.XMLDOM");
    //this._plugin.setElementClassId("{7E9FDB80-5316-11D4-B02C-00C04F0CD404}");
    this._plugin.create();
};
```

finalize

```
nexacro.basicPtAdp.prototype.finalize = function()
{
    trace("basicPtAdp finalize");
};
```

initialize 함수에서 생성한 외부 모듈을 삭제 처리합니다.

```
nexacro.basicPtAdp.prototype.finalize = function()
{
    this._plugin.destroy();
    delete this._plugin;
}
```

encrypt

전송할 데이터를 변환하고 return 하는 문자열 변수에 담아서 처리합니다.

```
nexacro.basicPtAdp.prototype.encrypt = function(strUrl, strData)
{
    trace("encrypt url=" + strUrl + ";data=" + strData);
    return strData;
};
```

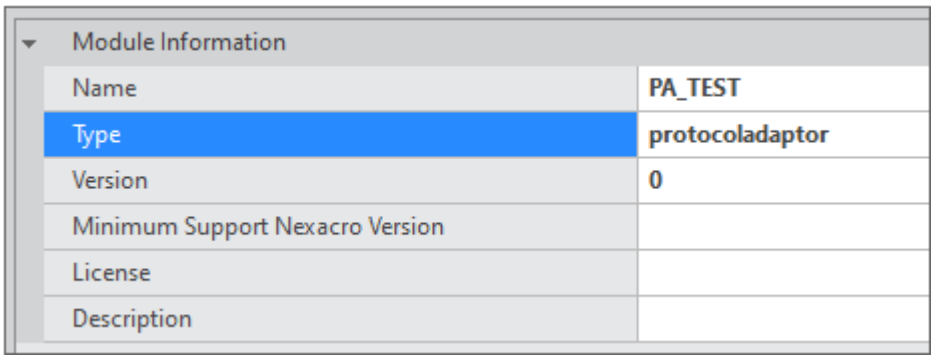
decrypt

응답으로 받은 데이터를 변환하고 호출한 앱에 전달합니다.

```
nexacro.basicPtAdp.prototype.decrypt = function(strUrl, strData)
{
    trace("decrypt url=" + strUrl + ";data=" + strData);
    return strData;
}
```

1.1.3 프로토콜 어댑터 모듈 배포하기

1. 넥사크로 모듈 디벨로퍼 메뉴[Deploy > Module Package]를 선택하고 Deploy Wizard를 실행합니다.
2. Module Information에서 Type 항목을 "protocoladaptor"으로 변경합니다.



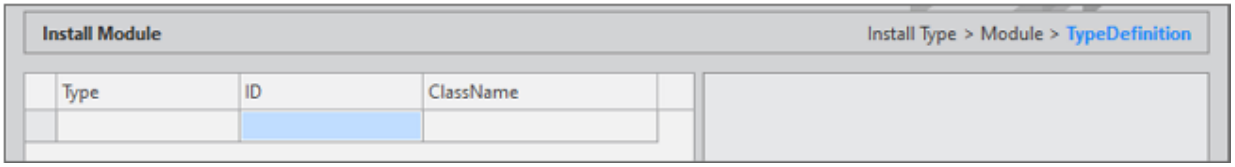
③ [Deploy] 버튼을 클릭해 모듈을 배포합니다.

1.1.4 모듈 설치하고 프로토콜 서비스 등록하기

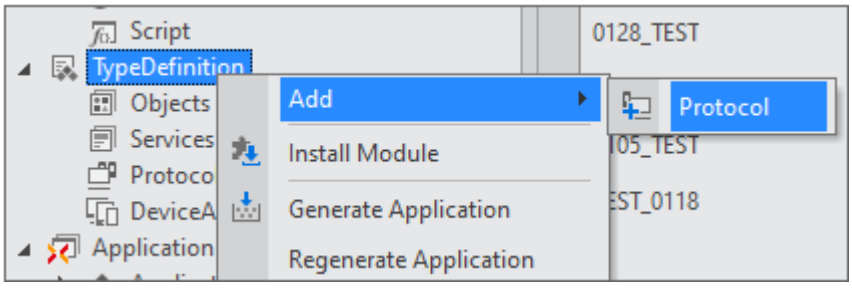
① 넥사크로 스튜디오 메뉴[File > Install Module]을 선택하고 Install Module Wizard를 실행합니다.

② 넥사크로 모듈 디벨로퍼에서 만든 모듈 설치 파일을 선택하고 설치합니다.

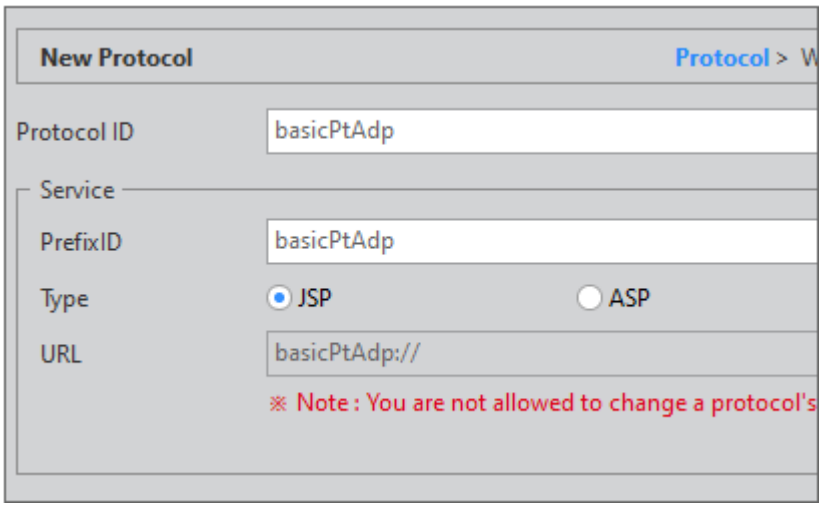
> 모듈 메타인포 속성 중 registration 속성값이 기본값("deny")으로 설정됐기 때문에 설치될 모듈 목록이 보이지는 않습니다.



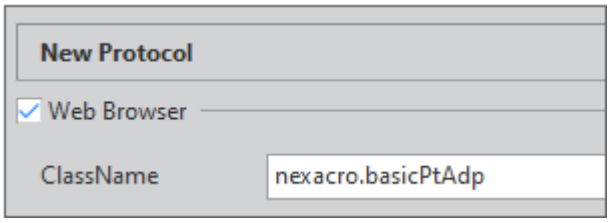
③ Project Explorer에서 TypeDefinition 항목을 선택하고 컨텍스트 메뉴에서 [Add > Protocol]을 선택합니다.



④ New Protocol Wizard에서 Protocol ID와 Prefix ID를 설정합니다.



5 [Next] 버튼을 클릭하고 Web Browser 항목을 체크하고 ClassName을 입력합니다.



1.1.5 프로토콜 어댑터를 통해 서비스 호출하기

transaction 메소드 실행 시 아래와 같이 strURL 파라미터에 프로토콜 어댑터 URL 형식을 적용하면 통신 요청이 바로 서버로 전달되지 않고 프로토콜 어댑터를 거치게 됩니다.

```
this.Button00_onclick = function(obj:nexacro.Button,e:nexacro.ClickEventInfo)
{
    this.transaction("srv00", "basicPtAdp://127.0.0.1:4098/test.xml", "","dsOut=Dataset00","
value=a","fnCallback");
};

this.fnCallback = function(id, code, message)
{
    trace("Error["+code+"]:"+message);
}
```

Button 클릭 시 아래와 같은 순서로 프로토콜 어댑터 동작이 처리되는 것을 확인할 수 있습니다.

1. initialize

예제에서는 초기화는 없고 trace 메소드만 실행합니다.

2. encrypt
   strArgument 파라미터로 전달한 값이 처리되는 것을 확인합니다.
3. decrypt
   요청에 대한 응답 데이터를 확인합니다.
4. CallbackFunc
   정상 처리 후 콜백 함수가 동작하는 것을 확인합니다.

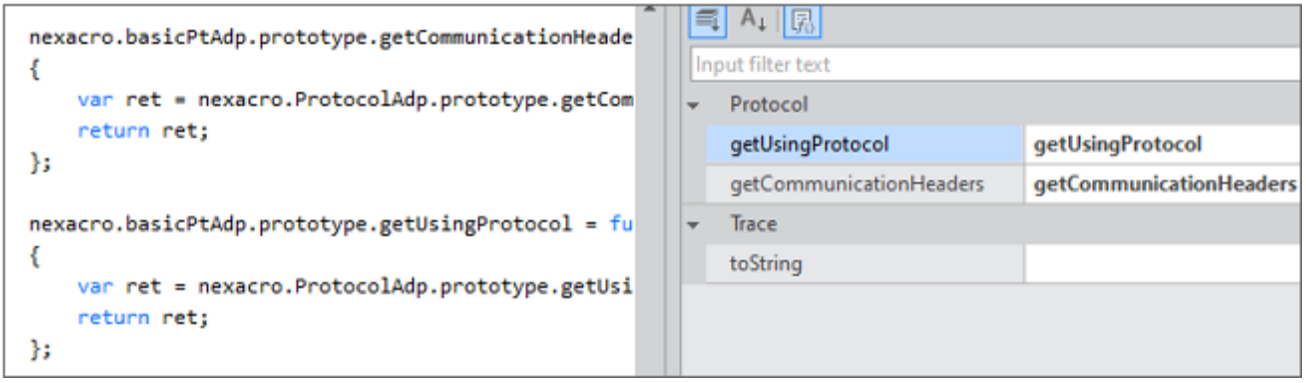
```
basicPtAdp initialize

encrypt url=basicPtAdp://127.0.0.1:8080/test.xml;data=<?xml version="1.0" encoding="UTF-8"?>
<Root xmlns="http://www.nexacroplatform.com/platform/dataset">
    <Parameters>
        <Parameter id="value">b</Parameter>
    </Parameters>
</Root>

decrypt url=basicPtAdp://127.0.0.1:8080/test.xml;data=<?xml version="1.0" encoding="utf-8"?>
<Root xmlns="http://www.nexacroplatform.com/platform/dataset" ver="5000" >
     <Parameters>
          <Parameter id="ErrorCode" type="int">0</Parameter>
          <Parameter id="ErrorMsg" type="string">SUCC</Parameter>
     </Parameters>
     <Dataset id="customers">
          <ColumnInfo>
               <Column id="id" type="STRING" size="4"/>
...

Error[0]:SAUCC
```

1.2 추가로 설정할 수 있는 인터페이스 함수

프로토콜 어댑터에서는 추가로 설정할 수 있는 인터페이스 함수를 제공합니다.

1.2.1 재정의할 수 있는 인터페이스 함수

상속받은 nexacro.ProtocolAdp에서 제공하는 함수를 재정의할 수 있습니다. 속성창에서 추가해 사용할 수 있습니다.



Function	설명
getUsingProtocol	프로토콜로 사용할 문자열을 설정합니다.
getCommunicationHeaders	통신 시 쿠키에 정보를 추가로 설정합니다.

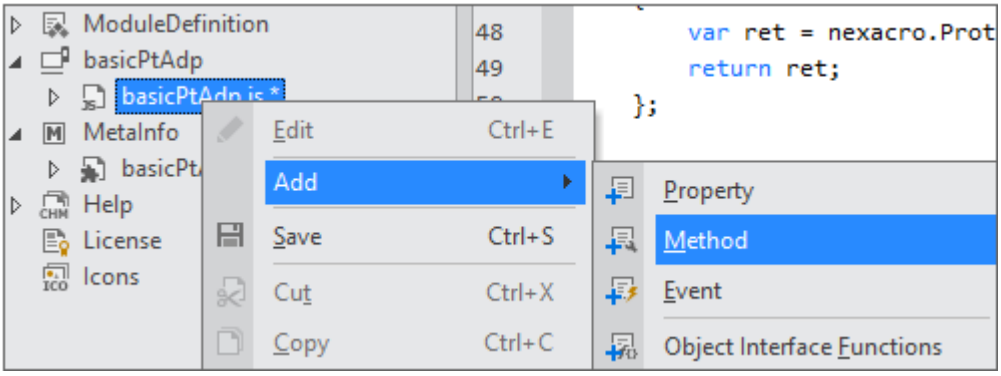
getCommunicationHeaders 함수는 아래와 같이 설정할 수 있습니다.

```
nexacro.basicPtAdp.prototype.getCommunicationHeaders = function (strUrl)
{
    var ret = nexacro.ProtocolAdp.prototype.getCommunicationHeaders.call(this, strUrl);
    ret.push({id:"PHPSESSID", value:"298zf09hf012fh2"});
    ret.push({id:"csrftoken", value:"u32t4o3tb3gg43"});
    return ret;
};
```

통신 시 쿠키 정보가 추가로 설정된 것을 확인할 수 있습니다.

1.2.2 메소드로 추가하는 함수

Project Explorer에서 오브젝트를 선택하고 컨텍스트 메뉴에서 Method를 추가해 설정합니다.



Function	설명
getHTTPHeader	HTTP 헤더값을 설정합니다. 원하는 헤더값의 id, value 값을 지정할 수 있습니다.
version	getHTTPHeader 함수를 사용하기 위해서는 "1.1"보다 큰 값으로 version 함수를 설정해주어야 합니다.

getHTTPHeader, version 함수는 아래와 같이 설정할 수 있습니다.

```
nexacro.basicPtAdp.prototype.initialize = function()
{
    this._httpheaders = [];
    this._httpheaders.push({ id: "Accept", value: "application/json, text/json, */*"});
};

nexacro.basicPtAdp.prototype.version = function ()
{
    return "1.1";
};

nexacro.basicPtAdp.prototype.getHTTPHeader = function ()
{
    return this._httpheaders;
};
```

통신 시 헤더 Accept 정보가 변경된 것을 확인할 수 있습니다.

```
▼ Request Headers        view source
    Accept: application/xml, text/xml, */*
    Accept-Encoding: gzip, deflate, br

▼ Request Headers        view source
    Accept: application/json, text/json, */*
    Accept-Encoding: gzip, deflate, br
```

2.

X-PUSH 앱 개발하기

X-PUSH는 실시간으로 정보나 메시지를 전달할 수 있는 Server push 기술을 사용하는 제품입니다. 넥사크로는 X-PUSH 서버에서 전송하는 메시지를 처리할 수 있는 XPush 오브젝트를 제공합니다. 간단한 설정만으로 메시지를 요청하고 처리할 수 있습니다. 이번 장에서는 넥사크로 스튜디오에서 XPush 오브젝트를 등록하고 메시지를 처리하는 앱을 구현하는 과정을 살펴봅니다.

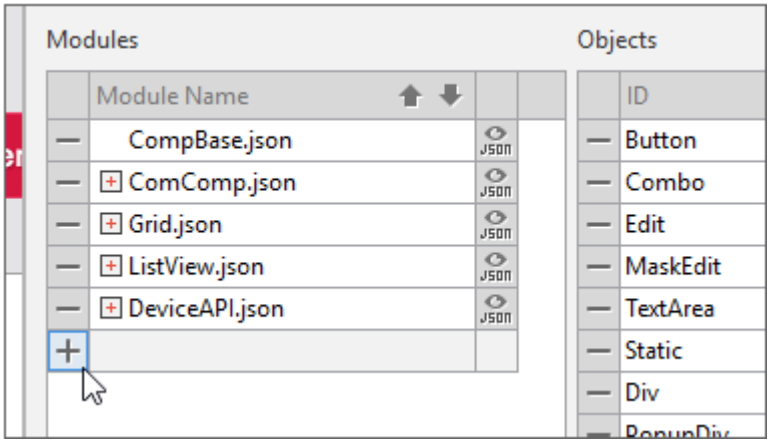
2.1 프로젝트에서 사용할 XPush 오브젝트 등록하기

XPush 오브젝트는 기본 오브젝트로 등록되어 있지 않습니다. Project Explorer 창에서 XPush 오브젝트를 등록한 후 사용할 수 있습니다.

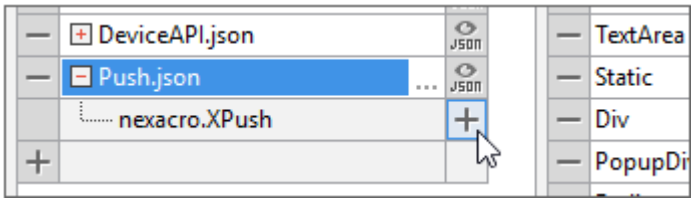
1. 메뉴[File > New > Project]를 선택하고 새로운 프로젝트를 생성합니다.

2. Project Explorer 창에서 [TypeDefinition > Objects] 항목을 선택합니다. Module을 추가할 수 있는 창이 나타납니다.

3. Modules 항목에서 [+] 버튼을 클릭합니다. 파일 탐색기가 나타나면 "Push.json" 파일을 선택합니다.



4 Modules 목록에 Push.json 항목이 추가된 것을 확인하고 nexacro.XPush 항목 옆에 있는 [+] 버튼을 클릭해 XPush 오브젝트를 Objects 목록에 추가합니다.



5 Objects 목록에 XPush 오브젝트가 추가된 것을 확인하고 [OK] 버튼을 클릭해 창을 닫습니다. XPush 오브젝트를 추가하면서 변경된 설정을 반영하기 위해 프로젝트를 새로 로딩합니다.

2.2 실시간 메시지를 Grid 컴포넌트에 표시하기

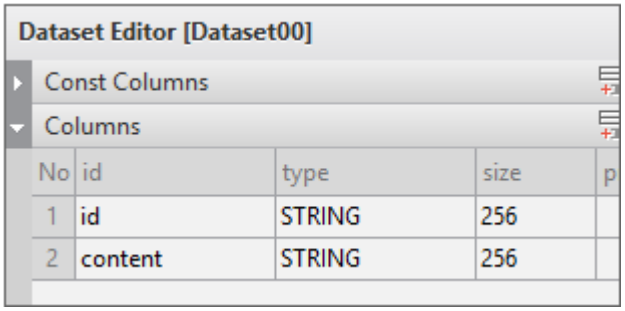
XPush 오브젝트를 추가해서 X-PUSH 서버에서 실시간으로 전송되는 메시지를 수신하고 수신된 메시지를 Grid 컴포넌트에 표시합니다.

> 실시간 메시지를 수신하기 위해서는 X-PUSH 서버가 동작하고 있으며 메시지 공급자(Message Provider) 서비스가 실행된 상태여야 합니다. 메시지 전송은 run_provider_demo_Push.bat 파일 실행 시 전송하며 실행을 멈출 때까지 계속 전송합니다.

1 메뉴[File > New > Form]를 선택하고 새로운 Form 화면을 생성합니다.

2 Form 화면에 Grid 컴포넌트와 Dataset 오브젝트를 추가합니다.

3 Dataset Editor에서 Dataset 오브젝트의 컬럼 속성을 아래와 같이 설정하고 Grid 컴포넌트에 바인딩합니다.



> Dataset 오브젝트의 컬럼 속성은 layouturl 속성값으로 지정하는 메시지 레이아웃 XML 파일에 정의된 id 값이 같아야 합니다.

```
...
<message type="OPDT">
    <field id="id" type="string" size="9"/>
    <field id="content" type="string" size="255"/>
</message>
...
```

4 컴포넌트 툴바에서 XPush 오브젝트를 확인하고 마우스 드래그앤드롭 동작으로 Form 화면에 XPush 오브젝트를 추가합니다. XPush 오브젝트는 Dataset 오브젝트처럼 Invisible Object 영역에 추가됩니다.

5 XPush 오브젝트를 선택하고 속성창에서 layouturl, iplist 속성값을 지정합니다. layouturl 속성값은 정의된 메시지 레이아웃 XML 파일의 URL값을 지정하거나 로컬 위치의 파일을 지정할 수 있습니다. iplist 속성값은 X-PUSH 서버 IP와 PORT 값을 지정합니다.

> iplist 속성값은 TCP 통신(NRE)과 HTTP 통신(웹브라우저) 중 하나를 선택할 수 있습니다.
> TCP 통신 방식을 지정하는 경우에는 PORT 번호를 10081로 지정합니다.
> HTTP 통신 방식을 지정하는 경우에는 PORT 번호를 10080으로 지정합니다.
> NRE와 웹브라우저 모두 서비스해야 하는 경우에는 iplist 속성값에 두 가지 PORT 설정을 모두 지정해야 합니다.
> `iplist="http://localhost:10080, localhost:10081"`

6 Form 오브젝트의 onload 이벤트 함수를 아래와 같이 추가합니다. onload 이벤트 함수에서는 X-PUSH 서버에 연결을 시도합니다.

```
this.Form_Work_onload = function(obj:nexacro.Form,e:nexacro.LoadEventInfo)
{
    this.XPush00.connect("user00", "password");
};
```

7 XPush 오브젝트의 onsuccess 이벤트 함수를 아래와 같이 추가합니다. 연결이 성공한 경우에 메시지 수신을 위해 subscribe 메소드를 실행합니다. 실시간으로 메시지를 수신해서 Dataset 오브젝트에 메시지 데이터를 추가합니다. 추가된 메시지는 Grid 컴포넌트를 통해 표시됩니다.

```
this.XPush00_onsuccess = function(obj:nexacro.XPush,e:nexacro.XPushEventInfo)
{
    if(e.action == nexacro.XPushAction.AUTH) // 0
    {
        this.XPush00.subscribe("OPDT", "ALL", this, this.Dataset00, "append", "fn_
PushCallback");
    }
};

this.fn_PushCallback = function(Row,ChangeColumn,AllColumn,Type,ActionType)
{
    trace(Row,ChangeColumn,AllColumn,Type,ActionType);
}
```

8 XPush 오브젝트의 onerror 이벤트 함수를 아래와 같이 추가합니다. 연결이 실패한 경우에는 원인을 확인할 수 있습니다.

```
this.XPush00_onerror = function(obj:nexacro.XPush,e:nexacro.XPushErrorEventInfo)
{
    trace(e.errormsg, e.statuscode);
};
```

9 앱을 실행하면 onload 이벤트 함수 내에서 X-PUSH 서버에 연결하고 전송되는 메시지를 수신하기 시작합니다. 실시간으로 수신된 메시지가 Grid 컴포넌트에 표시되는 것을 확인합니다.

id	content
ALL	pushed current time : 13:35:45
ALL	pushed current time : 13:35:47
ALL	pushed current time : 13:35:49
ALL	pushed current time : 13:35:51
ALL	pushed current time : 13:35:53
ALL	pushed current time : 13:35:55

2.3 신뢰성 메시지 처리하기

> Form 화면을 생성하고 XPush 오브젝트를 초기화하는 과정은 실시간 메시지를 Grid 컴포넌트에 표시하기와 같습니다. onload 이벤트 함수에서 달라지는 부분부터 설명합니다.

> 신뢰성 메시지를 수신하기 위해서는 X-PUSH 서버가 동작하고 있으며 메시지 공급자(Message Provider) 서비스가 실행된 상태여야 합니다. 메시지 전송은 run_provider_demo_Reli.bat 파일 실행 시 하나의 메시지를 전송합니다.

2.3.1 신뢰성 메시지 수신을 위해 토픽 등록하기

신뢰성 메시지(Reliable messaging)는 사용자에게 메시지를 전달하는 것을 보장하는 것을 의미합니다. 실시간 메시지는 메시지를 전송하고 사용자가 메시지를 수신했는지 여부를 확인하지 않습니다. 마치 라디오 방송처럼 계속 메시지를 전달하고 사용자가 이를 수신하는 것입니다. 하지만 신뢰성 메시지는 특정 사용자에게 보내는 음성 메시지와 비슷합니다. 메시지를 전송하는 시점에 확인하지 못했더라도 나중에 확인할 수 있도록 보장하는 것입니다.

신뢰성 메시지를 처리하기 위해서는 어떤 사용자가 어떤 메시지를 수신할 것인지 미리 약속되어 있어야 합니다. 토픽 등록하기를 통해 데이터베이스에 관련 정보를 등록합니다.

1. Form 오브젝트의 onload 이벤트 함수를 아래와 같이 추가합니다. onload 이벤트 함수에서는 X-PUSH 서버에 연결을 시도합니다.

```
this.Form_Work_onload = function(obj:nexacro.Form,e:nexacro.LoadEventInfo)
{
    this.XPush00.connect("user00", "password");
}
```

2. XPush 오브젝트의 onsuccess 이벤트 함수를 아래와 같이 추가합니다. 연결이 성공한 경우에 메시지 수신을 위해 subscribe 메소드를 실행합니다. 실시간으로 메시지를 수신해서 Dataset 오브젝트에 메시지 데이터를 추가합니다. 추가된 메시지는 Grid 컴포넌트를 통해 표시됩니다.

```
this.XPush00_onsuccess = function(obj:nexacro.XPush,e:nexacro.XPushEventInfo)
{
    if(e.action == nexacro.XPushAction.AUTH) // 0
    {
```

```
        this.XPush00.subscribe("NOTI", "ALL", this, this.Dataset00, "append", "fn_
PushCallback");
    }
};
```

③ Form 화면에 Button 컴포넌트를 추가하고 Button 컴포넌트의 onclick 이벤트 함수를 아래와 같이 추가합니다. connect 메소드 실행 시 사용한 userid로 토픽을 등록합니다. 토픽 등록은 한 번만 실행합니다.

```
this.Button00_onclick = function(obj:nexacro.Button,e:nexacro.ClickEventInfo)
{
    this.XPush00.registerTopic("NOTI", "ALL");
};
```

④ XPush 오브젝트의 onerror 이벤트 함수를 아래와 같이 추가합니다. 토픽 등록이 실패하거나 이미 등록된 토픽이 있는 경우에는 실패 원인을 확인할 수 있습니다.

```
this.XPush00_onerror = function(obj:nexacro.XPush,e:nexacro.XPushErrorEventInfo)
{
    trace(e.errormsg, e.statuscode);
};
```

⑤ 토픽이 등록되면 신뢰성 메시지를 수신할 수 있습니다. 데이터베이스 T_TOPIC 테이블에서 등록된 토픽을 확인할 수 있습니다.

SELECT * FROM T_TOPIC;

ID	USER_ID	TOPIC_TYPE	TOPIC_ID	REGISTER_DATE	ACTIVE
2	user00	NOTI	ALL	20181011160747	Y

(1 row, 2 ms)

2.3.2 신뢰성 메시지 수신하기

앱이 실행되지 않은 상태에서 메시지가 전송되면 해당 메시지는 데이터베이스 T_USER_MESSAGE 테이블에 저장됩니다. MESSAGE_STATE 컬럼값이 0이면 사용자에게 메시지가 전달되지 않은 상태입니다.

SELECT * FROM T_USER_MESSAGE;

ID	USER_ID	MESSAGE_ID	MESSAGE_STATE	CHECK_DATE
4	user00	387c[illegible]277523	0	*null*

(1 row, 2 ms)

1. Form 화면에 Button 컴포넌트를 추가하고 Button 컴포넌트의 onclick 이벤트 함수를 아래와 같이 추가합니다. connect 메소드 실행 시 사용한 userid로 미수신된 메시지가 있는지 확인하고 메시지를 수신합니다. 서버에서 미수신된 메시지를 전송하고 나면 MESSAGE_STATE 컬럼값이 1로 변경됩니다.

```
this.Button01_onclick = function(obj:nexacro.Button,e:nexacro.ClickEventInfo)
{
    this.XPush00.requestMessage("NOTI", "ALL");
};
```

2. requestMessage 메소드는 별도의 콜백함수를 지정하지 않고 subscribe 메소드 실행 시 지정한 콜백함수를 사용합니다. requestMessage 메소드를 실행해 미수신된 메시지를 수신하면 sendResponse 메소드를 실행해 해당 메시지 수신에 대한 응답을 보낼 수 있습니다. 서버에서 응답을 수신하면 MESSAGE_STATE 컬럼값이 2로 변경됩니다.

```
this.fn_PushCallback = function(Row,ChangeColumn,AllColumn,Type,ActionType,MessageId)
{
    trace(Row,ChangeColumn,AllColumn,Type,ActionType,MessageId);
    if(ActionType=="RELI")
        this.XPush00.sendResponse(MessageId);
}
```

2.3.3 등록된 토픽 취소하기

더 이상 해당 메시지를 수신할 필요가 없다면 등록된 토픽을 취소해주어야 합니다. unregisterTopic 메소드를 사용해 등록된 토픽을 취소할 수 있습니다.

1. Form 화면에 Button 컴포넌트를 추가하고 Button 컴포넌트의 onclick 이벤트 함수를 아래와 같이 추가합니다. connect 메소드 실행 시 사용한 userid로 토픽을 등록한 토픽을 취소합니다.

```
this.Button02_onclick = function(obj:nexacro.Button,e:nexacro.ClickEventInfo)
{
    this.XPush00.unregisterTopic("NOTI", "ALL");
};
```

2. 등록된 토픽을 취소하면 신뢰성 메시지를 수신할 수 없습니다. 데이터베이스 T_TOPIC 테이블에서 등록된 토픽 항목 중 ACTIVE 컬럼값이 "N"으로 변경된 것을 확인할 수 있습니다.

SELECT * FROM T_TOPIC;

ID	USER_ID	TOPIC_TYPE	TOPIC_ID	REGISTER_DATE	ACTIVE
2	user00	NOTI	ALL	20181012112527	N

(1 row, 9 ms)

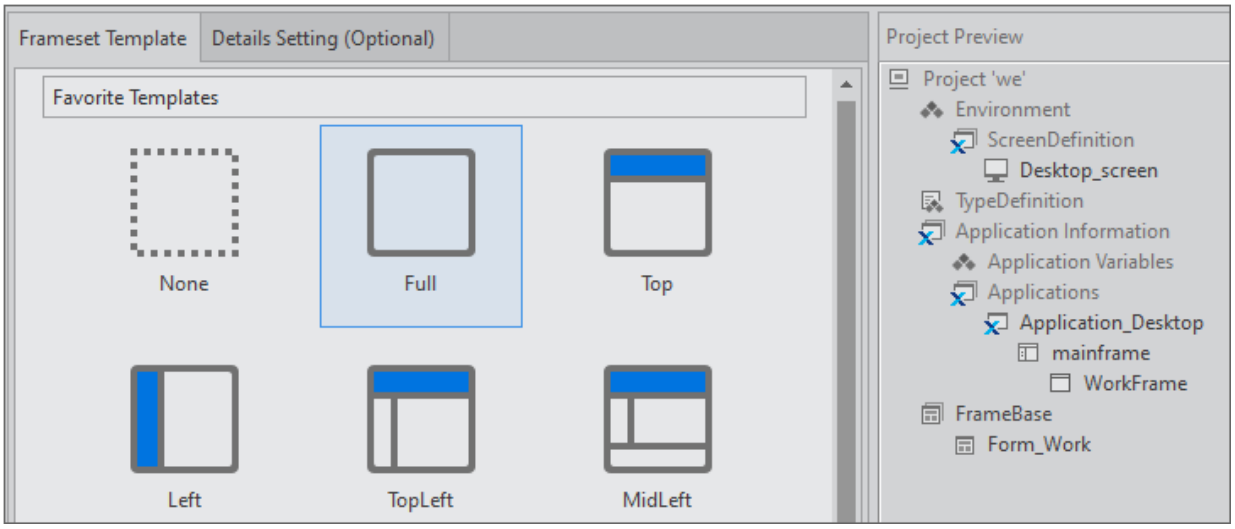
3.

위젯 앱 개발하기

위젯은 시계나 CPU 현황 같은 실시간 정보를 모니터링하기 위한 용도로 화면에 항상 띄워놓는 작은 UI 도구입니다. 넥사크로에서는 투명한 레이어를 가지는 새 창을 띄우는 방식으로 위젯 UI를 구현할 수 있습니다.

3.1 간단한 달력 위젯 만들기

1. 새로운 프로젝트를 생성합니다. Frameset Template에서 Full 타입을 선택합니다.



2. Form_Work 폼에 Button 컴포넌트를 추가하고 아래와 같이 onclick 이벤트 함수를 생성합니다.

   layered 속성은 ChildFrame 생성 시 배경 윈도우를 투명하게 설정하는 속성입니다. 이 속성값을 true로 설정하면 원하는 모양의 배경 이미지를 화면에 표현할 수 있습니다.

```
this.Button00_onclick = function(obj:nexacro.Button,e:nexacro.ClickEventInfo)
{
    nexacro.open("NewWidget",
        "FrameBase::WidgetMain1.xfdl",
```

```
        this.getOwnerFrame(), "",
        "showontaskbar=false showtitlebar=false showstatusbar=false topmost=true layered=
true",
        0, 0);
};
```

WidgetMain1.xfdl 폼은 위젯 형태로 보일 폼입니다. 위젯 형태로 필요한 UI만 노출하기 때문에 OpenStyle 파라미터 설정에서 태스크바나 타이틀바, 상태표시바는 보이지 않게 처리합니다. 그리고 다른 프로그램에 가리지 않도록 topmost 속성값을 true로 지정합니다.

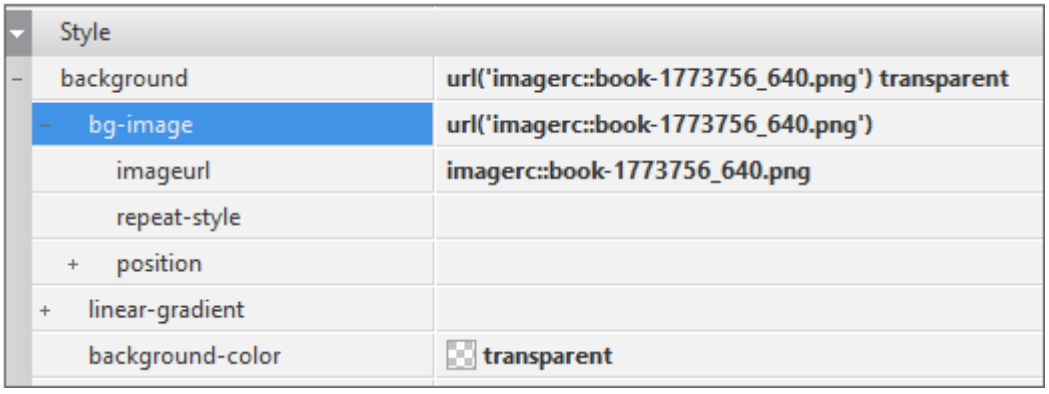
3 WidgetMain1 라는 이름으로 새로운 Form 오브젝트를 생성합니다.

4 ResourceExplorer 탭을 선택하고 컨텍스트 메뉴에서 'Import'를 선택합니다.

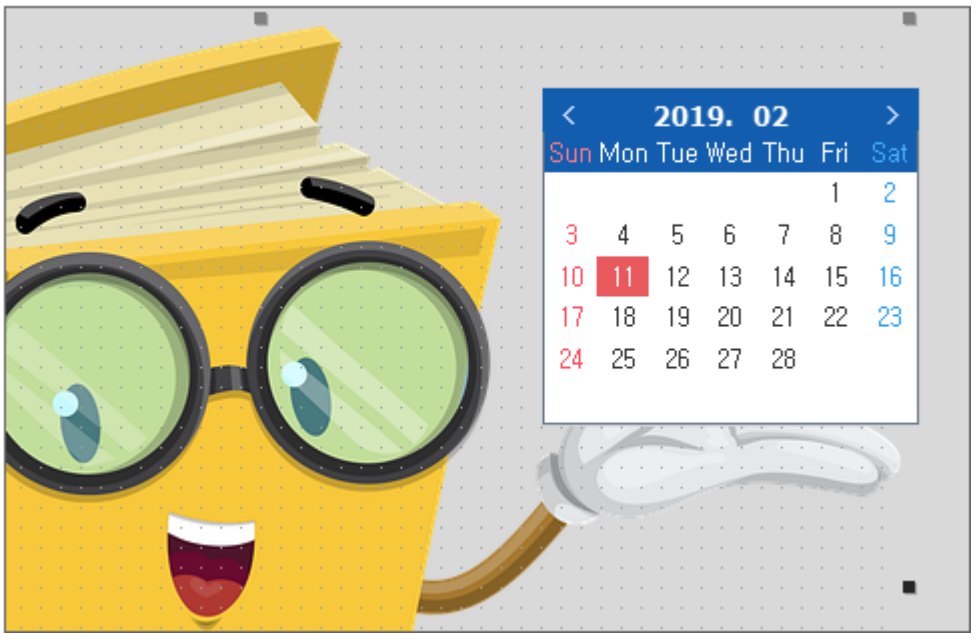
배경 이미지로 사용할 투명한 이미지 파일을 가져옵니다. 예를 들어 원 형태의 이미지라면 원 모양만 보여지고 바탕 영역은 투명하게 보여집니다. 예제에서는 아래 이미지를 사용했습니다.

> https://pixabay.com/en/book-character-glasses-show-1773756/

5 Form 오브젝트의 background 속성에서 imageurl 속성값으로 투명 이미지 파일을 선택하고 background-color 속성값은 transparent로 설정합니다.



6 투명 이미지 파일에 맞게 Form 오브젝트의 크기를 조정하고 Calendar 컴포넌트를 적절한 위치에 배치합니다.



7 툴바에서 실행옵션으로 NRE를 선택하고 Launch 아이콘을 클릭하거나 단축키 (Ctrl + F5)를 입력하고 앱을 실행합니다.

> Launch 실행 시 NRE 이외의 웹브라우저에서는 위젯 기능을 사용할 수 없습니다.

8 앱에서 버튼을 클릭하면 WidgetMain1 으로 만든 위젯 UI가 화면에 표시됩니다.

9 투명하지 않은 배경 이미지 영역을 마우스로 드래그하면 원하는 위치로 이동할 수 있습니다.

> 투명한 배경 영역 클릭 시에는 뒤에 보여지는 윈도우에 포커스가 이동하기 때문에 마우스로 드래그해서 위젯 위치를 변경할 수 없습니다.
> 개별 컴포넌트 클릭 시에는 개별 컴포넌트 동작을 해야 하기 때문에 마우스로 드래그해서 위젯 위치를 변경할 수 없습니다.

3.2 바로 실행되는 위젯 만들기

이전 예제에서는 메인 폼을 실행하고 버튼 클릭 시 위젯 UI가 나타났습니다. 이번 예제에서는 사용자의 입력 없이 바로 실행되는(것 처럼 보이는) 위젯을 만들어봅니다.

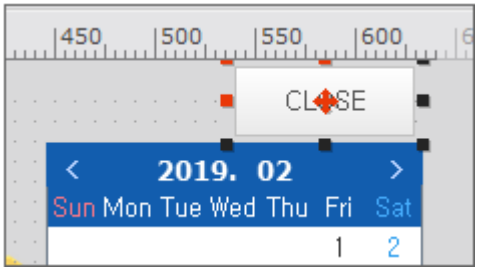
1 이전 예제에서 사용한 Form_Work 폼에 아래와 같이 oninit 이벤트 함수를 생성합니다.

```
this.Form_Work_oninit = function(obj:nexacro.Form,e:nexacro.EventInfo)
{
    nexacro.getApplication().mainframe.set_visible(false);
    nexacro.open("NewWidget",
        "FrameBase::WidgetMain1.xfdl",
        this.getOwnerFrame(), "",
        "showontaskbar=false showtitlebar=false showstatusbar=false topmost=true layered=
true",
```

```
        0, 0);
};
```

② 위젯을 종료하기 위한 Button 컴포넌트를 위젯 폼(WidgetMain1)에 추가합니다.

이전 예제에서는 메인 폼을 닫으면 위젯이 같이 종료되는데, 위젯에는 타이틀바가 표시되지 않아서 닫기 버튼을 따로 만들어주어야 합니다.



③ Button 컴포넌트를 추가하고 아래와 같이 onclick 이벤트 함수를 생성합니다.

```
this.Button00_onclick = function(obj:nexacro.Button,e:nexacro.ClickEventInfo)
{
    this.close();
};
```

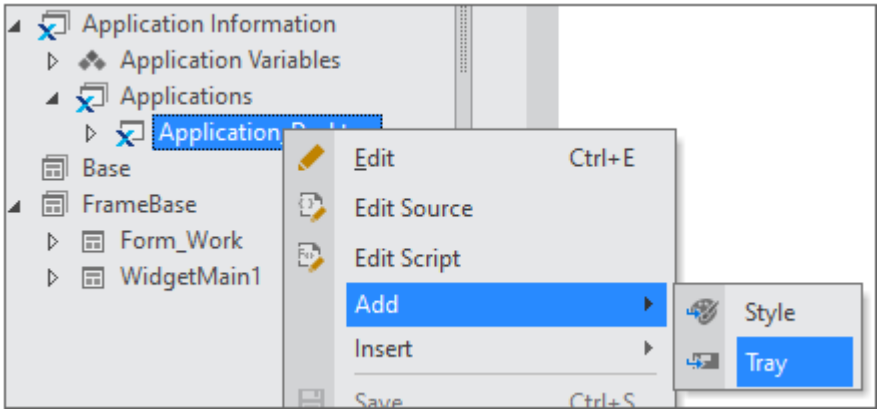
④ 툴바에서 실행옵션으로 NRE를 선택하고 Launch 아이콘을 클릭하거나 단축키 (Ctrl + F5)를 입력하고 앱을 실행합니다.

앱이 실행되면서 메인프레임을 보이지 않게 하면서 open 메소드가 실행되기 때문에 위젯 UI만 화면에 표시됩니다. [CLOSE] 버튼을 클릭하면 위젯 UI가 닫힙니다.

3.3 트레이에서 위젯 제어하기

위젯이 떠있으면 다른 작업을 할 때 방해가 될 수 있습니다. 그런 경우에는 잠시 위젯을 감추어두었다가 다시 보이도록 할 수 있습니다. 앱에 트레이를 등록하고 트레이에서 위젯을 제어할 수 있습니다.

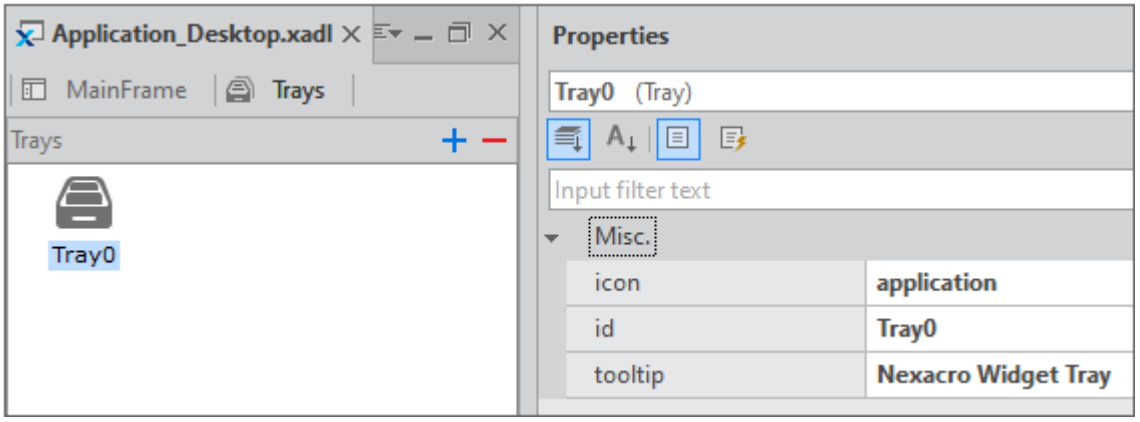
1. Project Explorer에서 App을 선택하고 컨텍스트 메뉴에서 [Add > Tray] 항목을 선택합니다.



2. 트레이 아이콘을 선택하고 속성창에서 icon, tooltip 속성값을 설정합니다.

```
icon: "application"
tooltip: Nexacro Widget Tray
```

> icon 속성값은 기본으로 제공되는 시스템 아이콘을 사용하거나 URL 값을 설정할 수 있습니다.

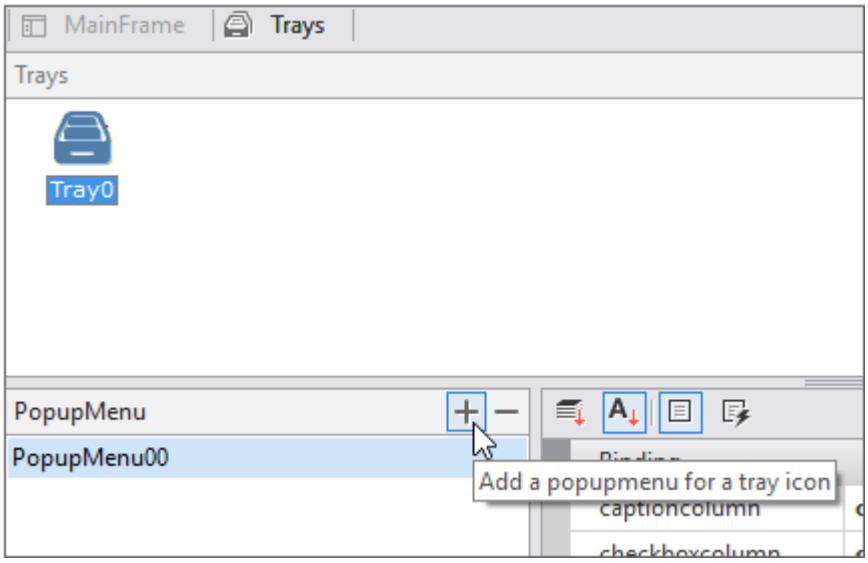


3. onrbuttonup 이벤트 함수를 아래와 같이 추가합니다.

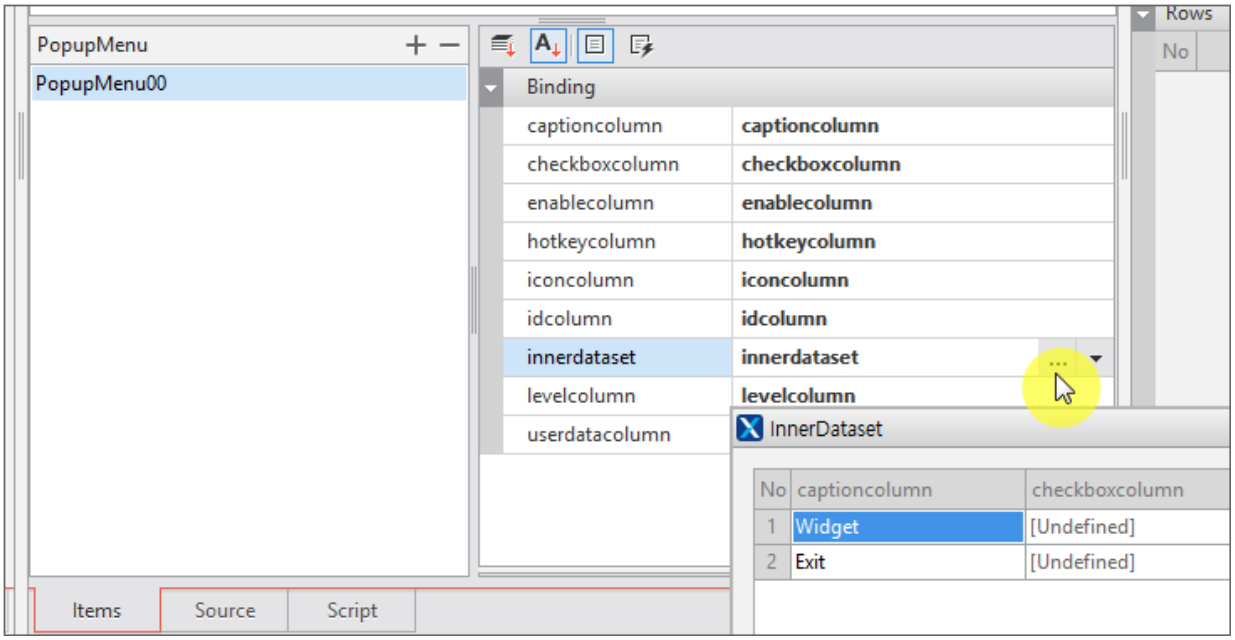
   "PopupMenu00"은 다음 단계에서 만들 PopupMenu 컴포넌트 id 값입니다. 트레이에서 마우스 오른쪽 버튼을 클릭했을 때 메뉴가 표시되도록 합니다.

```
this.Tray0_onrbuttonup = function(obj:nexacro.Tray,e:nexacro.MouseEventInfo)
{
    var pdivWigetMenu = obj.items[obj.findItem("PopupMenu00")];
    pdivWigetMenu.trackPopup()
};
```

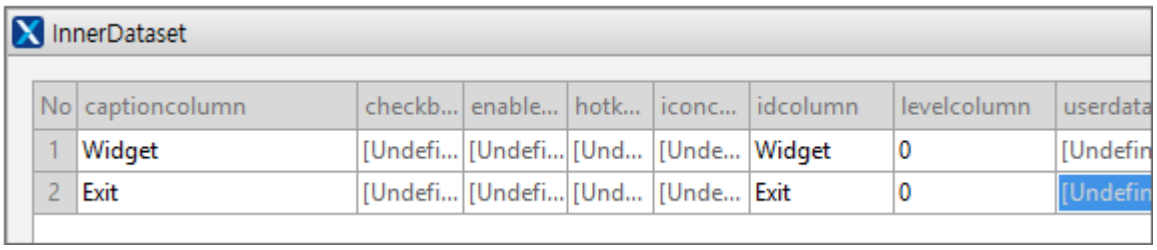
4 트레이 아이콘 아래에 PopupMenu 창에서 추가하기 버튼을 클릭합니다.



5 PopupMenu00 항목을 선택하면 오른쪽 속성창에서 innerdataset을 설정할 수 있습니다.



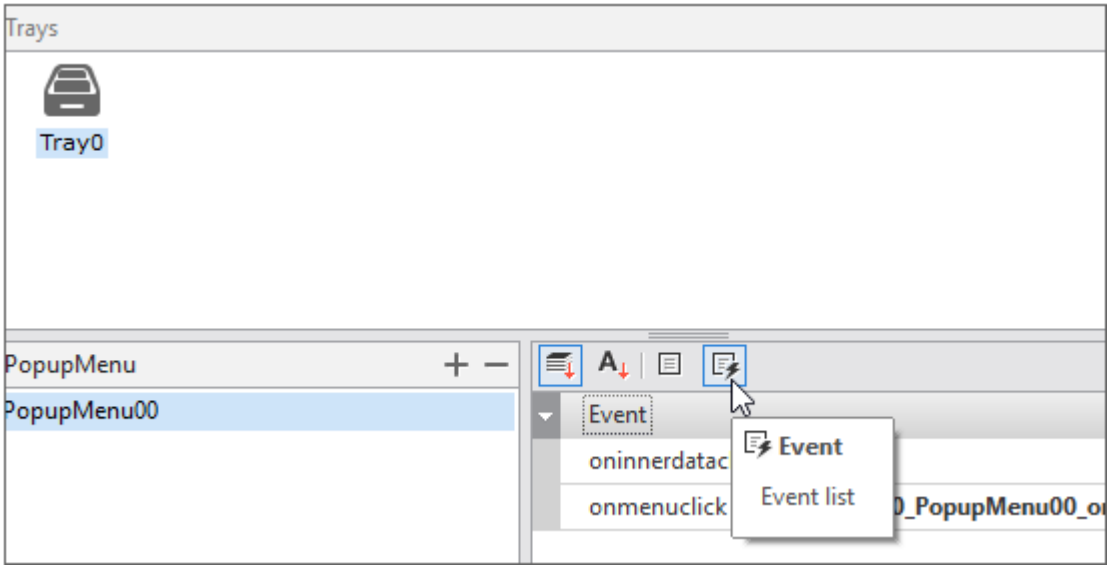
captioncolumn, idcolumn 항목은 "Widget", "Exit"로 지정하고 levelcolumn 항목은 0으로 지정합니다.



No	captioncolumn	checkb...	enable...	hotk...	iconc...	idcolumn	levelcolumn	userdata
1	Widget	[Undefi...	[Undefi...	[Und...	[Unde...	Widget	0	[Undefin
2	Exit	[Undefi...	[Undefi...	[Und...	[Unde...	Exit	0	[Undefin

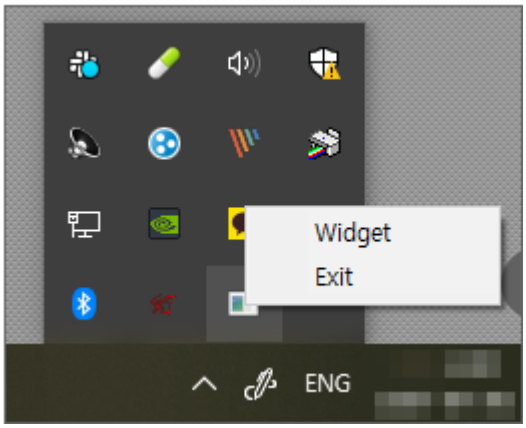
6 속성창에서 Event 아이콘을 선택하면 이벤트 함수를 추가할 수 있습니다. onmenuclick 이벤트 함수를 아래와 같이 추가합니다.

트레이에서 오른쪽 마우스 버튼을 클릭해서 메뉴를 표시하고 메뉴 항목을 선택했을 때의 동작을 지정하는 코드입니다. "Widget" 항목을 선택했을 때 위젯이 닫혀있는(보이지 않는) 경우에는 위젯을 보여주고 이미 위젯이 표시되고 있는 경우에는 로그만 기록합니다. "Exit" 항목을 선택한 경우에는 앱을 종료합니다.




```
this.Tray0_PopupMenu00_onmenuclick = function(obj:nexacro.TrayPopupMenu,e:nexacro.
MenuClickEventInfo)
{
    if(e.id == "Widget")
    {
        var widgetFrame = nexacro.getPopupFrames()["NewWidget"];
        if(widgetFrame.visible)
        {
            trace(widgetFrame);
        }
        else
        {
            widgetFrame.set_visible(true);
        }
```

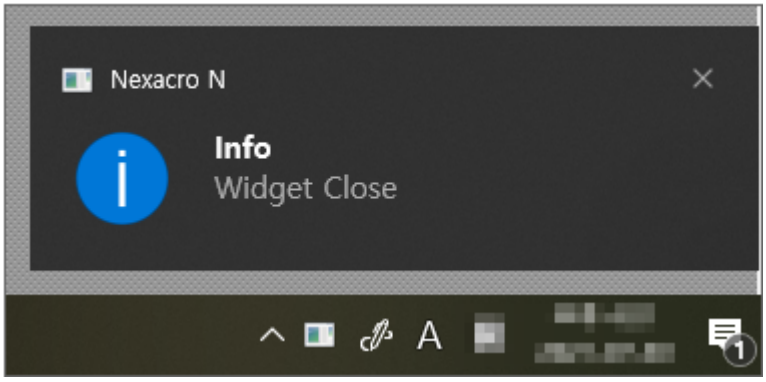
```
    }
    else if(e.id == "Exit")
    {
        nexacro.getApplication().exit();
    }
};
```



7 위젯 폼(WidgetMain1)에서 버튼 클릭 시 발생하는 이벤트 스크립트를 수정합니다.

트레이에서 메뉴 선택 시 매번 새로 위젯 화면을 생성하지 않고 visible 속성값만 수정해서 표시 여부만 변경합니다. 그리고 위젯이 닫힐 때 트레이 영역에 메시지를 표시하도록 코드를 추가했습니다.

```
this.Button00_onclick = function(obj:nexacro.Button,e:nexacro.ClickEventInfo)
{
    //this.close();
    this.parent.set_visible(false);
    nexacro.getApplication().trays["Tray0"].showBalloonTip("information", "Info", "Widget
Close");
};
```



8 툴바에서 실행옵션으로 NRE를 선택하고 Launch 아이콘을 클릭하거나 단축키 (Ctrl + F5)를 입력하고 윈도우 화면 오른쪽 아래에 트레이가 정상적으로 표시되고 오른쪽 마우스 버튼 클릭 시 메뉴가 표시되는지 확인합니다.

4.

윈도우 운영체제 NRE에서 단위 테스트 실행하기

WinAppDriver 연동 기능은 넥사크로에서 제공하는 UIAutomation 인터페이스를 사용해 NRE에서 실행하는 앱의 속성을 읽거나 쓰고 파라미터를 지정해 메소드를 실행하는 것을 지원합니다. 셀레니움 같은 UI 자동화 테스트 도구를 사용해 단위 테스트를 작성하고 테스트할 수 있습니다.

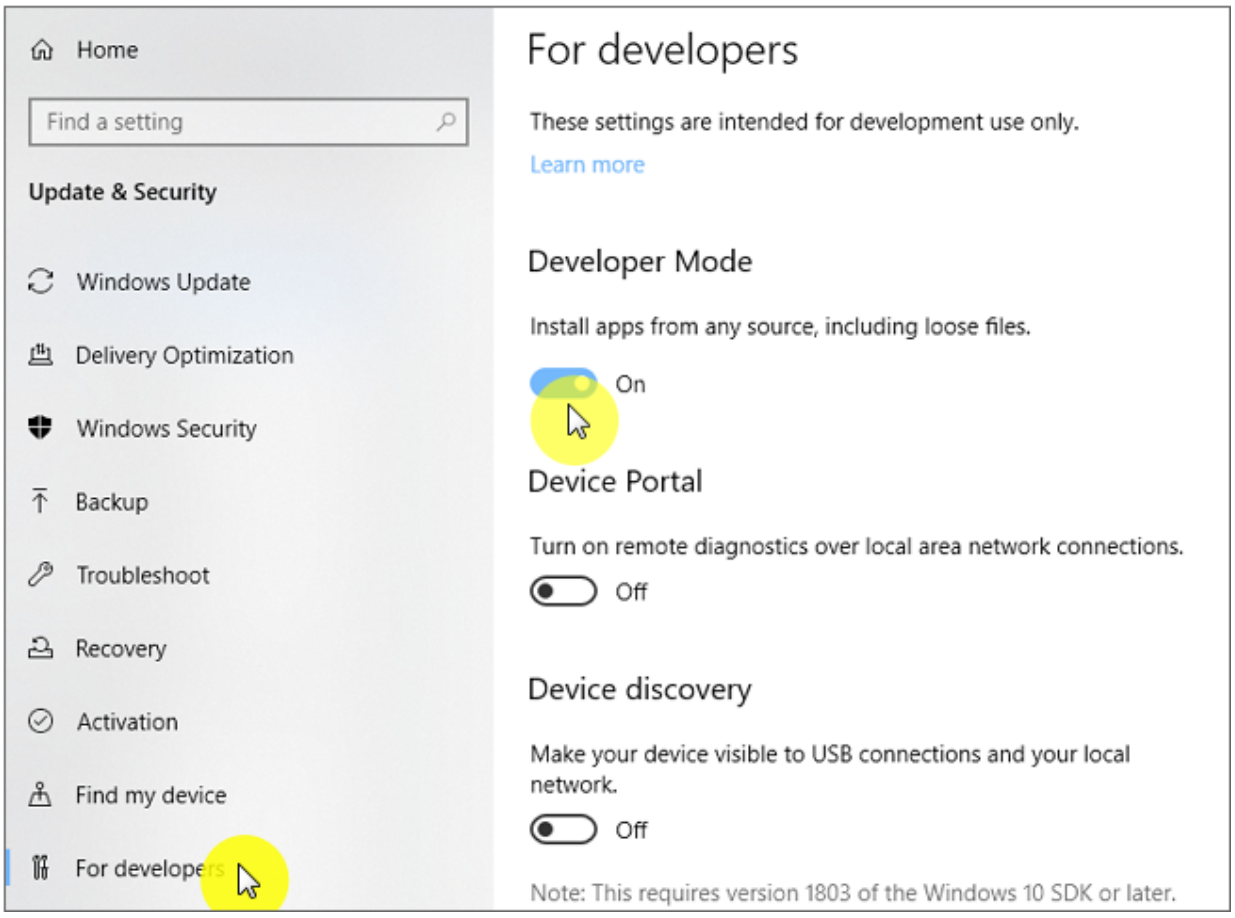
이번 장에서는 WinAppDriver를 설치하고 비주얼 스튜디오와 이클립스에서 기본적인 C#, Java 프로젝트를 생성해 넥사크로 앱에 연결하는 단계를 살펴봅니다.

> WinAppDriver는 윈도우 10 이상 환경에서 동작합니다.
> https://github.com/Microsoft/WinAppDriver#system-requirements

4.1 WinAppDriver 연동 환경 설정하기

4.1.1 WinAppDriver 설치하기

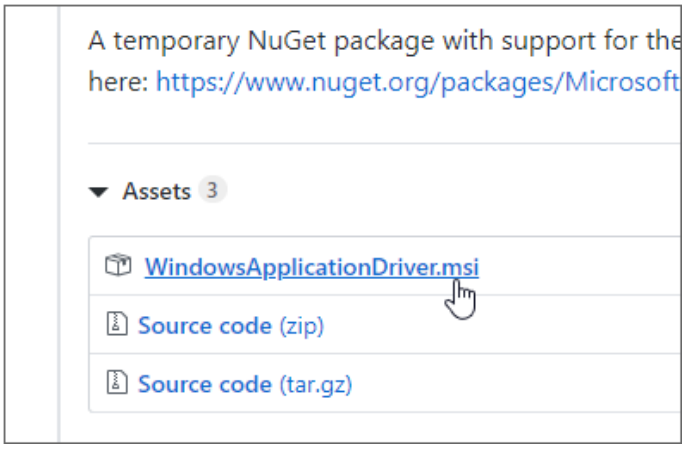
1. 윈도우 운영체제 [설정 > 개발자 기능 사용]에서 "개발자 모드"를 활성화합니다.



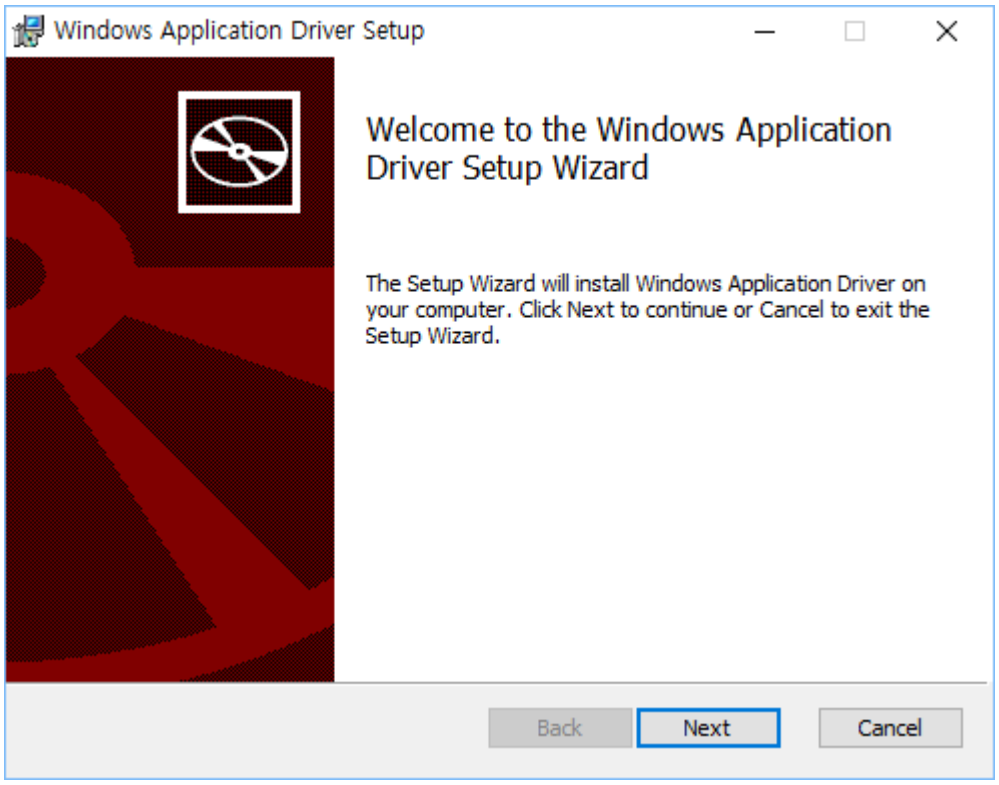
해당 모드를 최초 활성화하는 경우에는 운영체제 설정 작업에 약간의 시간이 걸릴 수 있습니다. 시간 여유를 가지고 모드를 활성화해주세요.

② WinAppDriver 설치 파일(WindowsApplicationDriver.msi)을 내려받습니다.

https://github.com/Microsoft/WinAppDriver
https://github.com/Microsoft/WinAppDriver/releases

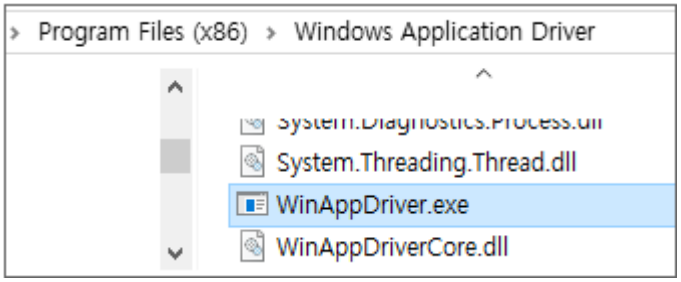


③ 내려받은 설치 파일(WindowsApplicationDriver.msi)을 실행해 WinAppDriver를 설치합니다.

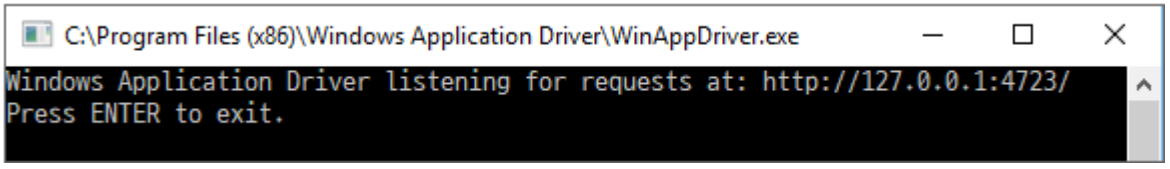


④ 설치 경로에서 WinAppDriver를 관리자 권한으로 실행합니다.

> 일반적인 설치 경로는 아래와 같습니다.
> C:₩Program Files (x86)₩Windows Application Driver₩



⑤ 정상적으로 실행이 되는지 확인합니다.



> ip, port 설정을 변경하고자 한다면 아래와 같이 실행합니다.
> > WinAppDriver.exe [ip] [port]

4.1.2 개발 도구 설치하기

C# 또는 Java 언어를 사용해 단위테스트 프로젝트를 만들 수 있습니다.

비주얼 스튜디오 설치하기

C# 언어를 사용하고 비주얼 스튜디오를 설치하지 않은 경우 무료로 제공되는 개발 도구를 설치할 수 있습니다.

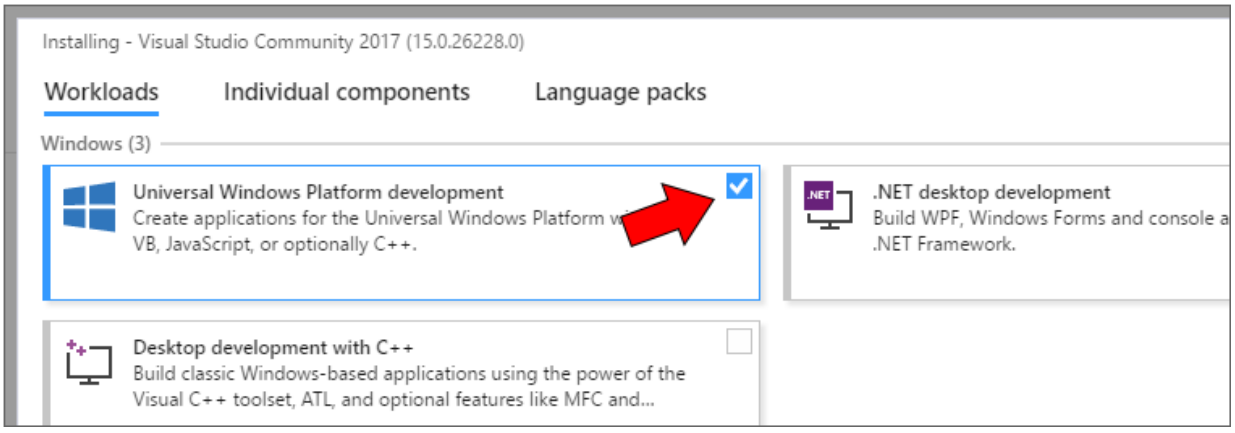
1. 비주얼 스튜디오 커뮤니티 설치 파일(vs_community__xxxx.xxxx.exe)을 내려받습니다.

> Visual Studio Community
> https://visualstudio.microsoft.com/vs/community/

2. 내려받은 설치 파일(vs_community__xxxx.xxxx.exe)을 실행해 비주얼 스튜디오를 설치합니다.

> 설치 시 필요한 패키지를 내려받는 과정에서 약간의 시간이 걸릴 수 있습니다. 시간 여유를 가지고 설치를 진행해주세요.

3. Workloads 항목에서 "Universal Windows Platform development"를 선택하고 설치를 진행합니다.



개발 환경은 설치 후에도 추가할 수 있습니다.

실행 시 마이크로소프트 계정 로그인하는 창이 표시되는데, 로그인 없이도 테스트는 해볼 수 있습니다.

이클립스 설치하기

Java 언어를 사용하고 이클립스를 설치하지 않은 경우 무료로 제공되는 개발 도구를 설치할 수 있습니다.

1. 이클립스 설치 파일(eclipse-inst-win64.exe)을 내려받습니다.

eclipse.org
https://www.eclipse.org/downloads/

2. 내려받은 설치 파일(eclipse-inst-win64.exe)을 실행해 이클립스를 설치합니다.

3. 개발 유형 선택 단계에서는 "Java Developers" 항목을 선택합니다.

4.2 단위테스트 만들고 실행하기

4.2.1 넥사크로 앱 만들고 배포하기

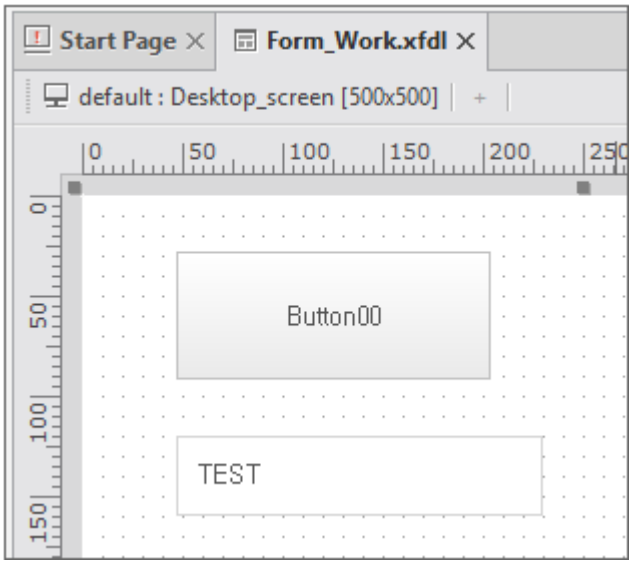
간단한 넥사크로 앱을 만들고 배포합니다.

> 이미 설치한 넥사크로 앱을 사용하려는 경우에는 이 단계는 지나갑니다.

1. 넥사크로 스튜디오를 실행합니다.

2. 프로젝트를 생성하고 간단한 폼 화면을 작성합니다.

   Button 컴포넌트와 Edit 컴포넌트만 화면에 배치합니다.



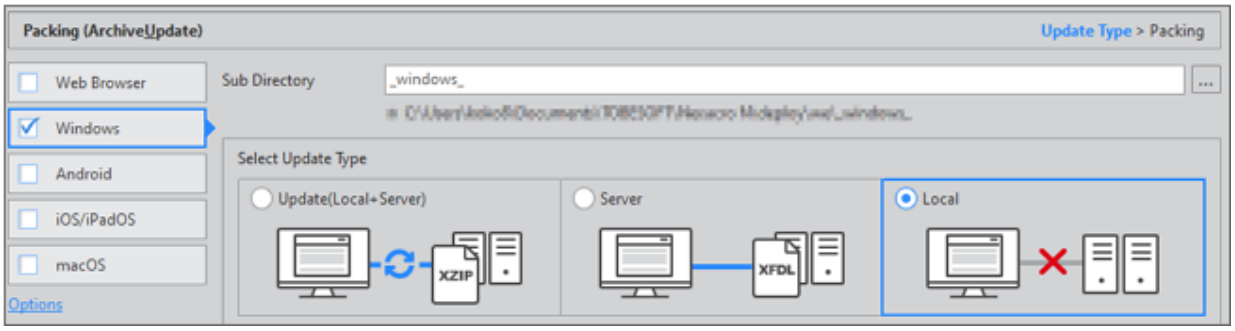
3. Button 컴포넌트의 onclick 이벤트 함수를 작성합니다.

   Button 컴포넌트 onclick 이벤트가 발생하면 Edit 컴포넌트에 value 속성값을 변경합니다.

```
this.Button00_onclick = function(obj:nexacro.Button,e:nexacro.ClickEventInfo)
{
    this.Edit00.set_value("BUTTON CLICK");
};
```

4. 메뉴 [Deploy > Packing (Archive&Update)] 항목을 선택하고 배포할 아카이브 파일을 생성합니다.

서버 연결 없이 실행할 수 있도록 Update Type은 "Local"로 지정합니다.



5 [Packing] 버튼을 클릭해 아카이브 파일을 생성되면 [Build App] 버튼을 클릭합니다.

6 Build App 창에서 [Build] 버튼을 클릭해 설치 파일을 생성합니다.

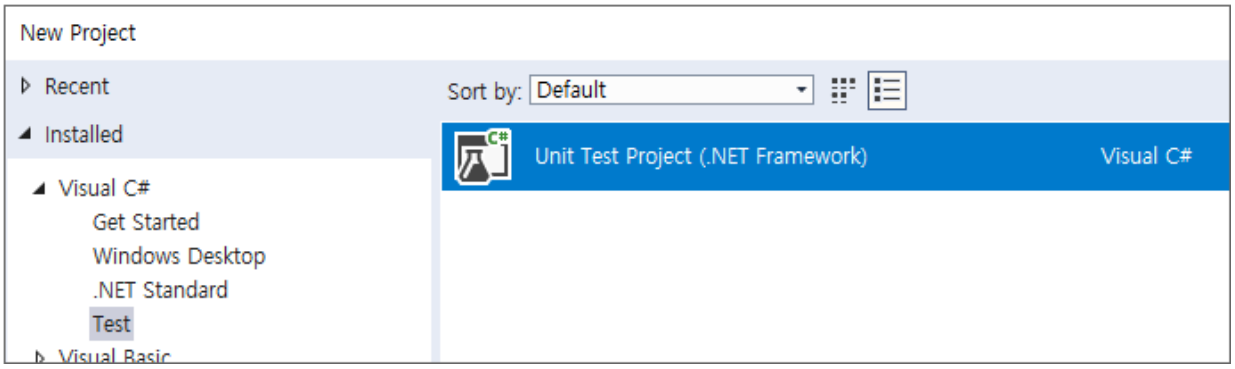
7 생성한 설치 파일을 실행해 넥사크로 앱을 설치합니다.

4.2.2 단위테스트 프로젝트 만들기

비주얼 스튜디오에서 단위테스트 프로젝트 만들기

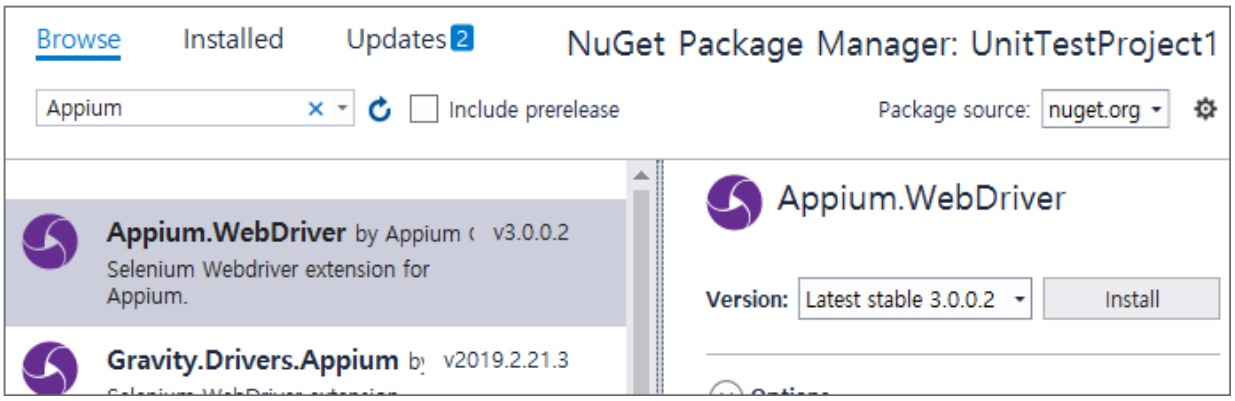
1 비주얼 스튜디오를 실행합니다.

2 새로운 프로젝트 만들기에서 "Unit Test Project"를 선택하고 새로운 프로젝트를 생성합니다.



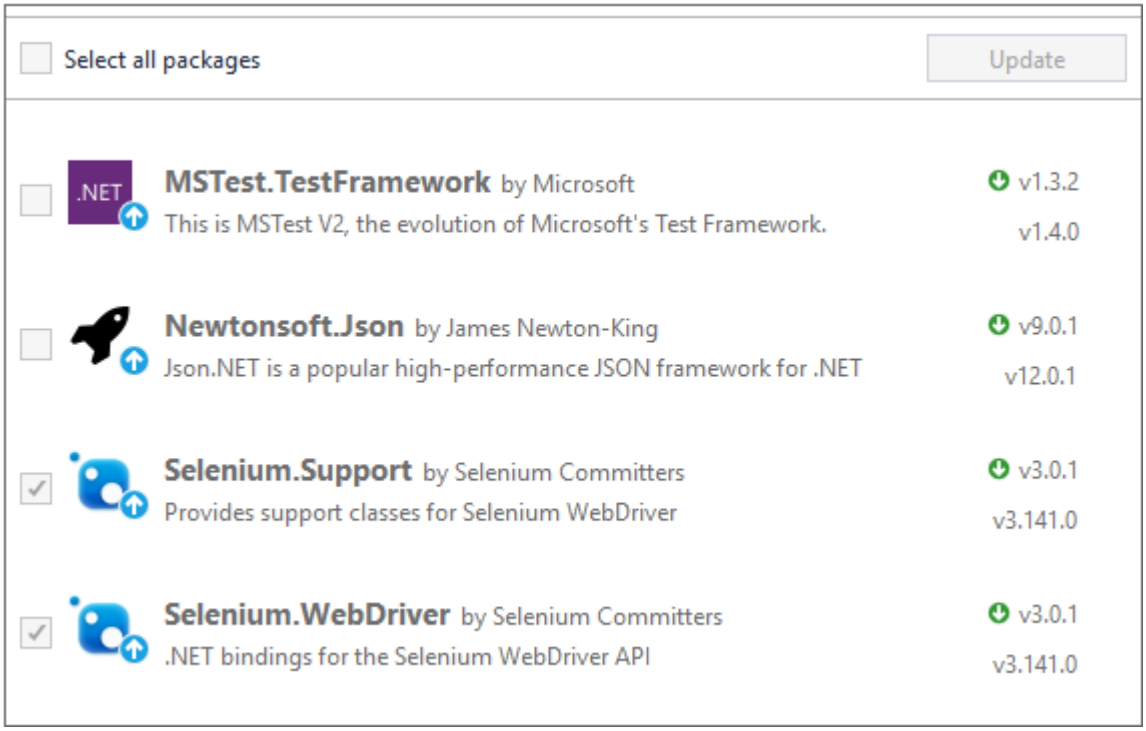
3 Appium WebDriver 라이브러리를 설치합니다.

메뉴 [Project > Manage NuGet Package]를 선택하고 검색창에서 "Appium"을 입력해서 검색 결과에서 Appium WebDriver 라이브러리를 확인합니다. [Install] 버튼을 클릭하면 라이브러리를 설치합니다.

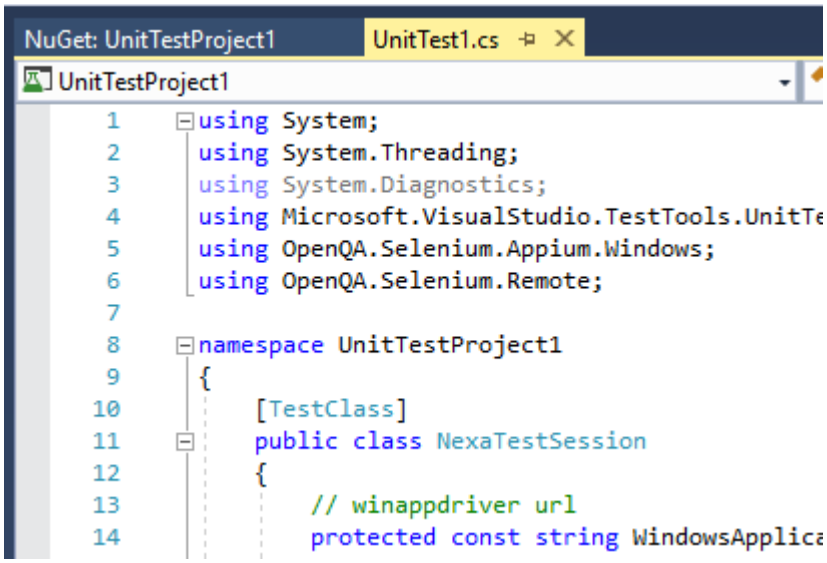


4 Updates 탭에서 Selenium 관련 패키지를 확인합니다.

> 최신 버전이 설치된 경우에는 코드가 동작하지 않을 수 있습니다. 최신 버전인 경우에는 3.0.1 버전으로 마이너 업데이트해주세요.
> https://github.com/Microsoft/WinAppDriver/issues/497



5 생성된 C# 소스코드파일을 아래와 같이 수정합니다.



아래 코드에서 넥사크로 앱 관련 설정을 수정합니다. 앱이 설치된 경로, nexacro.exe 실행 파일 경로, nexacro.exe 실행 옵션을 실제 설치 경로에 맞게 수정합니다.

	value
WindowsApplicationDriverUrl	http://127.0.0.1:4723/
NexacroAppDr	C:\\Program Files\\TOBESOFT\\Nexacro N\\21\\app0201
NexacroAppId	C:\\Program Files\\TOBESOFT\\Nexacro N\\21\\app0201\\nexacro.exe
NexacroAppAg	-k 'UnitTest' -s 'C:\\Program Files\\TOBESOFT\\Nexacro N\\21\\app0201\\app0201\\start.json'

```
using System;
using System.Threading;
using System.Diagnostics;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using OpenQA.Selenium.Appium.Windows;
using OpenQA.Selenium.Remote;

namespace UnitTestProject1
{
    [TestClass]
    public class NexaTestSession
    {
        // winappdriver url
        protected const string WindowsApplicationDriverUrl = "http://127.0.0.1:4723/";
        // app path
        private const string NexacroAppDr = "C:\\Program Files\\TOBESOFT\\Nexacro N\\21\\
app0201";
```

```
        // nexacro.exe path
        private const string NexacroAppId = "C:\\Program Files\\TOBESOFT\\Nexacro N\\21\\
app0201\\nexacro.exe";
        // nexacro.exe argument
        private const string NexacroAppAg = "-k 'UnitTest' -s 'C:\\Program Files\\TOBESOFT
\\Nexacro N\\21\\app0201\\app0201\\start.json'";
        // search root
        private const string FindNexacro = "Root";
        // OS/Device
        private const string TestOs = "Windows";
        private const string TestDevice = "WindowsPC";

        // Session Object : WindowsDriver
        protected static WindowsDriver<WindowsElement> session;

        // WinAppDriver Session Create
        [ClassInitialize]
        public static void Setup(TestContext context)
        {
            // Launch a new instance of nexacro application or find nexacro application
        }

        // WinAppDriver Session Delete
        public static void TearDown()
        {
            // Close the application and delete the session
            if (session != null)
            {
                // Close Session  -  Close Application
                session.Close();
                // Quit Session
                session.Quit();
                // Delete Session
                session = null;
            }
        }

        [TestMethod]
        public void Test_Session()
        {
            Debug.WriteLine("session:" + session);
```

```
            Debug.WriteLine("session.SessionId:" + session.SessionId);
        }
    }
}
```

6 넥사크로 앱을 실행하고 세션을 생성하는 코드를 추가합니다.

위의 코드에서 Setup 메소드를 아래와 같이 수정합니다.

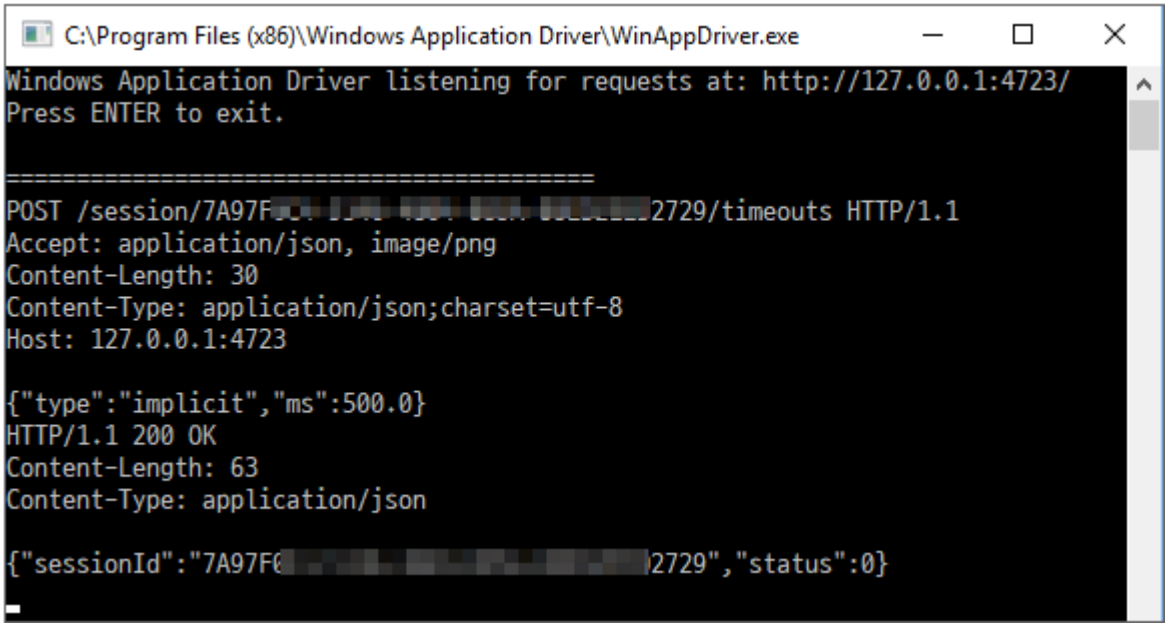
```
// WinAppDriver Session Create
[ClassInitialize]
public static void Setup(TestContext context)
{
    // Launch a new instance of nexacro application or find nexacro application
    if (session == null)
    {
        // Create a new session to launch nexacro application
        DesiredCapabilities appCapabilities = new DesiredCapabilities();
        TimeSpan time = new TimeSpan(0, 0, 10);

        // Launch Nexacro
        appCapabilities.SetCapability("app", NexacroAppId);
        appCapabilities.SetCapability("appArguments", NexacroAppAg);
        appCapabilities.SetCapability("appWorkingDir", NexacroAppDr);
        appCapabilities.SetCapability("platformName", TestOs);
        appCapabilities.SetCapability("deviceName", TestDevice);

        // Request Remote WinAppDriver Service
    session = new WindowsDriver<WindowsElement>(new Uri(WindowsApplicationDriverUrl),
appCapabilities, time);

        // Set implicit timeout to 1.5 seconds to make element search to retry every 500
ms for at most three times
        session.Manage().Timeouts().ImplicitlyWait(TimeSpan.FromSeconds(0.5));
        Thread.Sleep(TimeSpan.FromSeconds(1.0));
    }
}
```

TimeSpan 설정값에 따라 타임아웃 오류가 발생하는 경우가 있습니다. 그런 경우에는 아래와 같이 설정값을 변경해서 테스트를 진행합니다.

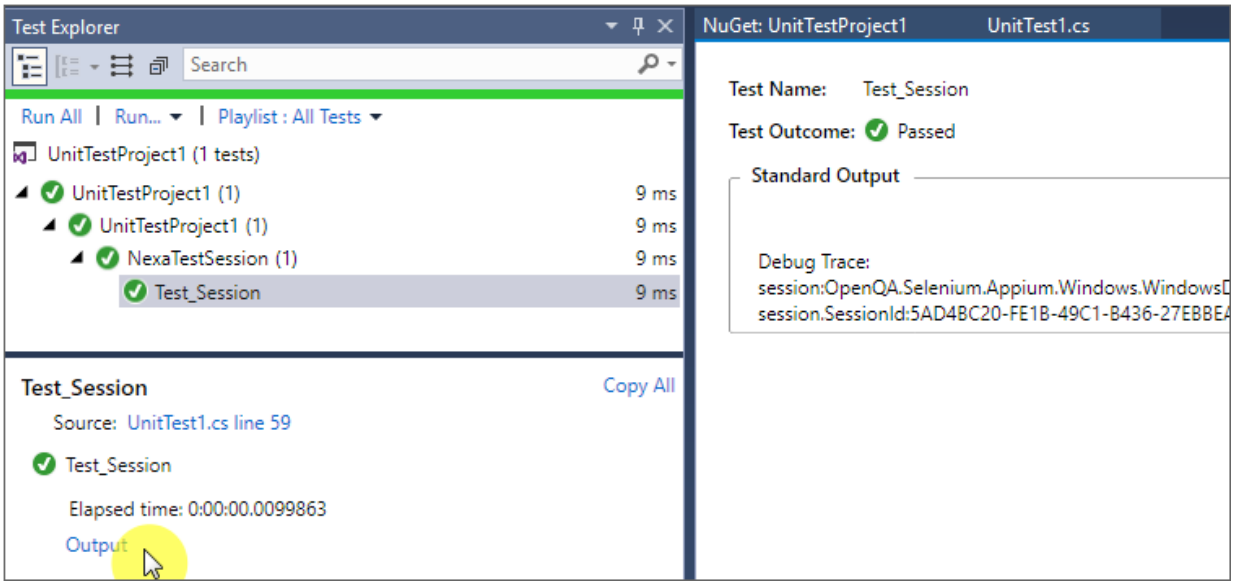
```
        TimeSpan time = new TimeSpan(0, 0, 30);
```

7 메뉴 [Test > Run > All Tests] 항목을 선택하면 테스트를 진행합니다. 설치한 앱이 실행되는 것을 확인할 수 있습니다.

WinAppDriver을 실행한 콘솔 화면에서 접속 정보와 세션이 생성된 것을 확인할 수 있습니다.



8 비주얼 스튜디오 테스트 탐색기에서는 성공한 테스트 정보를 확인할 수 있습니다.



이클립스에서 단위테스트 프로젝트 만들기

1. 이클립스를 실행합니다.

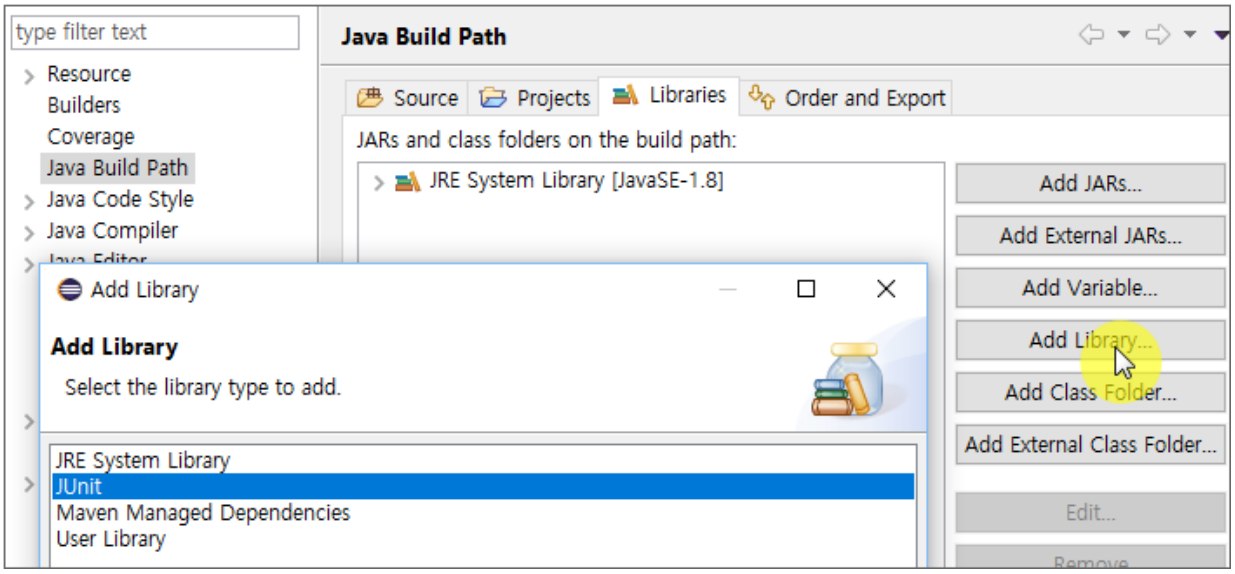
2. 메뉴 [File > New > Java Project]를 선택하고 새로운 프로젝트를 생성합니다.

3. 필요한 라이브러리를 내려받습니다. 압축된 파일(.zip)은 압축을 해제합니다.

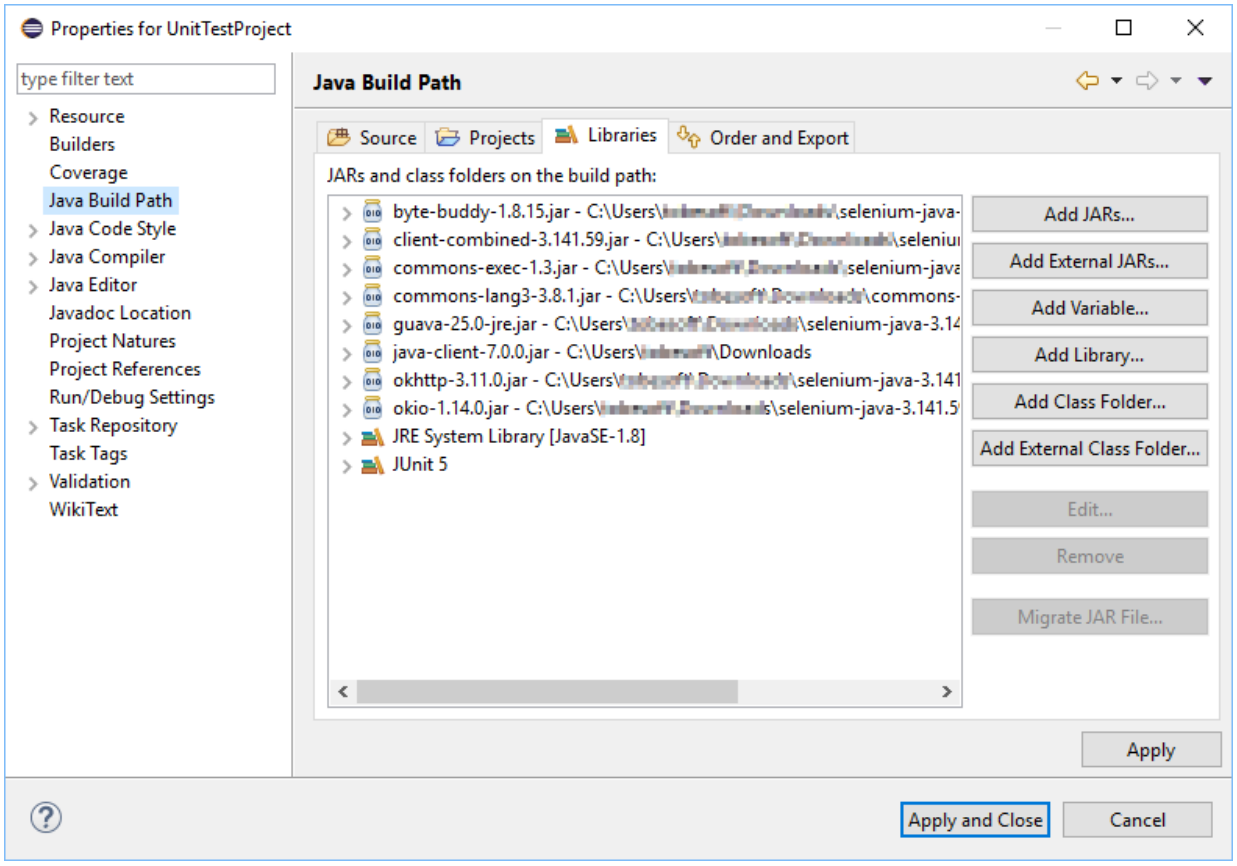
라이브러리	다운로드 경로
Appium Client Libraries	http://appium.io/downloads.html java-client-7.0.0.jar
Selenium Client	https://www.seleniumhq.org/download/ selenium-java-3.141.59.zip
Apache Commons Lang	http://commons.apache.org/proper/commons-lang/download_lang.cgi commons-lang3-3.8.1-bin.zip

> Apache Commons Lang 라이브러리는 실행 중 아래와 같은 오류 발생 시 내려받아 설치합니다. 개발 환경에 따라 필요없을 수도 있습니다.
> `java.lang.NoClassDefFoundError: org/apache/commons/lang3/StringUtils`

4. JUnit 라이브러리를 추가합니다. 메뉴 [Project > Properties > Java Build Path > Libraries] 탭을 선택하고 [Add Library] 버튼을 클릭합니다.



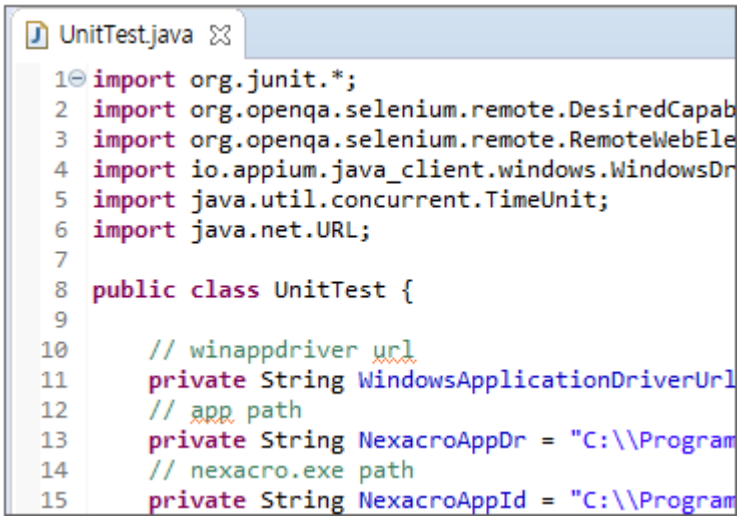
5. Appium, Selenium 라이브러리를 추가합니다. [Add External JARs] 버튼을 클릭하고 내려받은 라이브러리 파일에서 JAR 파일을 추가합니다.



6 [Apply and Close] 버튼을 클릭해 변경한 내용을 적용합니다.

7 메뉴 [File > New > Class]를 선택하고 새로운 클래스 파일을 생성합니다. Name 항목만 입력하고 다른 옵션은 수정하지 않습니다.

8 생성된 Java 소스코드파일을 아래와 같이 수정합니다.



```
UnitTest.java
1  import org.junit.*;
2  import org.openqa.selenium.remote.DesiredCapab
3  import org.openqa.selenium.remote.RemoteWebEle
4  import io.appium.java_client.windows.WindowsDr
5  import java.util.concurrent.TimeUnit;
6  import java.net.URL;
7
8  public class UnitTest {
9
10     // winappdriver url
11     private String WindowsApplicationDriverUrl
12     // app path
13     private String NexacroAppDr = "C:\\Program
14     // nexacro.exe path
15     private String NexacroAppId = "C:\\Program
```

아래 코드에서 넥사크로 앱 관련 설정을 수정합니다. 앱이 설치된 경로, nexacro.exe 실행 파일 경로, nexacro.

exe 실행 옵션을 실제 설치 경로에 맞게 수정합니다.

	value
WindowsApplicationDriverUrl	http://127.0.0.1:4723/
NexacroAppDr	C:\\Program Files (x86)\\TOBESOFT\\Nexacro N\\21\\app0201
NexacroAppId	C:\\Program Files (x86)\\TOBESOFT\\Nexacro N\\21\\app0201\\nexacro.exe
NexacroAppAg	-k 'UnitTest' -s 'C:\\Program Files (x86)\\TOBESOFT\\Nexacro N\\21\\app0201\\app0201\\start.json'

```
import org.junit.*;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebElement;
import io.appium.java_client.windows.WindowsDriver;
import java.util.concurrent.TimeUnit;
import java.net.URL;

public class UnitTest {

    // winappdriver url
    private String WindowsApplicationDriverUrl = "http://127.0.0.1:4723/";
    // app path
    private String NexacroAppDr = "C:\\Program Files (x86)\\TOBESOFT\\Nexacro N\\21\\app
0201";
    // nexacro.exe path
    private String NexacroAppId = "C:\\Program Files (x86)\\TOBESOFT\\Nexacro N\\21\\app
0201\\nexacro.exe";
    // nexacro.exe argument
    private String NexacroAppAg = "-k 'UnitTest' -s 'C:\\Program Files (x86)\\TOBESOFT\\
Nexacro N\\21\\app0201\\app0201\\start.json'";
    // search root
    // private String FindNexacro = "Root";

    // OS/Device
    private String TestOs = "Windows";
    private String TestDevice = "WindowsPC";

    // Session Object : WindowsDriver
    private static WindowsDriver<RemoteWebElement> session = null;
```

```
    @Before
    public void setUp() throws Exception {
        // Launch a new instance of nexacro application or find nexacro application
    }

    @After
    public void tearDown() throws Exception {
        if (session != null) {
            // NexacroSession.quit();
        }
    }

    @Test
    public void Test_01_Session()
    {
        System.out.println("session : " + session);
        System.out.println("session.SessionId : " + session.getSessionId());
    }
}
```

⑨ 넥사크로 앱을 실행하고 세션을 생성하는 코드를 추가합니다.

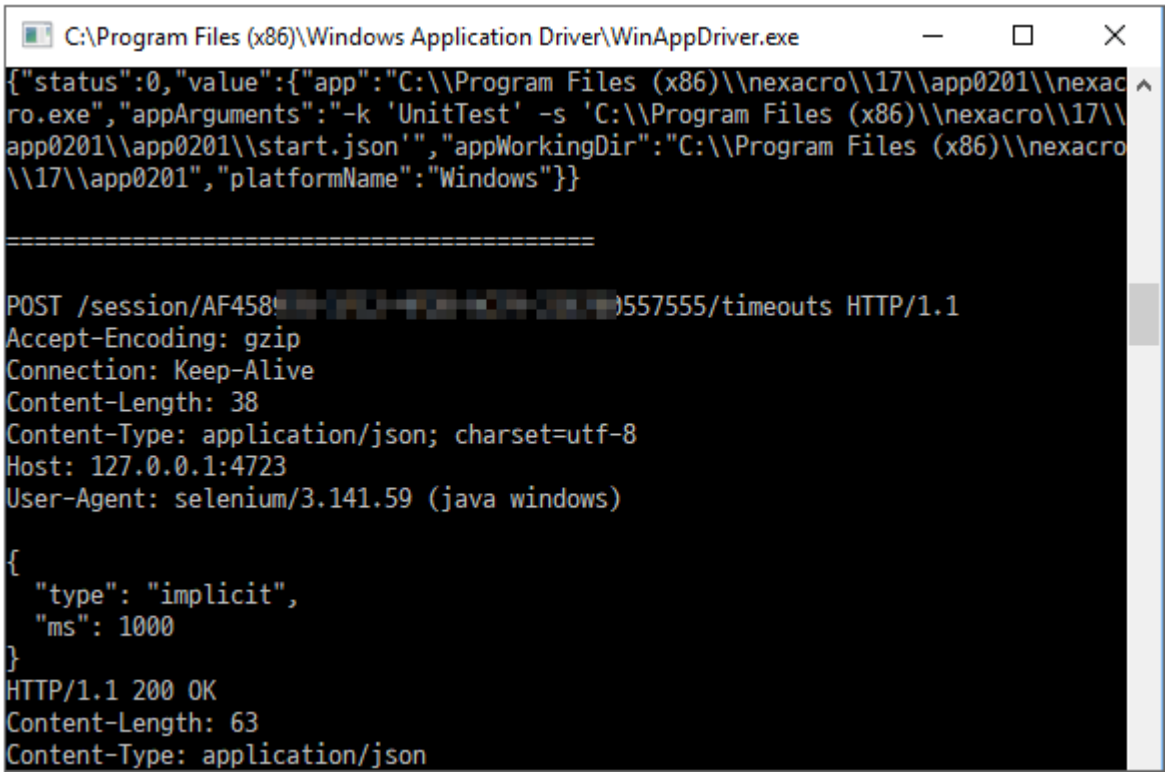
위의 코드에서 Setup 메소드를 아래와 같이 수정합니다.

```
@Before
public void setUp() throws Exception {
    // Launch a new instance of nexacro application or find nexacro application
    try {
        // Create a new session to launch nexacro application
        DesiredCapabilities appCapabilities = new DesiredCapabilities();

        appCapabilities.setCapability("app", NexacroAppId);
        appCapabilities.setCapability("appArguments", NexacroAppAg);
        appCapabilities.setCapability("appWorkingDir", NexacroAppDr);
        appCapabilities.setCapability("platformName", TestOs);
        appCapabilities.setCapability("deviceName", TestDevice);

        // Request Remote WinAppDriver Service
        session = new WindowsDriver<RemoteWebElement>(new URL(WindowsApplicationDriverUrl
), appCapabilities);
        session.manage().timeouts().implicitlyWait(1, TimeUnit.SECONDS);
```

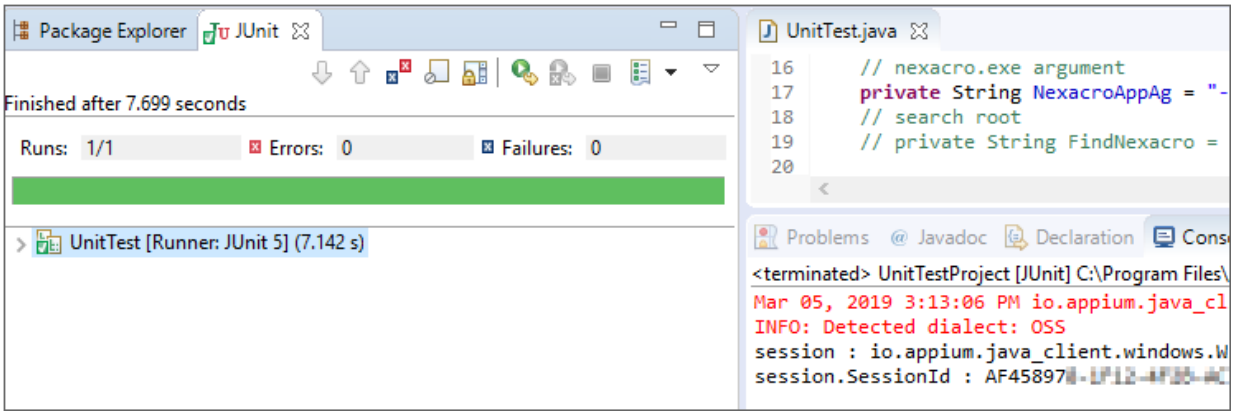
```
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
    }
}
```

10 메뉴 [Run > Run As > JUnit Test] 항목을 선택하면 테스트를 진행합니다. 설치한 앱이 실행되는 것을 확인할 수 있습니다.

WinAppDriver을 실행한 콘솔 화면에서 접속 정보와 세션이 생성된 것을 확인할 수 있습니다.



11 JUnit 패널에서 성공한 테스트 정보를 확인할 수 있습니다.



4.2.3 단위테스트 실행하기

1 단위테스트를 실행할 코드를 아래와 같이 추가합니다.

현재 세션에서 Button 컴포넌트를 찾고 마우스 클릭 명령을 처리합니다.

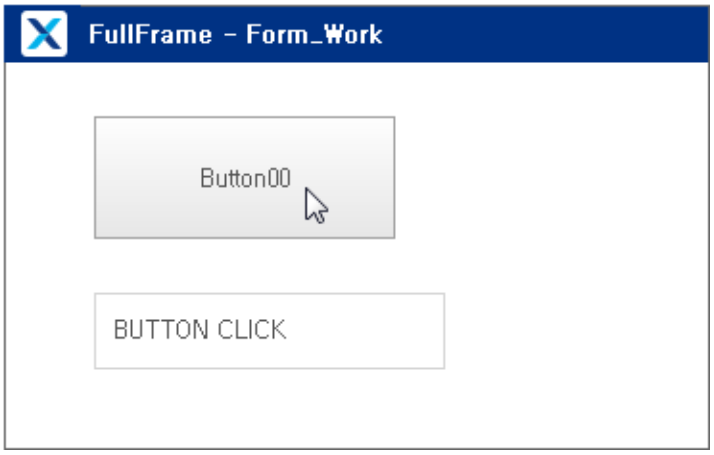
C#

```
[TestMethod]
public void Test_Button()
{
    WindowsElement element = session.FindElementByAccessibilityId("mainframe.WorkFrame.
form.Button00");
    Assert.IsNotNull(element);
    Thread.Sleep(TimeSpan.FromSeconds(0.1));
    element.Click();
}
```

Java

```
@Test
public void Test_Button() throws Exception {
    WebElement element = session.findElementByAccessibilityId("mainframe.WorkFrame.form.
Button00");
    Assert.assertNotNull(element);
    Thread.sleep(1000);
    element.click();
}
```

2 테스트를 실행하면 설치한 앱이 실행되고 Button 컴포넌트 위에 마우스 커서가 위치하고 클릭 명령을 실행하는 것을 확인할 수 있습니다.



4.3 AccessibilityId 속성값 확인하기

컴포넌트나 컨트롤에 접근하기 위해서 FindElementByAccessibilityId 메소드를 사용하는데 이때 AccessibilityId 속성값을 확인해야 합니다.

4.3.1 컴포넌트 속성값 확인하기

Form에 배치된 컴포넌트 자체에 접근하는 방식은 아래와 같습니다.

```
[MainFrame id 속성값].[ChildFrame id 속성값].form.[Component id 속성값]
```

id 속성값이 "Button00"인 Button 컴포넌트의 엘리먼트를 얻기 위해서는 아래와 같이 코드를 작성합니다.

C#

```
WindowsElement element = session.FindElementByAccessibilityId("mainframe.ChildFrame00.form.
Button00");
```

Div 컴포넌트 안에 배치한 컴포넌트는 아래와 같이 접근할 수 있습니다.

C#

```
WindowsElement element = session.FindElementByAccessibilityId("mainframe.ChildFrame00.form.Div
00.form.CheckBox00");
```

4.3.2 컨트롤, 아이템 속성값 확인하기

예를 들어 Div 컴포넌트 안에 배치된 Combo 컴포넌트의 dropbutton은 아래와 같이 엘리먼트를 얻을 수 있습니다.

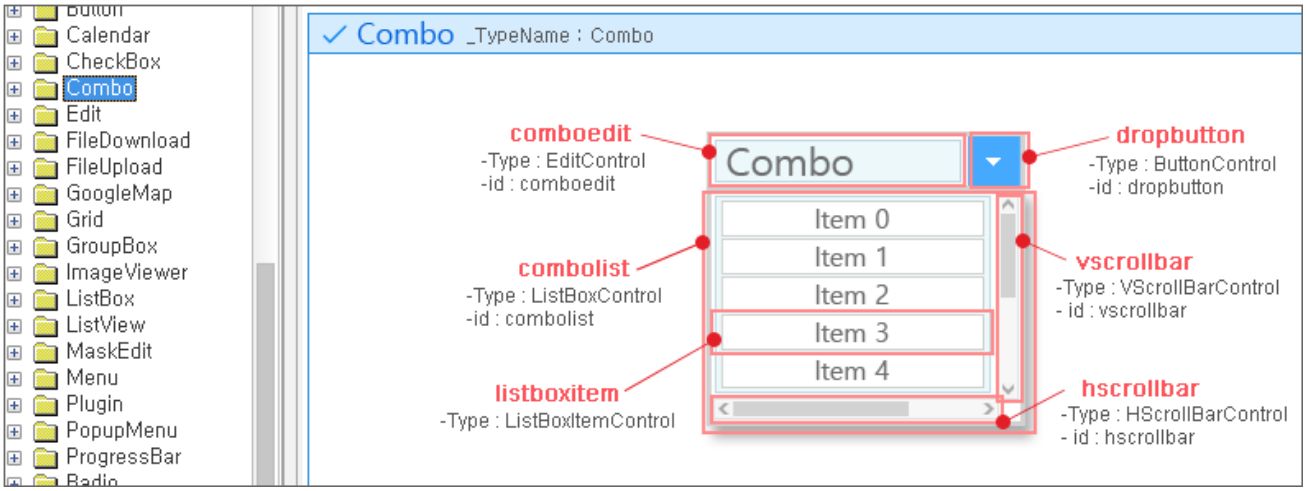
C#

```
WindowsElement element = session.FindElementByAccessibilityId("mainframe.ChildFrame00.form.Div
00.form.Combo00.dropbutton");
```

단위 테스트를 진행하기 위해서는 Combo 컴포넌트의 드롭다운목록을 펼치고 아이템을 선택해야 합니다. 드롭다운목록을 펼치기 위해서는 Combo 컴포넌트 내 ButtonControl에 접근해야 하고 해당 컨트롤의 속성값을 확인해야 합니다.

컨트롤 속성값은 넥사크로 스튜디오에서 Help(단축키 F1)을 실행하면 컴포넌트 개요에서 확인할 수 있습니다. Com

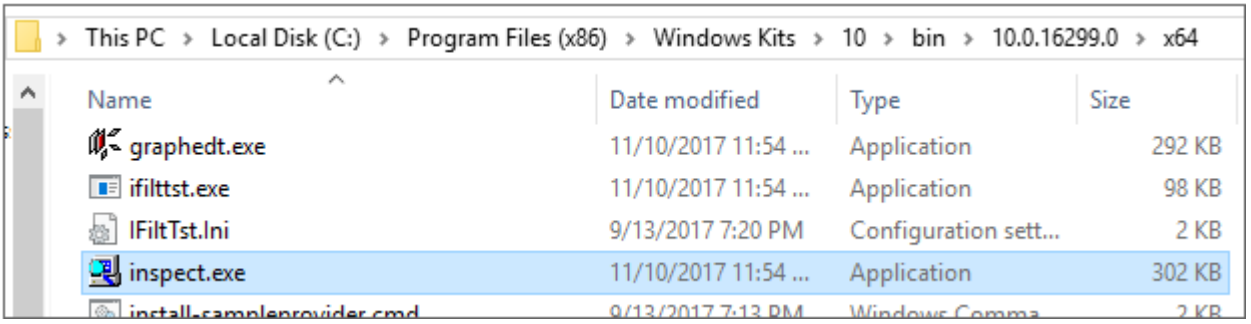
bo 컴포넌트에서 드롭다운목록을 펼치기 위해 클릭하는 버튼의 id 속성값이 "dropbutton"이라는 것을 확인할 수 있습니다.



4.3.3 Inspect로 속성값 확인하기

> 도움말에서 노출되지 않는 속성값(하위 컨트롤)은 제품 버전에 따라 변경될 수 있습니다.
> 해당 속성값 변경 시 작성된 테스트가 동작하지 않을 수 있습니다.

하지만 드롭다운목록을 펼친 후에 Item 3에 접근하려면 어떻게 해야 하는지는 알려주지 않습니다. 이런 경우에는 "Inspect" 라는 도구를 사용합니다. "Inspect"는 윈도우 SDK가 설치되어 있다면 설치 경로에서 찾을 수 있습니다.



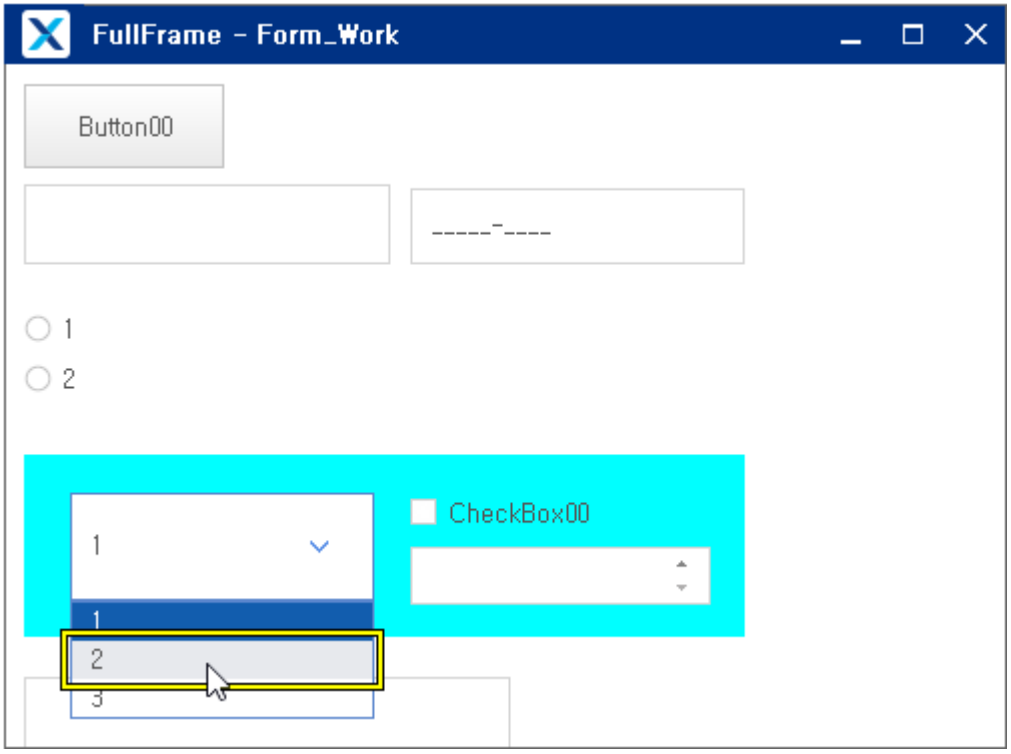
> Inspect
> https://docs.microsoft.com/en-us/windows/desktop/winauto/inspect-objects

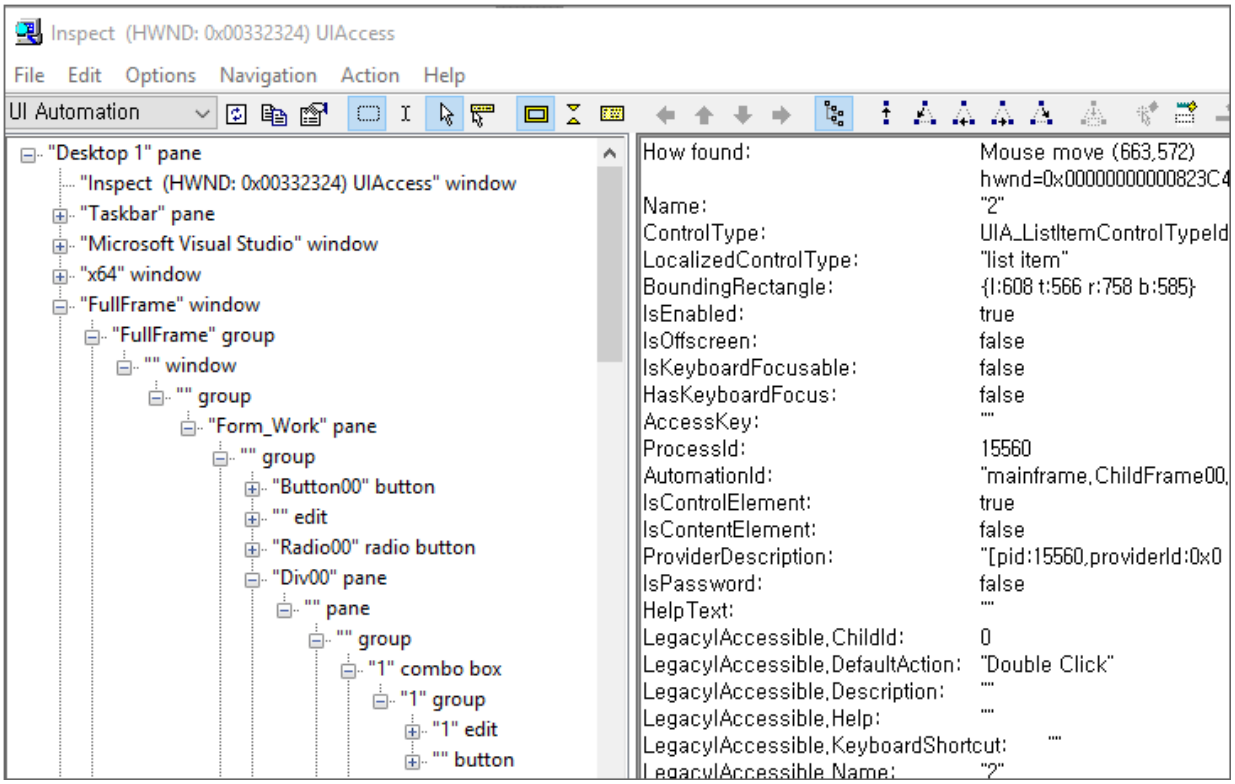
1. 윈도우 SDK 설치 경로에서 inspect.exe 파일을 실행합니다.

2. 넥사크로 앱을 실행하고 마우스 커서를 원하는 위치로 가져갑니다.

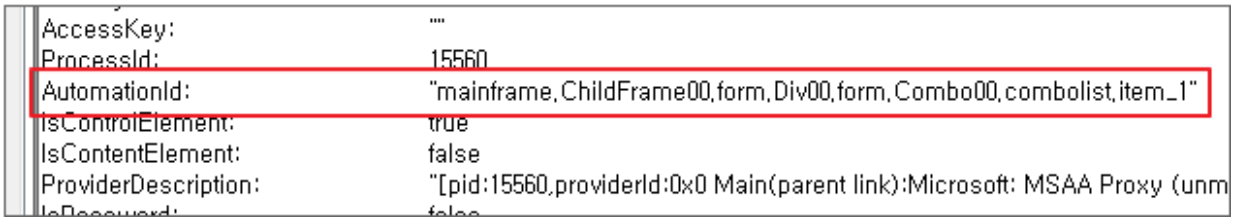
③ Combo 컴포넌트의 dropdown 버튼을 클릭해서 드롭다운 목록을 펼칩니다.

④ 원하는 아이템 영역에 마우스 커서를 가져갑니다. 아래 그림처럼 Inspect가 인식한 대상 영역에 테두리선이 표시되고 Inspect 창에 관련 정보가 표시됩니다.





여러 정보가 보여지는데 그 중에서 AccessibilityId 속성값으로 사용할 항목은 "AutomationId"입니다.

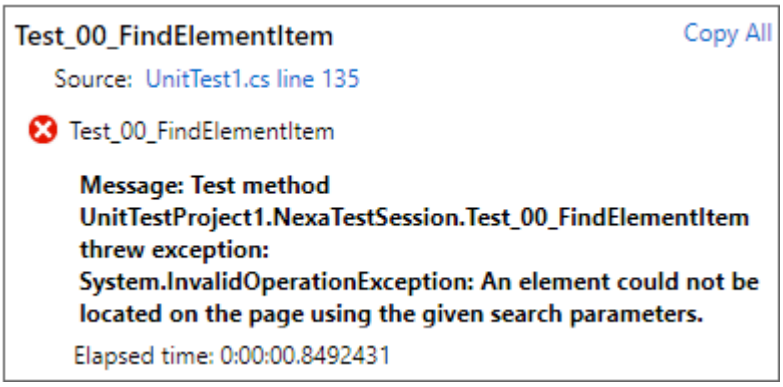


드롭다운목록에서 2번째 아이템을 선택할 때는 아래와 같이 엘리먼트를 얻을 수 있습니다.

C#

```
WindowsElement element = session.FindElementByAccessibilityId("mainframe.ChildFrame00.form.Div00.form.Combo00.combolist.item_1");
```

아이템에 해당하는 엘리먼트는 드롭다운목록이 펼쳐진 후에 접근할 수 있습니다. 목록을 펼치기 전에 해당 아이템에 접근하려 한다면 테스트 실패로 처리됩니다.



테스트 코드에서 아래와 같이 먼저 dropbutton 엘리먼트를 찾아서 마우스 클릭 이벤트를 처리하고 드롭다운목록이 펼쳐지면 그때 원하는 아이템에 접근합니다.

C#

```
[TestMethod]
public void Test_00_FindElementItem()
{
    WindowsElement element = session.FindElementByAccessibilityId("mainframe.ChildFrame00.form.Div00.form.Combo00.dropbutton");
    Assert.IsNotNull(element);
    element.Click();
    Thread.Sleep(TimeSpan.FromSeconds(0.1));
    WindowsElement elementItem = session.FindElementByAccessibilityId("mainframe.ChildFrame00.form.Div00.form.Combo00.combolist.item_1");
    Assert.IsNotNull(elementItem);
```

```
    Debug.WriteLine("elementItem.Text:" + elementItem.Text);
}
```

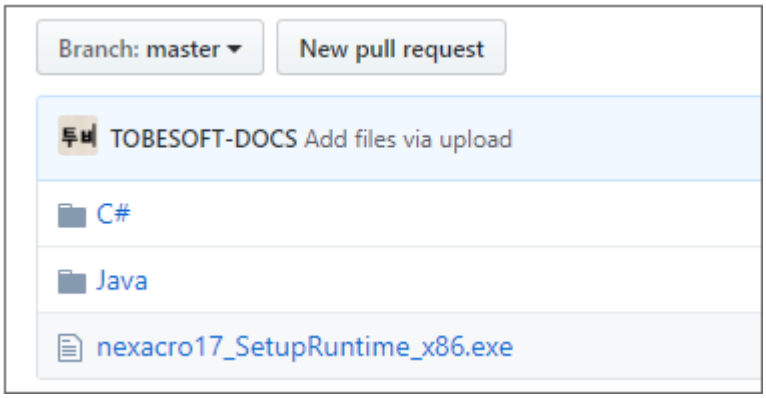
4.4 기능별 데모 단위 테스트 실행하기

기능별 데모를 실행하기 위한 넥사크로 앱 설치파일과 Java, C# 샘플 파일을 내려받을 수 있습니다.

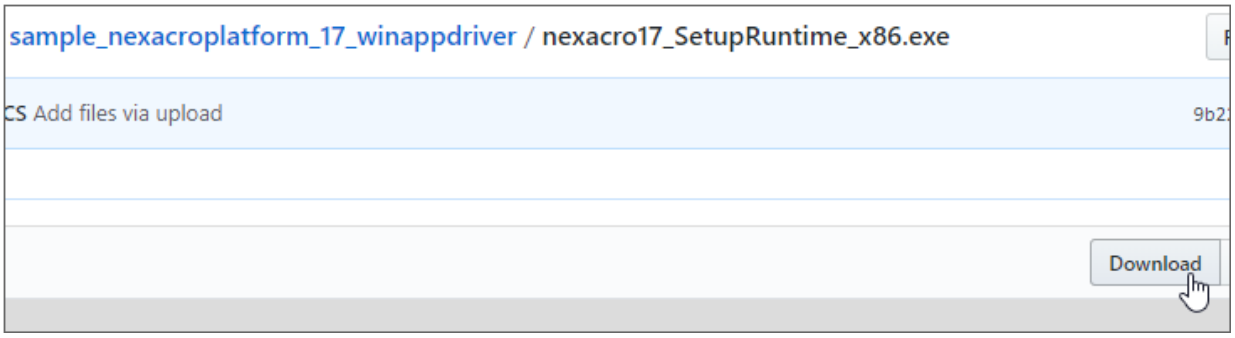
1. 웹브라우저에서 아래 주소로 접속합니다.

   https://github.com/TOBESOFT-DOCS/sample_nexacroplatform_17_winappdriver

   2개의 폴더와 하나의 설치파일을 확인할 수 있습니다.



2. nexacro17_SetupRuntime_x86.exe 파일을 내려받습니다. 파일명을 마우스로 클릭하면 [Download] 버튼을 확인할 수 있습니다.

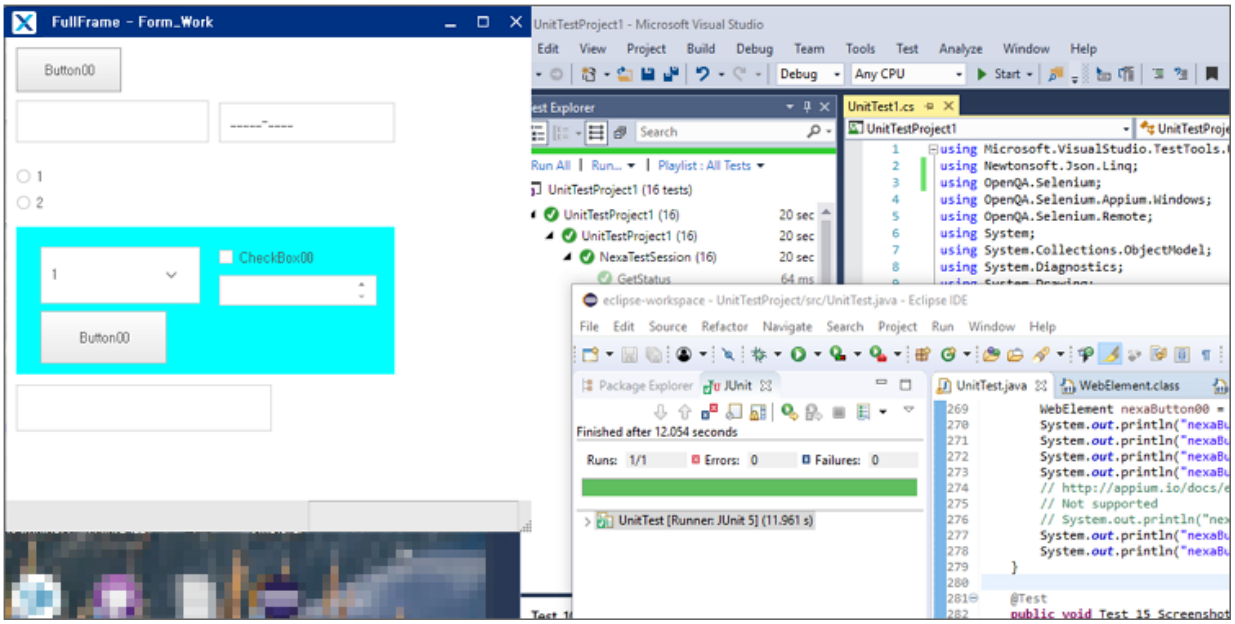


3. 내려받은 설치파일을 실행해 앱을 설치합니다. 아래 경로에 앱이 설치됩니다.

```
C:\Program Files (x86)\nexacro\17\app0201
```

④ 단위테스트 프로젝트 만들기에서 만든 프로젝트에 C#, Java 샘플을 붙여넣습니다.

⑤ 단위테스트를 실행합니다. 정상적으로 단위테스트가 처리되는지 확인합니다.

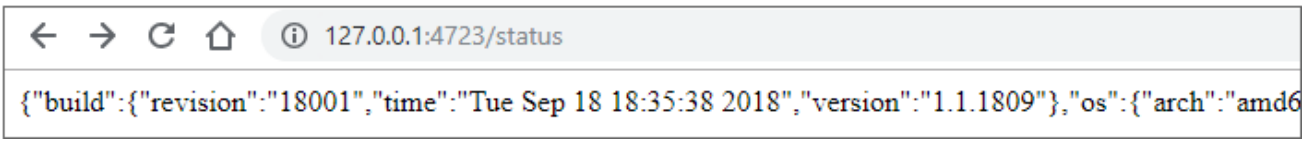


4.5 지원 인터페이스

> SeleniumRC, Appium 인터페이스를 모두 지원하지는 않습니다.
> 넥사크로 플랫폼에서 지원할 수 없거나 WinAppDriver에서 지원하지 않는 기능은 제외했습니다.

> 아래 항목은 클라이언트에서 별도 인터페이스를 지원하지 않고 웹브라우저에서 직접 호출하거나 Http WebResponse 등을 사용해 호출한 경우 사용할 수 있습니다.

HTTP	WebDriver	WinAppDriver
GET	/status	/status
GET		/sessions
GET	/session/{session id}/window	/session/:sessionId/window
GET	/session/{session id}/window/handles	/session/:sessionId/window/handles
POST	/session/{session id}/window/maximize	/session/:sessionId/window/maximize



4.5.1 Session

세션 생성하기

winappdriver 정보로 세션을 생성하고 앱을 실행합니다.

C#	`session = new WindowsDriver(new Uri(WinAppDriverUrl), appCapabilities);`
Java	`session = new WindowsDriver(new URL(WindowsApplicationDriverUrl), appCapabilities);`
통신	`POST /session { …capabilities json data… }`

세션 종료하기

세션을 종료하고 해당 앱을 종료합니다.

C#	`session.Quit();`
Java	`session.quit();`
통신	`DELETE /session/:session_id`

WinAppDriver에서 제공하는 인터페이스 중 일부는 넥사크로에서 지원하지 않습니다.

4.5.2 Timeout

타임아웃 설정하기

세션 동작 시 적용할 timeout 값을 설정합니다.

C#	`session.Manage().Timeouts().ImplicitlyWait(TimeSpan.FromSeconds(0.5));`
Java	`session.manage().timeouts().implicitlyWait(1, TimeUnit.SECONDS);`
통신	`POST /session/:session_id/timeouts {"type":"implicit","ms":500.0}`

4.5.3 Frame

Mainframe 정보 확인하기

MainFrame 오브젝트의 titletext 속성값을 반환합니다.

C#	`session.Title;`
Java	`session.getTitle();`
통신	`GET /session/:sessionId/title`

4.5.4 Window

활성화된 윈도우 변경하기

현재 윈도우를 변경합니다. Windowhandle은 session.WindowHandles에서 얻을 수 있습니다. 최초 실행시에는 MainFrame 오브젝트가 현재 윈도우로 설정됩니다.

C#	`session.SwitchTo().Window(session.WindowHandles[0]);`
Java	`Set<String> handles = session.getWindowHandles();` `Set handles = session.getWindowHandles();`
통신	`POST /session/:sessionId/window {"name":"0x000A12BC"}`

활성화된 윈도우 종료하기

현재 윈도우를 종료합니다.

C#	`session.Close();`
Java	`session.close();`
통신	`DELETE /session/:sessionId/window`

활성화된 윈도우 핸들 얻기

현재 윈도우 핸들 오브젝트를 반환합니다.

C#	`handle = session.CurrentWindowHandle;`
Java	`Handle = session.getWindowHandle();`
통신	`GET /session/:sessionId/window_handle`

윈도우 핸들 배열 얻기

모든 윈도우 핸들 오브젝트를 배열 형태로 반환합니다.

C#	`handlearray = session.WindowHandles;`
Java	`Set handles = session.getWindowHandles();`
통신	`GET /session/:sessionId/window_handles`

윈도우 크기 정보 확인하기

윈도우 크기를 Size 오브젝트 형태로 반환합니다.

C#	`session.Manage().Window.Size;`
Java	`session.getWindowHandles().size()`
통신	`GET /session/:sessionId/window/:windowHandle/size`

윈도우 크기 변경하기

윈도우 크기를 변경합니다.

C#	`session.Manage().Window.Size = new Size(width,height);`
Java	`session.manage().window().setSize(new Dimension(width, height));`
통신	`POST /session/:sessionId/window/:windowHandle/size`

윈도우 위치 정보 확인하기

윈도우 위치를 Point 오브젝트 형태로 반환합니다.

C#	`point = session.Manage().Window.Position;`
Java	`point = session.manage().window().getPosition()`
통신	`GET /session/:sessionId/window/:windowHandle/position`

윈도우 위치 변경하기

윈도우 좌표를 변경합니다.

C#	`session.Manage().Window.Position = new Point(left,top);`
Java	`session.manage().window().setPosition(new Point(left, top))`

통신	POST /session/:sessionId/window/:windowHandle/position

윈도우 최대화

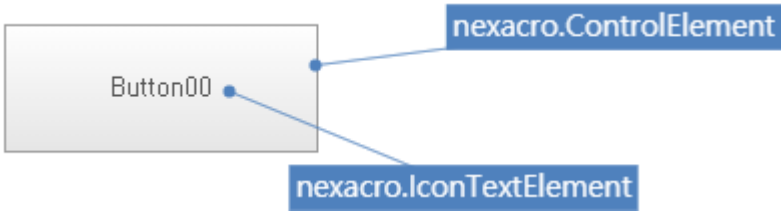
윈도우 크기를 최대화합니다.

C#	session.Manage().Window.Maxmize();
Java	session.manage().window().maximize();
통신	POST /session/:sessionId/window/:windowHandle/maximize

4.5.5 Element

Element는 WinAppDriver 동작의 최소단위입니다. 넥사크로에서는 컴포넌트, 컨트롤의 구성요소인 nexacro Element를 Element로 처리합니다.

ControlElement가 컴포넌트를 둘러싸고 있고 그 안에 ContainerElement, InputElement, IconTextElement, TextElement 등의 Element가 자리잡고 있습니다. Button 컴포넌트의 경우는 아래와 같이 구성됩니다.



엘리먼트 얻기

컴포넌트, 컨트롤의 ControlElement 오브젝트를 반환합니다.

C#	element = session.FindElementByAccessibilityId("mainframe.ChildFrame00.form.Button00");
Java	element = session.findElementByAccessibilityId("mainframe.ChildFrame00.form.Button00");
통신	POST /session/:sessionId/element { "using":"accessibilityid", "value":"mainframe.ChildFrame00.form.Button00" }

> 넥사크로 플랫폼에서 open 메소드 실행 시 objParentFrame 파라미터값을 null로 설정한 경우에는 ChildFrame ID를 기준으로 접근합니다.

C#

```
// parent null 지정시 : nexacro.open("openframe", url, null, ...)
nexaButton00 = session.FindElementByAccessibilityId("openframe.form.Button00");

// parent frame 지정시 : nexacro.open("openframe", url, ownerframe, ...)
nexaButton00 = session.FindElementByAccessibilityId("mainframe.ChildFrame00.openframe.form.
Button00");
```

엘리먼트 배열 얻기

컴포넌트, 컨트롤의 ControlElement 오브젝트 목록을 배열값으로 반환합니다.

C#	`element = session.FindElementsByAccessibilityId("mainframe.ChildFrame00.form.Button00");`
Java	`element = session.findElementsByAccessibilityId("mainframe.ChildFrame00.form.Button00");`
통신	`POST /session/:sessionId/elements` `{ "using":"accessibilityid", "value":"mainframe.ChildFrame00.form.Button00" }`

엘리먼트 id값 확인하기

WinAppDriver Element의 id값을 반환합니다.

C#	`id = element.Id;`
Java	`id = element.getTagName()`
통신	`GET /session/:sessionId/element/:id/name`

엘리먼트에 표시되는 텍스트 정보 확인하기

Element의 텍스트값을 반환합니다.

C#	`text = element.Text;`
Java	`text = element.getText();`
통신	`GET /session/:sessionId/element/:id/text`

> 넥사크로에서 지정한 속성값은 화면에 보여지는 텍스트를 우선적으로 처리합니다. 우선순위는 아래와 같습니다.
> displaytext > text > titletext > value

엘리먼트 마우스 클릭 이벤트 처리하기

Element의 가운데 위치에서 마우스 클릭 이벤트가 발생합니다.

C#	element.Click();
Java	element.click();
통신	POST /session/:sessionId/element/:id/click

> 엘리먼트의 가운데 위치에서 마우스 클릭 이벤트가 발생하기 때문에 MaskEdit 컴포넌트 같은 경우 캐럿 위치가 맨 앞으로 이동하지 않습니다. 이런 경우에는 CTRL+"A" 키를 입력해서 입력 영역 전체 선택 후 문자열을 입력하는 방법을 사용할 수 있습니다.

C#

```
[TestMethod]
public void Test_10_MaskEditInput()
{
    WindowsElement nexaMaskEdit00 = session.FindElementByAccessibilityId("mainframe.ChildFrame
00.form.MaskEdit00");
    Assert.IsNotNull(nexaMaskEdit00);
    Thread.Sleep(TimeSpan.FromSeconds(0.1));

    nexaMaskEdit00.Clear();
    nexaMaskEdit00.Click();
    nexaMaskEdit00.SendKeys(Keys.Control + "a");
    Thread.Sleep(TimeSpan.FromSeconds(0.1));
    nexaMaskEdit00.SendKeys("abcde1234"+Keys.Enter);
    Assert.AreEqual("abcde-1234", nexaMaskEdit00.Text);
}
```

엘리먼트 입력값 지우기

텍스트 편집 영역을 가지고 있는 컴포넌트의 입력값을 지웁니다.

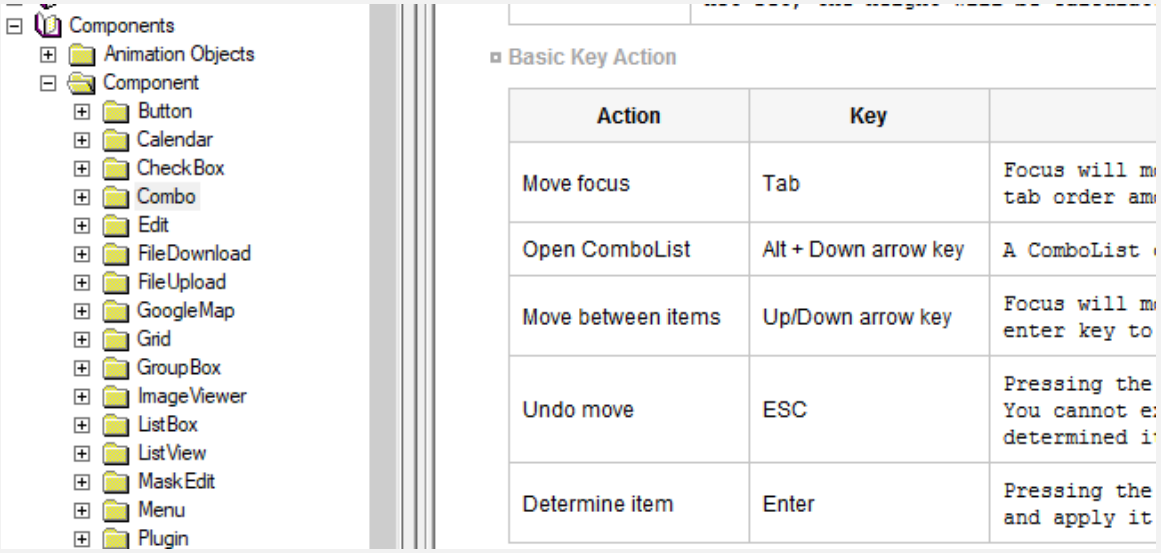
C#	element.Clear();
Java	element.clear();
통신	POST /session/:sessionId/element/:id/clear

엘리먼트에 키보드 입력값 전달하기

키보드 입력값을 처리합니다.

C#	`element.SendKeys(KeyString)`
Java	`element.sendKeys(KeyString)`
통신	`POST /session/:sessionId/element/:id/value`

컴포넌트에 따라 입력값이 처리되는 방식이 다릅니다.
제품 도움말에서 컴포넌트의 "Basic Key Action" 항목을 참고해주세요.



키보드 입력 값 전달 시에는 해당 앱이 활성화되어 있어야 하고 2개 이상의 윈도우가 표시된 경우 동작 대상 윈도우가 활성화되어야 합니다.

동작 대상 엘리먼트에 마우스 클릭 이벤트를 처리한 후 키보드 입력 값을 전달해야 정상적으로 동작합니다.

2개 이상의 디스플레이를 사용하는 경우 키보드 입력 값이 정상적으로 전달되지 않는 경우가 있습니다. 디스플레이를 하나만 사용하던지, WinAppDriver를 다시 시작해주세요.

엘리먼트 선택 여부 확인하기

Element의 Selected 상태값을 반환합니다.

C#	`isSelected = element.Selected;`
Java	`isSelected = element.isSelected();`
통신	`GET /session/:sessionId/element/:id/selected`

넥사크로에서 지원하는 컴포넌트, 컨트롤은 아래와 같습니다.
CheckBox, RadioButton, TabButton, ListBoxItem, MenuItem, TreeCell, GridCell

엘리먼트 사용 가능 여부 확인하기

Element의 Enabled 상태값을 반환합니다.

C#	`isEnabled = element.Enabled;`
Java	`isEnabled = element.isEnabled();`
통신	`GET /session/:sessionId/element/:id/enabled`

엘리먼트 화면 표시 여부 확인하기

Element의 화면 표시 여부를 반환합니다.

C#	`isDisplayed = element.Displayed;`
Java	`isDisplayed = element.isDisplayed();`
통신	`GET /session/:sessionId/element/:id/displayed`

컴포넌트의 visible 속성값을 반환하는 것이 아니라 실제 화면에서의 표시 여부를 반환합니다.

element.Displayed	입력동작
true	visible = true 화면 영역에 컴포넌트 전부 또는 일부가 보여지는 경우
false	visible = false visible = true 이지만 상위 컨테이너 컴포넌트의 스크롤바를 이동하지 않으면 화면에 보이지 않는 영역에 위치한 경우

> 컴포넌트의 visible 속성값을 false로 설정하고 앱이 시작하는 경우 해당 Element를 찾을 수 없습니다. 넥사크로에서는 visible 속성값이 true인 경우에만 Element를 생성하기 때문입니다.
> 앱 로드 시에는 visible 속성값을 true로 설정하고 onload 이벤트 함수 내에서 컴포넌트의 visible 속성값을 false로 변경하면 Element를 처리할 수 있습니다.

엘리먼트 크기 정보 확인하기

Element 크기를 Size 오브젝트 형태로 반환합니다.

C#	`size = element.Size;`
Java	`size = element.getSize();`
통신	`GET /session/:sessionId/element/:id/size`

엘리먼트 위치 정보 확인하기 (상위 엘리먼트 기준)

Element 위치를 상위 Element 기준에서 Point 오브젝트 형태로 반환합니다.

C#	`size = element.Location;`
Java	`size = element.getLocation();`
통신	`GET /session/:sessionId/element/:id/location`

엘리먼트 위치 정보 확인하기 (Screen 기준)

Element 위치를 Screen 기준에서 Point 오브젝트 형태로 반환합니다.

C#	`size = element.LocationOnScreenOnceScrolledIntoView;`
Java	지원하지 않음
통신	`GET /session/:sessionId/element/:id/location_in_view`

활성화된 엘리먼트 얻기

현재 활성화된 Element를 반환합니다.

C#	`element = session.SwitchTo().ActiveElement();`
Java	`element = session.switchTo().activeElement();`
통신	`GET /session/:sessionId/element/active`

2개의 엘리먼트 비교하기

Element를 비교해서 같은 Element인지 확인합니다.

C#	`isEqual = element.Equals(element2);`
Java	`isEqual = element.equals(element2);`
통신	`GET /session/:sessionId/element/:id/equals`

엘리먼트의 UIA 인터페이스 속성값 확인하기

Element의 UIA Interface의 attribute 값을 제공합니다.

C#	`element.GetAttribute("AutomationId");`
Java	`element.getAttribute("AutomationId");`
통신	`GET /session/:sessionId/element/:id/attribute/:name`

> Automation Element Property Identifiers
> https://docs.microsoft.com/en-us/windows/desktop/WinAuto/uiauto-automation-element-propids

4.5.6 ScreenShot

윈도우, 엘리먼트 스크린샷 얻기

Screen 윈도우 영역 또는 Element를 Screenshot 오브젝트로 반환합니다.

C#	`Screenshot sessionshot = session.GetScreenshot();`
Java	`Screenshot sessionshot = session.getScreenshotAs();`
통신	`GET /session/:sessionId/screenshot` `GET /session/:sessionId/element/:id/screenshot`

> SaveAsFile 메소드, AsBase64EncodedString 속성을 사용해 이미지 파일 또는 Base64 형태로 변환할 수 있습니다.

윈도우가 최상위에 노출되어야 정상적인 스크린샷을 얻을 수 있습니다.

Java 코드에서는 권한이 있는 저장소만 접근할 수 있습니다.

Java

```
FileHandler.copy(sessionshot, new File(System.getProperty("user.home")+"\\Downloads\\
SessionScreenShot.png"));
```

4.6 관련 링크 정보

WebDriver Interface

http://www.w3.org/TR/webdriver1/
https://www.seleniumhq.org/projects/webdriver/
https://developer.mozilla.org/en-US/docs/Web/WebDriver

AppDriver Interface

https://github.com/Microsoft/WinAppDriver
https://appium.io/docs/en/drivers/windows/
https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/

UIA Interface

https://docs.microsoft.com/en-us/windows/desktop/WinAuto/entry-uiauto-win32
https://docs.microsoft.com/en-us/windows/desktop/winauto/testing-tools
https://en.wikipedia.org/wiki/Microsoft_UI_Automation

MSAA Interface

https://docs.microsoft.com/en-us/windows/desktop/WinAuto/microsoft-active-accessibility
https://docs.microsoft.com/en-us/windows/desktop/winauto/testing-tools
https://en.wikipedia.org/wiki/Microsoft_Active_Accessibility

5.

안드로이드 운영체제 NRE에서 단위 테스트 실행하기

Appium 연동 기능은 넥사크로에서 제공하는 UIAutomation2 인터페이스를 사용해 Android NRE에서 실행하는 앱의 속성을 읽거나 쓰고 파라미터를 지정해 메소드를 실행하는 것을 지원합니다. 셀레니움 같은 UI 자동화 테스트 도구를 사용해 단위 테스트를 작성하고 테스트할 수 있습니다.

이번 장에서는 Appium를 설치하고 비주얼 스튜디오 코드에서 기본적인 Java 프로젝트를 생성해 넥사크로 앱에 연결하는 단계를 살펴봅니다.

> 아래 내용은 Appium 1.19.1, 안드로이드 스튜디오 4.1.1 버전을 기준으로 작성했습니다.

5.1 Appium 연동 환경 설정하기

5.1.1 Appium 설치하고 단말기 연결 확인하기

1. 안드로이드 스튜디오와 안드로이드 SDK, JDK가 설치되어 있는지 확인합니다.

2. 테스트할 단말기에 개발자 옵션을 활성화하고 아래 3개 옵션을 사용으로 체크해줍니다.

   - USB debugging
   - Install via USB
   - USB debugging (Security settings)

3. 테스트를 진행할 PC에 Appium 설치파일을 내려받고 설치합니다.

   > https://github.com/appium/appium-desktop/releases/latest

4. 단말기와 PC를 USB 케이블을 사용해 연결합니다.

5 adb 명령어로 연결된 단말기 일련번호를 확인합니다.

아래에서는 "0a388e93"이 연결된 단말기 일련번호입니다.

```
$ adb devices
List of devices attached
0a388e93       device usb:1-1 product:razor model:Nexus_7 device:flo
```

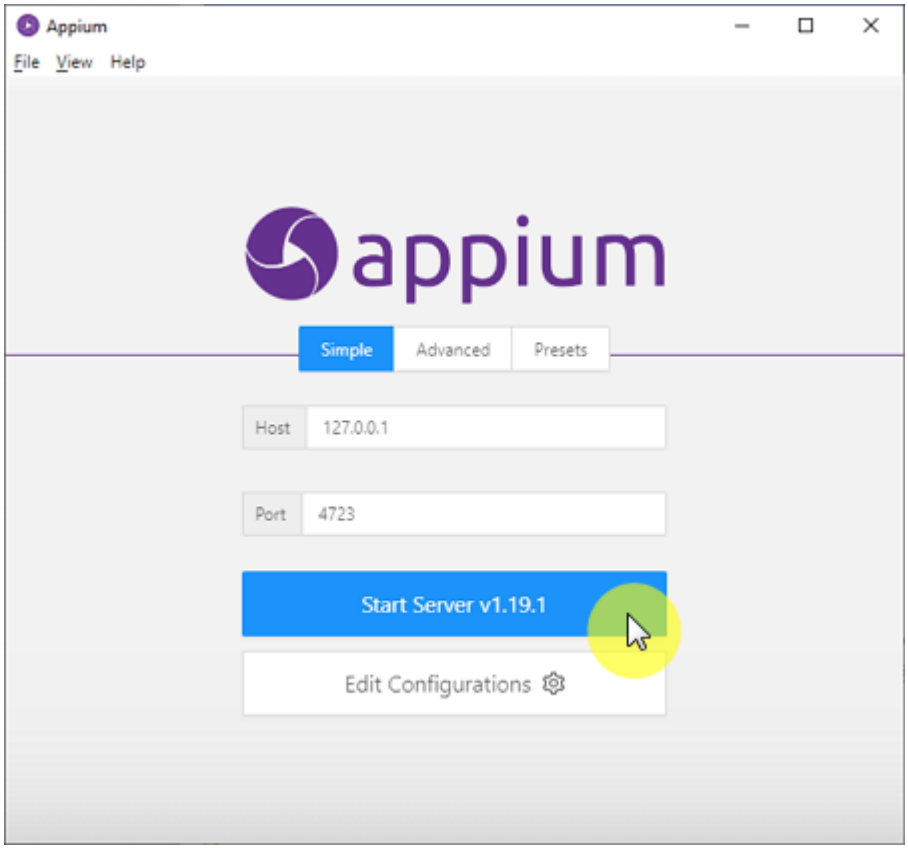
6 테스트할 단말기에 넥사크로 앱을 설치합니다.

앱 빌드 시 Package Name은 "nexacro", Program Name은 "TEST0120"으로 설정했습니다.

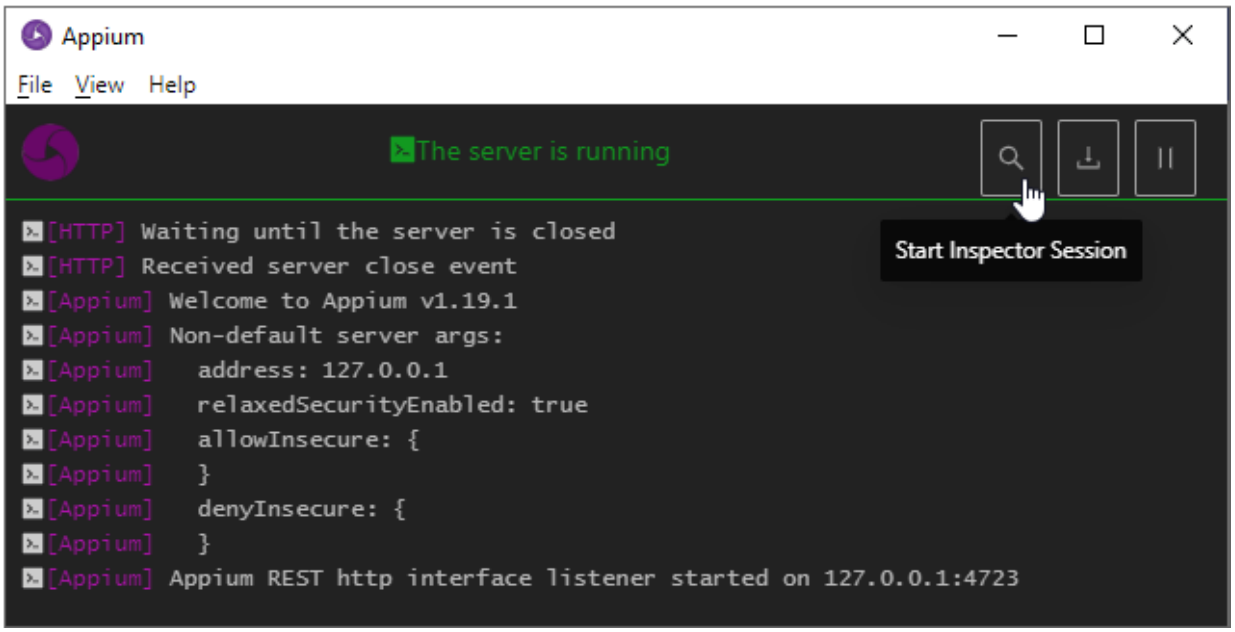
7 PC에서 Appium을 실행합니다.

8 [Start Server] 버튼을 클릭해 Appium 서버를 시작합니다.

윈도우 운영체제 환경이라면 Host 설정을 "127.0.0.1"로 변경합니다.



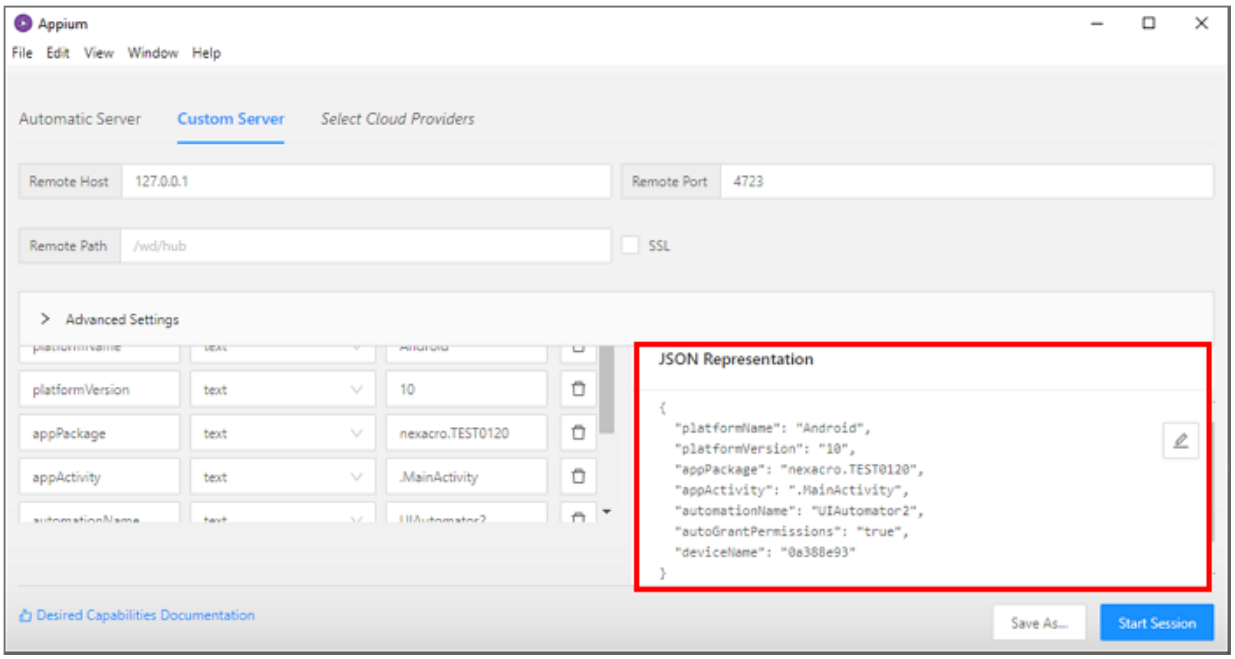
9 [Start Inspector Session] 버튼을 클릭해 Appium Inspector 창을 띄웁니다.



10 [Custom Server] 탭에서 JSON Representaion 항목을 아래와 같이 입력합니다.

```
{
  "platformName": "Android",
  "platformVersion": "10",
  "appPackage": "nexacro.TEST0120",
  "appActivity": ".MainActivity",
  "automationName": "UIAutomator2",
  "autoGrantPermissions": "true",
  "deviceName": "0a388e93"
}
```

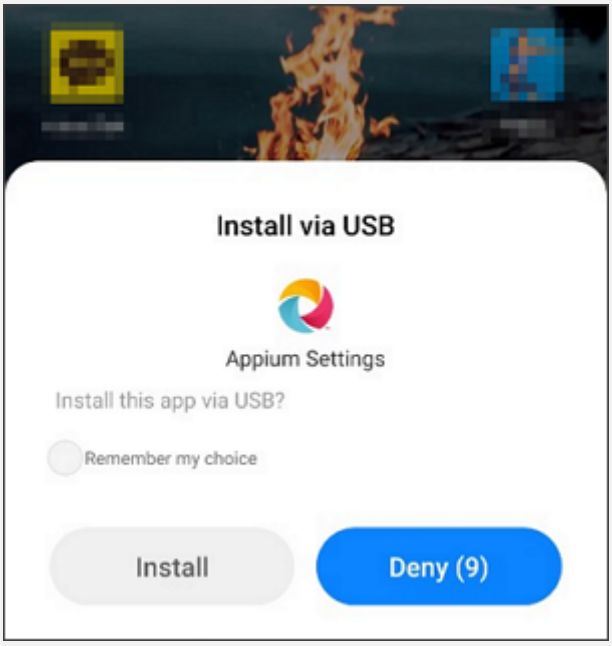
항목	설명 (굵게 표시한 항목은 고정값)
platformName	**"Android"**
platformVersion	테스트할 단말기 운영체제 버전
appPackage	테스트할 앱 (빌드 시 설정한 값) [Package Name].[Program Name] 형식
appActivity	앱 빌더로 빌드한 경우에는 **".MainActivity"** 안드로이드 스튜디오에서 빌드한 경우에는 설정한 액티비티
automationName	**"UIAutomator2"**
autoGrantPermissions	**"true"**
deviceName	adb 명령어로 확인한 단말기 일련번호



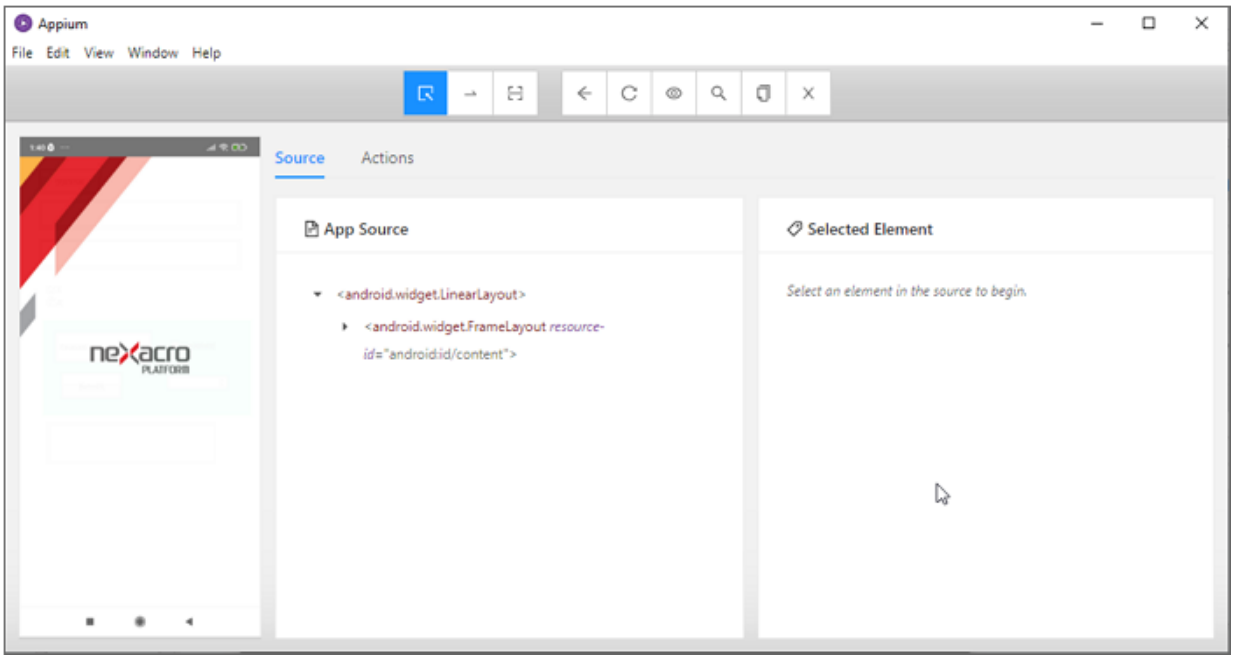
추가로 설정할 수 있는 항목은 아래 링크를 참고하세요.
http://appium.io/docs/en/writing-running-appium/caps/

⑪ [Start Session] 버튼을 클릭합니다.

최초 실행 시에는 단말기에 Appium Settings 앱이 설치됩니다.



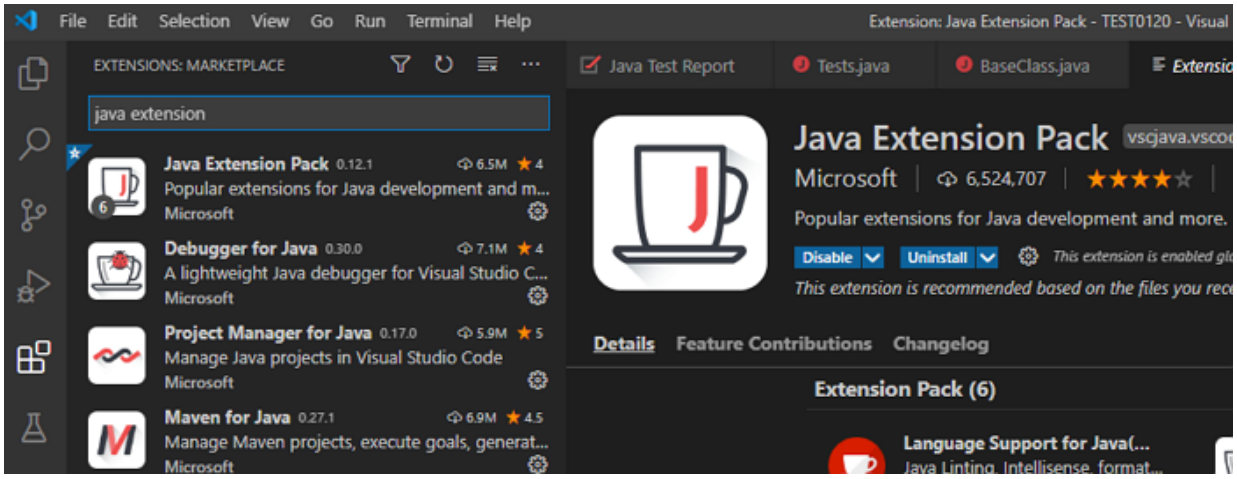
⑫ 테스트할 단말기에서 앱이 실행되고 Appium Inspector 창에 앱이 실행된 상태가 표시되는지 확인합니다.



5.1.2 테스트 프로젝트 실행 환경 설정하고 테스트하기

1. Visual Studio Code를 설치합니다.

2. 사이드바에서 Extensions를 선택하고 Java Extension Pack을 설치합니다.



3. 프로젝트 실행에 필요한 appium, selenium, testng 라이브러리를 설정합니다.

> Maven, Gradle, Eclipse plug-in 등의 설정은 관련 문서를 참고하세요.

- client-combined-3.141.59.jar (https://www.selenium.dev/downloads/)

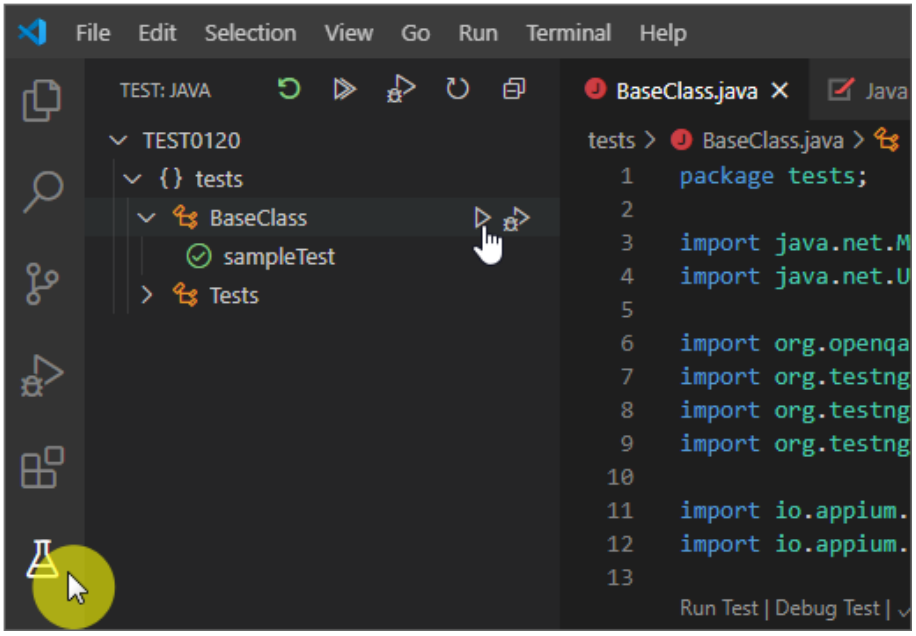
  - byte-buddy-1.8.15.jar
  - commons-exec-1.3.jar
  - guava-25.0-jre.jar
  - okhttp-3.11.0.jar
  - okio-1.14.0.jar
- testng-7.3.0.jar (https://testng.org/doc/download.html)
  - jcommander-1.78.jar
- java-client-7.4.1.jar (https://github.com/appium/java-client)
- commons-lang3-3.11.jar (https://commons.apache.org/proper/commons-lang/)

4 Java Project를 생성하고 BaseClass.java 파일을 아래와 같이 작성합니다.

```
package tests;

import java.net.MalformedURLException;
import java.net.URL;

import org.openqa.selenium.remote.DesiredCapabilities;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

import io.appium.java_client.MobileElement;
import io.appium.java_client.android.AndroidDriver;

public class BaseClass {
    AndroidDriver<MobileElement> driver;
    final String URL_STRING = "http://127.0.0.1:4723/wd/hub";

    @BeforeTest
    public void setup() {
        System.out.println("setup");
        DesiredCapabilities capabilities = new DesiredCapabilities();
        capabilities.setCapability("automationName", "UiAutomator2");
        capabilities.setCapability("platformName", "Android");

        capabilities.setCapability("platformVersion", "10");
        capabilities.setCapability("deviceName", "0a388e93");
        capabilities.setCapability("autoGrantPermissions", "true");

        capabilities.setCapability("appPackage", "nexacro.TEST0120");
```

```
        capabilities.setCapability("appActivity", ".MainActivity");

        try {
            driver = new AndroidDriver<MobileElement>(new URL(URL_STRING), capabilities);
        } catch (MalformedURLException e) {
            System.out.println("MalformedURLException");
            e.printStackTrace();
        } catch (Exception e) {
            System.out.println("Exception");
            e.printStackTrace();
        }
    }

    @Test
    public void sampleTest() {
        System.out.println("sampleTest");
    }

    @AfterTest
    public void teardown() {
        System.out.println("teardown");
        driver.close();
    }
}
```

5 사이드바에서 Test를 선택하고 BaseClass 항목에 있는 [Run] 버튼을 클릭합니다.



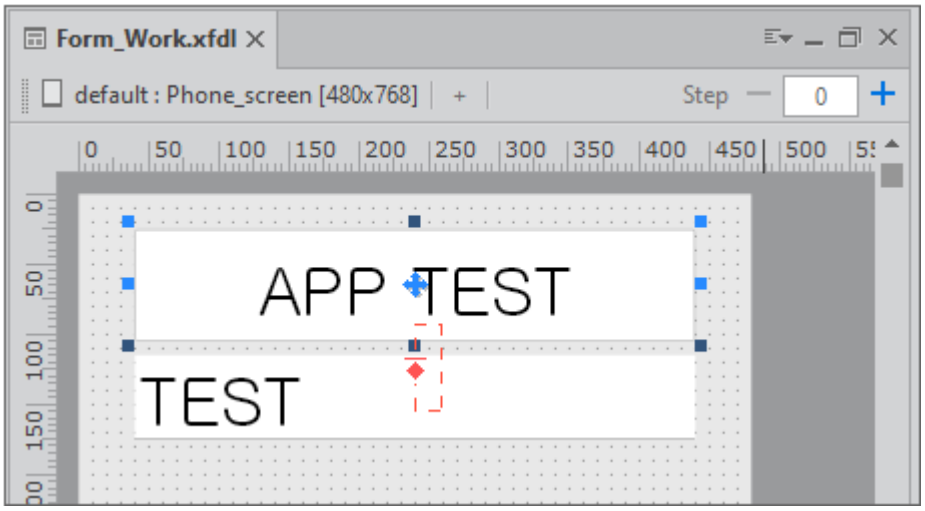
⑥ 테스트할 단말기에서 앱이 실행되고 DEBUG CONSOLE에 정상적으로 테스트 성공 메시지가 표시되는 것을 확인합니다.

```
OUTPUT   TERMINAL   DEBUG CONSOLE   PROBLEMS 15
setup
Jan 21, 2021 3:01:33 PM io.appium.java_client.remote.AppiumCor
INFO: Detected dialect: W3C
sampleTest
teardown

===============================================
Default Suite
Total tests run: 1, Passes: 1, Failures: 0, Skips: 0
Configuration Failures: 1, Skips: 0
===============================================
```

5.2 단위 테스트 만들고 실행하기

① 넥사크로 앱을 아래와 같이 수정하고 업데이트합니다.

상단에 Button 컴포넌트를 배치하고 그 아래에 Edit 컴포넌트를 배치합니다.



Button 컴포넌트의 onclick 이벤트 핸들러 함수를 아래와 같이 작성합니다.

```
this.Button00_onclick = function(obj:nexacro.Button,e:nexacro.ClickEventInfo)
{
    this.Edit00.set_value("BUTTON CLICK");
```

```
};
```

2 Tests.java 파일을 아래와 같이 작성합니다.

```
package tests;

import java.net.MalformedURLException;
import java.util.concurrent.TimeUnit;

import org.testng.annotations.Test;

public class Tests extends BaseClass {

    @Test
    public void testOne() throws MalformedURLException, InterruptedException {
        System.out.println("testOne");
        int nTestSpeed = 2000;

        driver.manage().timeouts().implicitlyWait(15, TimeUnit.SECONDS);
        driver.findElementById("mainframe.WorkFrame.form.Button00").click();
        Thread.sleep(nTestSpeed);
    }
}
```

3 사이드바에서 Test를 선택하고 Tests 항목에 있는 [Run] 버튼을 클릭합니다.

4 테스트할 단말기에서 앱이 실행되고 Button 클릭 이벤트가 정상적으로 처리되어 Edit 컴포넌트의 값이 변경되는지 확인합니다.

5.3 지원 인터페이스

Method	Return	Argument	Requirement
AndroidDriver `driver.findElementById("mainframe.WorkFrame.form.Button00")`			
findElement	WebElement	By by	
findElementById	WebElement	String id	

Method	Return	Argument	Requirement
findElementByName	WebElement	String using	
findElementByClassName	WebElement	String using	
findElementByXPath	WebElement	String using	
getKeyboard	Keyboard	-	
pressKeyCode	-	AndroidKeyCode	
WebElement `driver.findElementByClassName("android.widget.CheckBox").click();`			
getText	String	-	
getAttribute	String	String name	
getClass	String	-	
getRect	Rectangle	-	
getSize	Dimension	-	
getLocation	Point	-	
isEnabled	Boolean	-	
click	-	-	
sendKeys	-	CharSequence... keysToSend	

TouchAction
`TouchAction touch = new TouchAction(driver);`
`touch.longPress(PointOption.point(500,1100))`
`.release()`
`.perform();`			

longPress	PointOption	-	
waitAction	WaitOptions	-	
moveTo	PointOption	-	
release	-	-	
perform	-	-	
Keyboard `driver.getKeyboard().sendKeys("Keyboard Input Test");`			
sendKeys	-	CharSequence... keysToSend	
pressKey	-	CharSequence... keysToSend	
releaseKey	-	CharSequence... keysToSend	

파트 II.

동작 원리

6.

프로젝트 및 파일 구조

넥사크로에서 화면을 만들려면 먼저 프로젝트를 생성해야 합니다. 그리고 프로젝트에 필요한 컴포넌트, 스크린 정보, 환경 정보, 테마 또는 스타일 등의 파일을 생성하거나 기존 정보를 연결합니다. 화면은 앱 형태로 실행될 수 있으며 Form 화면 자체로 실행될 수 있습니다.

이번 장에서는 프로젝트나 Form 생성 시 작업 폴더에 생성되는 각 파일은 어떤 것이 있으며 어떤 기능을 수행하는지 설명합니다. 추가로 제너레이트 과정을 거치면서 변환되는 파일도 설명합니다.

6.1 프로젝트 생성

프로젝트는 화면 개발을 위한 기본 설정 정보를 담고 있습니다. 프로젝트를 새로 만들면 프로젝트 설정 파일 뿐 아니라 프로젝트를 구성하기 위한 기본적인 파일이 해당 폴더에 생성됩니다.

일반적으로 프로젝트 생성 시 만들어지는 파일은 아래와 같습니다.

항목	확장자	설명
Nexacro Project	XPRJ	프로젝트 설정 파일 [프로젝트명]+확장자
Application Info	XADL	Application 설정 파일 프로젝트 생성 시 설정한 스크린 정보와 기본 테마 정보를 속성값으로 지정합니다. 기본 생성되는 파일은 다음과 같은 형태입니다. [프로젝트명]+확장자
Type Definition	XML	모듈, 컴포넌트, 서비스, 프로토콜, 업데이트 정보 설정 파일 생성되는 파일명은 typedefinition.xml 입니다.
Environment	XML	Screen 정보, Variable, Cookie, HTTP Header 정보를 관리합니다. 생성되는 파일명은 environment.xml 입니다.
AppVariables	XML	Application에서 공통으로 사용할 수 있는 Dataset, Variable 정보를 관리합니다. 해당 정보를 입력하지 않아도 파일은 생성됩니다. 생성되는 파일명은 appvariables.xml 입니다.
resource	폴더	Theme, InitValue, Image, font 관련 리소스를 관리합니다.

항목	확장자	설명
		아래와 같은 하위 폴더를 생성합니다. [_font_], [_images_], [_initvalue_], [_theme_], [_xcss_]
Base	폴더	서비스 폴더 프로젝트 생성 시 기본 form 타입 서비스를 지정합니다. 기본 생성되는 서비스 PrefixID는 [Base]입니다. 생성된 PrefixID나 폴더명은 다른 이름으로 수정하거나 삭제할 수 있습니다.

프로젝트 생성 시 [Frame] 항목 설정에 따라 서비스 폴더는 추가될 수 있습니다.
예를 들어 두 개 이상의 프레임을 가지는 화면을 설정했다면 Frame 서비스 폴더가 자동 생성됩니다.

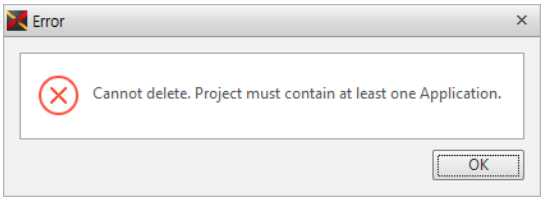
6.1.1 프로젝트 설정 파일 (XPRJ)

프로젝트 생성 시 자동으로 생성됩니다. Application 설정 파일 정보는 추가하거나 삭제할 수 있습니다.

```
<?xml version="1.0" encoding="utf-8"?>
<Project version="2.1">
  <EnvironmentDefinition url="environment.xml"/>
  <TypeDefinition url="typedefinition.xml"/>
  <AppVariables url="appvariables.xml"/>
  <AppInfos>
        <AppInfo url="[file name].xadl"/>
        <AppInfo url="[file name].xadl"/>
  </AppInfos>
</Project>
```

6.1.2 Application 설정 파일 (XADL)

프로젝트 생성 시 자동으로 생성되며 필요에 따라 추가로 생성할 수 있습니다. 프로젝트는 하나 이상의 App Info를 포함해야 합니다. App Info가 하나인 경우 이를 삭제하려 하면 아래와 같은 경고 메시지를 보여주고 삭제하지 않습니다.



```
<?xml version="1.0" encoding="utf-8"?>
<ADL version="2.0">
  <Application id="[id]" screenid="[screen id]">
    <Layout>
      <MainFrame/>
    </Layout>
  </Application>
  <Script type="xscript5.1"><![CDATA[…]]></Script>
</ADL>
```

Application 설정 파일은 스크립트 코드를 추가할 수 있습니다. ADL 또는 각 프레임 오브젝트의 이벤트 처리나 Application 단위에서 처리할 스크립트를 관리합니다.

6.1.3 Type Definition

모듈, 컴포넌트, 서비스, 프로토콜, 업데이트 정보 설정 파일 정보를 관리합니다. 각 항목은 프로젝트 생성 후 추가, 변경, 삭제할 수 있습니다. 프로토콜 정보와 배포 옵션은 빌드 과정에서 자동으로 편집되기도 합니다.

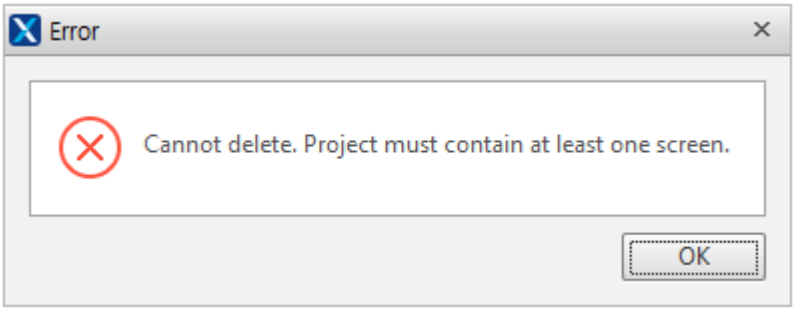
	항목	설명
Modules	모듈 정보	자바스크립트로 개발된 모듈 정보를 담고 있습니다. - 자체 개발된 모듈을 추가할 수 있습니다. - js 또는 json 파일 형태로 등록할 수 있습니다. 각 파일 간에 의존성을 가질 때에는 순서에 맞게 등록해야 합니다.
Components	오브젝트 정보	앱 실행에 필요한 오브젝트 정보를 담고 있습니다. - 지정된 오브젝트가 배포에 포함됩니다. - 각 오브젝트는 모듈을 기반으로 지정됩니다.
Services	서비스 그룹	프로젝트 규모에 따라 내부적으로 서비스 그룹을 정의합니다. - 각 서비스 그룹은 개별적으로 캐시 수준을 설정할 수 있습니다.
Protocols	프로토콜 정보	프로토콜을 등록/삭제하고 장치 타입에 맞게 편집합니다.
Update	배포 옵션	앱 배포 관련 정보를 지정합니다.

```
<?xml version="1.0" encoding="utf-8"?>
<TypeDefinition version="3.0">
  <Modules>
    <Module url="CompBase.json"/>
    <Module url="ComComp.json"/>
    <Module url="Grid.json"/>
    <Module url="DeviceAPI.json"/>
  </Modules>
```

```
  <Components>
    <Component type="JavaScript" id="Button" classname="nexacro.Button"/>
    <Component type="JavaScript" id="Combo" classname="nexacro.Combo"/>
    <Component type="JavaScript" id="Edit" classname="nexacro.Edit"/>
    ...
  </Components>
  <Services>
    <Service prefixid="theme" type="resource" url="./_resource_/_theme_/" version="0"
communicationversion="0"/>
    <Service prefixid="initvalue" type="resource" url="./_resource_/_initvalue_/" version="0"
communicationversion="0"/>
    <Service prefixid="xcssrc" type="resource" url="./_resource_/_xcss_/"/>
    <Service prefixid="imagerc" type="resource" url="./_resource_/_images_/"/>
    <Service prefixid="font" type="resource" url="./_resource_/_font_/" version="0"
communicationversion="0"/>
    <Service prefixid="Base" type="form" cachelevel="session" url="./Base/" version="0"
communicationversion="0"/>
  </Services>
  <Protocols/>
  <Update/>
</TypeDefinition>
```

6.1.4 Environment

Screen 정보, Variable, Cookie, HTTP Header 정보를 관리합니다. 프로젝트는 하나 이상의 Screen 정보를 포함해야 합니다. Screen 정보가 하나인 경우 이를 삭제하려 하면 아래와 같은 경고 메시지를 보여주고 삭제하지 않습니다.




```
<?xml version="1.0" encoding="utf-8"?>
<ENV version="2.1">
  <Environment themeid="theme::default" datatyperule="2.0">
    <ScreenDefinition>
```

```
      <Screen id="[Screen id]" type="[type property value]"/>
    </ScreenDefinition>
    <Variables/>
    <Cookies/>
  </Environment>
</ENV>
```

6.1.5 AppVariables

App에서 공통으로 사용할 수 있는 Dataset, Variable 정보를 관리합니다.

```
<?xml version="1.0" encoding="utf-8"?>
<AppVariables version="2.0">
  <Datasets>
    <Dataset id="[Dataset id]">
      <ColumnInfo/>
      <Rows/>
    </Dataset>
  </Datasets>
  <Variables>
    <Variable id="[Variable id]" initval="[init value]"/>
  </Variables>
</AppVariables>
```

6.2 리소스

6.2.1 initValue

프로젝트 생성 시에는 폴더만 생성하고 파일을 만들지는 않습니다. initValue 설정값을 저장할 XML 파일을 생성하고 해당 파일에 필요한 오브젝트의 속성 정보를 지정합니다. 파일을 생성하면 오브젝트를 선택하고 initValue를 추가할 수 있습니다. XML 파일 형식이지만 확장자는 .xiv입니다.

```
<?xml version="1.0" encoding="utf-8"?>
<GlobalPropInitvalueDefinition>
    <nexacro.Button>
        <initvalueid id="[initvalueid ]" ...  />
    </nexacro.Button>
    <nexacro.Edit>
        <initvalueid id="[initvalueid ]" ... />
    </nexacro.Edit>
</GlobalPropInitvalueDefinition>
```

6.2.2 테마 (XTHEME)

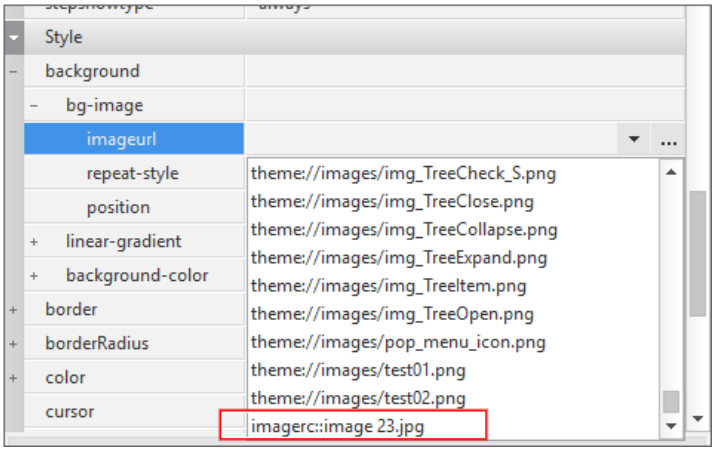
프로젝트 생성 후 넥사크로 스튜디오 [Resource Explorer] 창을 확인해보면 기본 설치 테마가 포함된 [Nexacro Theme] 항목과 새로운 테마를 추가할 수 있는 [Theme] 항목이 표시됩니다. [Theme] 항목에는 기존 테마를 복사해 넣거나 새 테마를 만들 수 있습니다.

기본 테마는 스타일 설정 파일(theme.xcss)과 이미지 파일 집합으로 구성됩니다.

> [Nexacro Theme] 항목 아래에 표시되는 테마 파일은 별도 캐시 공간에 저장되어 관리됩니다. 기본 테마 파일은 수정할 수 없습니다.

6.2.3 ImageResource

프로젝트 내에서 공통으로 사용할 수 있는 이미지 파일을 관리합니다. Image 리소스로 추가한 이미지 파일은 Form 작성 시 이미지 파일이 필요한 속성을 지정할 때 속성창에 노출되며 해당 이미지 파일을 속성값으로 지정할 수 있습니다.



6.2.4 UserFont

프로젝트 내에서 사용할 폰트 파일을 관리합니다. 폰트 파일(woff, ttf, off)과 폰트를 정의한 UserFont 파일을 관리합니다. UserFont 파일은 xfont 확장자로 저장되며 아래와 같은 형식으로 작성합니다.

```
@font-face {
    font-family : 'NanumGothic';
    font-style : normal;
    font-weight : 400;
    src : local("NanumGothic"),
            url(NanumGothic.ttf) format('truetype'),
            url(NanumGothic.eot),
            url(NanumGothic.eot?#iefix) format('embedded-opentype'),
            url(NanumGothic.wpff2) format('woff2'),
            url(NanumGothic.woff) format('woff'),
}
```

6.3 기타

6.3.1 Form (XFDL)

사용자가 업무를 수행하기 위해 접하는 화면을 정의하고 화면 내에서 보이지 않게 처리되는 연산이나 서버와 데이터 통신을 위한 프로그래밍 코드를 담고 있습니다.

요소	설명
컴포넌트 배치	Visible Object인 컴포넌트를 배치하는 정보를 갖고 있습니다. 컴포넌트의 속성 및 이벤트를 설정합니다.
오브젝트 설정	Dataset 등 Invisible Object의 Property 및 Event를 설정합니다.
Script	Form을 포함한 모든 오브젝트의 Event Function을 스크립트로 작성합니다. Event Function 외에도 필요한 Function들을 스크립트로 작성합니다.
BindItem	Form, 컴포넌트, Invisible Object들과 Dataset을 연결하는 BindItem Object를 설정합니다.

```
<?xml version="1.0" encoding="utf-8"?>
<FDL version="2.1">
    <Form>
```

```
        <Layouts>
            <Layout>
                <Button/>
            </Layout>
        </Layouts>
        <Script type="xscript5.1"><![CDATA[]]></Script>
        <Objects>
            <Dataset/>
        </Objects>
        <Bind>
            <BindItem/>
        </Bind>
    </Form>
</FDL>
```

> Form에서 스타일 설정 파일을 별도 지정할 수는 없습니다.
> 스타일 설정 파일은 Application 단위로만 지정할 수 있습니다.

6.3.2 스크립트 (XJS)

특정 Form에만 필요한 스크립트는 Form 파일에 같이 작성하지만, Application 내 전체 화면에서 사용하는 스크립트는 별도 작성하고 각 Form에서 이를 참조하도록 합니다.

작성된 스크립트는 Form에서 다음과 같이 참조할 수 있습니다.

```
include "lib::comLib.xjs"
```

공통으로 참조하는 스크립트가 많은 경우 여러 파일로 분리해 관리할 수도 있습니다. 참조하는 파일이 많은 경우에는 스크립트 코드 안에서 다시 다른 코드를 참조하도록 합니다. 예를 들어 comLib.xjs 스크립트 파일을 다음과 같이 구성할 수 있습니다. 이렇게 구성하면 참조할 파일이 늘어나더라도 업무에서 사용하는 Form 파일은 건드리지 않고 comLib.xjs 파일만 수정해 기능을 추가할 수 있습니다.

```
include "lib::comConstants.xjs";
include "lib::comService.xjs";
include "lib::comForm.xjs";
include "lib::comGrid.xjs";
include "lib::comUtil.xjs";
include "lib::comDataset.xjs";
```

6.3.3 스타일 설정 파일 (XCSS)

CSS 표준이 있지만, 여전히 웹브라우저마다 적용되는 방식이 다릅니다. 그래서 개발 과정에서 특정 브라우저마다 필요한 CSS 속성을 지정해주어야 합니다. 넥사크로는 CSS 표준에 따라 하나의 코드만 작성해주면 빌드 과정에서 각 브라우저에 맞는 CSS 코드를 자동으로 생성합니다.

```
<?xml version="1.0" encoding="utf-8"?>
<XCSS type="xcss1.0"><![CDATA[.Button
{
    background : #c0504d;
    -nexa-text-align : left;
}
]]></XCSS>
```

스타일 설정 파일에 위의 예제처럼 표준 CSS 속성과 넥사크로에서 추가 지원하는 속성이 같이 지정된 경우 빌드 과정에서 아래와 같은 CSS 코드가 생성됩니다.

```
.Button
{
    background        :        #c0504d;
}
.Button .nexacontentsbox
{
    text-align        :        left;
}
```

> 스타일 설정 파일은 XML 형식으로 작성합니다. 또한, 내부에서 사용하는 코드가 표준 CSS를 준수하는 것이지 모든 CSS 속성을 지원하는 것은 아닙니다.

> 스타일 설정 파일은 Application 단위로만 지정할 수 있습니다.
> 전체 프로젝트 스타일은 테마, Application 단위 스타일은 스타일 설정 파일을 사용합니다.

7.

앱 구동 시나리오

넥사크로 앱이 실행되는 과정은 아래와 같습니다.

7.1 bootstrap

Application을 로딩하기 위해 관련 정보를 확인하고 필요한 자원을 가져오는 과정을 설명합니다. 이 과정을 부트스트랩이라고 표현합니다.

<table>
<tr><th></th><th>WRE</th><th>NRE</th></tr>
<tr><td>❶</td><td>index.html</td><td>start.json
start_android.json
start_ios.json</td></tr>
<tr><td>❷</td><td colspan="2">get bootstrap info</td></tr>
<tr><td>❸</td><td rowspan="2"></td><td>update engine</td></tr>
<tr><td>❹</td><td>update resource (data)</td></tr>
<tr><td>❺</td><td colspan="2">load framework files</td></tr>
<tr><td>❻</td><td colspan="2">load component modules</td></tr>
<tr><td>❼</td><td colspan="2">load Application</td></tr>
</table>

> 일반적으로 부트스트랩은 컴퓨터가 외부 입력 없이 스스로 시작할 준비를 하는 과정을 의미합니다. 주로 메모리에 담긴 정보를 기반으로 동작하며 부팅(booting)이라고 표현하기도 합니다.
> https://en.wikipedia.org/wiki/Bootstrapping
> 넥사크로 앱 실행과정에서 사용하는 부트스트랩이라는 용어는 앱을 시작할 준비를 하는 과정이라는 표현을 담고 있습니다.

7.2 Application 로딩

앱이 실행되기 위해 Application과 연결된 폼을 로딩하는 과정은 아래와 같습니다.

	로딩 순서	이벤트 발생
❶	Environment 로딩	
❷	Screen 선택 (Screen 정보가 2개 이상인 경우)	
❸	Screen 환경 정보 로딩	
	- 선택된 Screen 정보를 screenid 속성값으로 지정한 App 확인	
	- Applicaiton 오브젝트 생성	
❹	Environment 설정 처리 (Typedefinition, 속성, Variables)	
		Environment.onload
❺	테마 로딩 (css, map.js 파일)	
❻	Initvalue 파일 로딩	
❼	Application 로딩 (*.xadl.js 파일)	
	- Application 속성 설정	
		Application.onloadingappvariables
	- Mainframe 생성 및 초기화	
		MainFrame.onactivate
	- Frame 생성 및 초기화	
		ChildFrame.onactivate
		Application.onload
❽	Form 로딩 (*.xfdl.js 파일)	
		Form.oninit
		Form.onload
		Form.onactivate
		Application.onloadforms

> 로딩 순서에 따라 아직 생성되지 않는 항목은 사용할 수 없습니다.
> 예를 들어 Environment.onload 이벤트 함수 내에서 AppVariables에 접근하게 되면 undefined로 표시됩니다.

```
this.Environment_onload = function(obj:nexacro.Environment,e:nexacro.LoadEventInfo)
{
    trace(nexacro.getApplication().all['Variable0']); //undefined
}
```

7.3 폼 로딩

앞에서 설명한 Application 로딩 항목에서 'Form 로딩'부터 'Form.onload 이벤트 발생' 사이의 과정을 좀 더 상세히 살펴보면 아래와 같습니다.

1. Form 로딩
2. Form 실행
3. 서브파일 다운로드 및 실행
4. Form 초기화
5. Form Style 속성 설정
6. Form 속성 설정
7. 오브젝트, 컴포넌트, 바인드아이템(BindItem) 생성
8. 엔진 내부에서 처리하는 스크립트 수행
9. 이벤트핸들러 등록
10. 오브젝트, 컴포넌트 메소드 호출: `Object.createComponent`
11. **이벤트 발생**: `Form.oninit`
12. 메소드 호출: `Form.on_created`
13. 오브젝트, 컴포넌트 메소드 호출: `Object.on_created`
14. 이미지 로딩, transaction 처리
15. **이벤트 발생**: `Form.onload`

서브파일은 include 방식으로 연결되는 스크립트 파일 등을 의미합니다.

Form에 포함된 Object, Component는 지정된 Z-Order 순서대로 생성합니다.

div, tab 컴포넌트는 url 속성으로 연결된 Form을 가져오는 네트워크 속도에 따라 로딩이 끝나는 시점이 달라질 수 있습니다.

8.

넥사크로플랫폼 스크립트 언어

넥사크로 앱을 개발할 때는 자바스크립트 문법을 기본적으로 사용합니다. 자바스크립트 기본 문법에서 제공하지 않는 추가적인 기능을 처리하기 위해 별도 API를 제공합니다. 작성된 스크립트 코드는 빌드 과정을 거쳐 그대로 웹 브라우저나 모바일 디바이스에서 동작하도록 자바스크립트 코드로 변환됩니다.

이번 장에서는 넥사크로에서 스크립트 작성 시 기본적으로 알아야 하는 문법적인 요소와 주의 사항을 설명합니다.

8.1 유효범위(Scope)

유효범위는 접근 가능한 변수의 범위와 관련된 내용을 설명합니다. 유효범위 또는 영역이라고 표기하며 영문 표기 그대로 사용하기도 합니다. 업무에 사용하는 앱은 여러 개의 화면을 동시에 조작하거나 하나의 변수를 여러 화면에서 참조하는 경우가 많습니다. 유효범위를 사용하면 원하는 자원에 정확하게 접근할 수 있습니다.

8.1.1 로컬 (local)

넥사크로의 Form 스크립트에서 첫 번째 줄에 아래와 같은 코드가 있는 경우 로컬 변수로 처리됩니다. 일반적인 자바스크립트 코드였다면 전역 변수로 처리됩니다. 하지만 넥사크로는 그렇지 않습니다.

```
var value0 = 0;
```

해당 코드가 빌드 과정을 거쳐 브라우저에서 동작하는 코드를 보면 어떤 차이가 있는지 확인할 수 있습니다. 빌드 과정을 거친 코드는 아래와 같습니다. 첫 번째 줄에 정의된 변수가 자동으로 생성된 함수 내에 포함되면서 로컬 변수로 처리됩니다.

```
this.registerScript("TEST.xfdl", function() {
    var value0 = 0;
});
```

> Form 스크립트에서 전역 변수를 처리하고자 한다면 var 키워드를 사용하지 않고 변수를 정의할 수 있습니다. 하지만 전역 변수를 사용하면 앱 전체에 영향을 미칠 수 있어 적절한 유효범위를 지정해주어야 합니다.

함수 내에서 선언된 변수는 유효범위를 따로 지정하지 않고 사용할 수 있으며 매개변수로 전달된 값 역시 유효범위를 지정하지 않고 사용할 수 있습니다.

```
this.Button00_onclick = function(obj:nexacro.Button, e:nexacro.ClickEventInfo)
{
    obj.set_text("button");
    var a = 3;
    trace(a); // 3
}
```

8.1.2 this

Application 또는 Form에서 사용되는 항목에 대해 유효범위를 지정할 때는 항상 this를 사용합니다. 변수뿐 아니라 속성, Form 간의 참조 시에도 적절한 유효범위를 지정해주어야 합니다.

Form 내에서 변수는 아래와 같이 선언합니다.

```
this.formvalue = 4;
```

함수를 선언할 때도 유효범위를 지정해주어야 합니다. 또한, 함수를 작성할 때 Form 내에 선언된 변수를 참조할 때도 유효범위를 지정해주어야 합니다.

```
this.formvalue = 4;

this.test = function()
{
    this.formvalue = 3;
    this.parent.value = 3;
    this.parent.parent.value = 3;
}
```

trace나 alert와 같은 메소드는 지정된 유효범위에 따라 자바스크립트 기본 문법에서 제공하는 메소드를 사용하거나 넥사크로에서 추가로 제공하는 메소드를 사용할 수 있습니다. 유효범위를 지정하지 않고 alert 메소드를 쓰는 것과 this 유효범위를 지정하는 것은 전혀 다른 메소드가 실행하는 것입니다.

```
this.Button00_onclick = function(obj:nexacro.Button, e:nexacro.ClickEventInfo)
{
    alert(e.button); // JavaScript
    this.alert(e.button); // Form.alert()
    nexacro.getApplication.alert(e.button); // Application.alert()
}
```

함수 선언 시에도 유효범위를 지정해 해당하는 Application 또는 Form의 멤버로 포함되도록 합니다.

```
this.Button00_onclick = function(obj:nexacro.Button, e:nexacro.ClickEventInfo)
{
...
}
```

eval 함수로 Form을 접근하는 경우에도 this를 사용해야 합니다.

```
eval("this.formfunc()");
```

아래와 같이 함수를 직접 선언할 수도 있지만, Global로 처리되며 이후 지원이 중단될 수 있으므로 권장하지 않습니다.

```
function Button00_onclick(obj:nexacro.Button, e:nexacro.ClickEventInfo)
{
...
}
```

8.1.3 Global

넥사크로 스크립트 내에서 유효범위를 지정하지 않은 변수나 함수는 모두 최상위 Global 멤버로 처리됩니다. 아래와 같이 유효범위를 지정하지 않고 사용된 스크립트는 모두 Global로 처리됩니다.

```
globalvar = 2;
var globalvar2 = 3;
```

함수 내에 사용된 변수라도 유효범위를 지정하지 않으면 모두 Global로 처리됩니다. 단 var 키워드를 사용해 정의한 변수는 함수 밖에서 사용할 수 없습니다.

```
this.test = function()
{
    trace(globalvar); //2
    trace(globalvar2); //3

    value = 4;
    var localvar = 5;
}


this.test = function()
{
    trace(value); // 4
    trace(localvar); // ReferenceError: localvar is not defined
}
```

Form 내에서 다른 스크립트 파일을 include 하는 경우에는 내부적으로 스크립트를 함수로 처리합니다. 그래서 var 키워드를 사용할 때 주의해야 합니다.

예를 들어 includecode.xjs 파일 내 정의된 변수가 실제 generate 된 includecode.xjs.js 코드는 아래와 같습니다 (generate 된 코드는 버전에 따라 달라질 수 있습니다).

includecode.xjs

```
bbb = "bbb";
var ccc = "ccc";
```

includecode.xjs.js

```
(function()
{
    return function(path)
    {
        var obj;

        // User Script
        this.registerScript(path, function() {
            bbb = "bbb";
            var ccc = "ccc";
        });
```

```
        this.loadIncludeScript(path);
        obj = null;
    };
}
)();
```

함수를 선언하거나 호출할 때도 유효범위를 지정해주어야 합니다. 아래 스크립트에서 test()와 this.test()는 서로 다른 함수를 호출합니다. 유효범위를 지정하지 않고 test()를 호출하게 되면 Global에 선언된 test 함수를 호출합니다.

```
this.Button00_onclick = function(obj:nexacro.Button, e:nexacro.ClickEventInfo)
{
     test();
     this.test();
}
```

아래와 같이 유효범위 없이 지정된 함수는 Global로 처리됩니다.

```
Button00_onclick = function(obj:nexacro.Button, e:nexacro.ClickEventInfo)
{
...
}

function Button00_onclick(obj:nexacro.Button, e:nexacro.ClickEventInfo)
{
...
}
```

transaction()과 같은 Application 또는 Form 메소드는 해당하는 유효범위를 지정해주어야 합니다.

```
//application
nexacro.getApplication().exit("");
nexacro.getApplication().transaction("");

//Form
this.close("");
this.go("");
this.transaction("");
```

> Application 전체에서 사용하고 싶은 변수는 Global 변수가 아닌 AppVariables로 등록해서 사용하는 것을 권장합니다.

```
nexacro.getApplication().addVariable("aaa", "TEST");
trace(nexacro.getApplication().aaa); // TEST
```

8.1.4 Expr

Grid 컴포넌트 내에서 사용하는 expr 스크립트는 넥사크로 내부적으로 바인딩 된 Dataset을 기준으로 처리되기 때문에 별도의 유효범위를 지정하지 않아도 사용할 수 있습니다. 바인딩 되지 않은 Dataset에 직접 접근하려면 유효범위를 지정해주어야 합니다.

아래와 같이 바인딩 된 Dataset에 대해 expr를 사용하는 경우에는 this를 붙이지 않습니다.

```
<Cell text="expr:Column00"/>
<Cell text="expr:Column00.min"/>
<Cell text="expr:currow"/>
<Cell text="expr:rowposition"/>
<Cell text="expr:getSumNF('col0')"/>
```

```
Dataset.set_filterstr("Column00=='test'");
Dataset.filter("Column00=='test'");
```

expr 스크립트 내에서 바인딩 된 Grid 오브젝트, Dataset 컴포넌트, Cell 오브젝트를 지정하고자 할 때는 아래와 같은 변수 또는 지시자를 사용합니다.

> Grid: comp
> Dataset: dataset
> Cell: this

```
<Cell text="expr:this.col"/> <!-- Cell -->
<Cell text="expr:dataset.rowcount"/> <!-- Binded Dataset -->
<Cell text="expr:comp.currentcell"/> <!-- Grid -->
<Cell text="expr:dataset.parent.func1()"/> <!-- Form -->
<Cell text="expr:comp.parent.func1()"/> <!-- Form -->
```

Form에 대한 지시자를 별도로 제공하지 않으며 필요하면 comp.parent 또는 dataset.parent 와 같이 접근합니다.

Form 내에 있는 함수에 접근할 때 comp.parent.func() 형식을 사용하지 않고 this를 사용하게 되면 오류로 처리됩니다. expr 스크립트에서 this는 Cell 오브젝트를 가리킵니다.
<Cell text="expr:this.func01()"/>

컴포넌트 속성으로 Expr과 Text가 같이 있는 경우에 화면에 보이는 텍스트를 반환하는 getDisplayText 메소드를 사용할 수 있습니다.

```
this.Button00.set_text("text");
this.Button00.set_expr("1+1");

trace(this.Button00.text); // "text"
trace(this.Button00.expr); // "1+1"
trace(this.Button00.getDisplayText()); // "2"
```

8.1.5 lookup

lookup 메소드는 유효범위를 지정해 접근하기 어려운 경우 사용할 수 있도록 설계된 추가 메소드입니다. 원하는 오브젝트나 함수를 id 또는 함수명을 가지고 찾을 수 있습니다.

lookup 메소드는 아래와 같은 형식으로 사용할 수 있습니다.

```
Form.lookup( strid );
nexacro.getApplication().lookup( strid );
[Component].addEventHandlerLookup( eventid, funcid, target );
```

실제 코드에서는 아래와 같이 사용됩니다.

```
// this에서 상위로 검색해서 objectid에 해당하는 오브젝트를 반환
var obj = this.lookup("objectid");

// this에서 상위로 검색해서 fn_onclick에 해당하는 함수를 반환
this.lookup("fn_onclick")();

// this에서 상위로 검색해서 fn_onclick에 해당하는 함수를 이벤트 핸들러에서 처리
```

```
btn00.addEventHandlerLookup( "onclick", "fn_onclick", this );
```

8.2 이벤트 핸들러

이벤트 처리를 위한 이벤트 핸들러는 넥사크로 스튜디오 속성창에서 추가하거나 각 컴포넌트, 오브젝트에 제공하는 메소드를 사용해 스크립트에서 추가할 수 있습니다.

8.2.1 메소드

스크립트 내에서 이벤트 처리를 위해 이벤트 함수를 추가, 설정하거나 삭제할 수 있는 메소드를 제공합니다.

```
addEventHandler( eventid, funcobj, target )
addEventHandlerLookup( eventid, funcstr, target )
setEventHandler( eventid, funcobj, target )
setEventHandlerLookup( eventid, functstr, target )
removeEventHandler( eventid, funcobj, target )
removeEventHandlerLookup( eventid, funcstr, target )
```

이벤트를 처리할 함수의 유효범위에 따라 적절한 메소드를 선택해 사용합니다.

```
this.btn00.setEventHandler("onclick", this.fn_onclick, this);
this.dataset00.addEventHandlerLookup("onrowposchange", "fn_onchange", this);
```

addEventHandlerLookup, setEventHandlerLookup, removeEventHandlerLookup 메소드는 앱 성능에 영향을 미칠 수 있으므로 필요한 경우만 사용을 권장합니다.

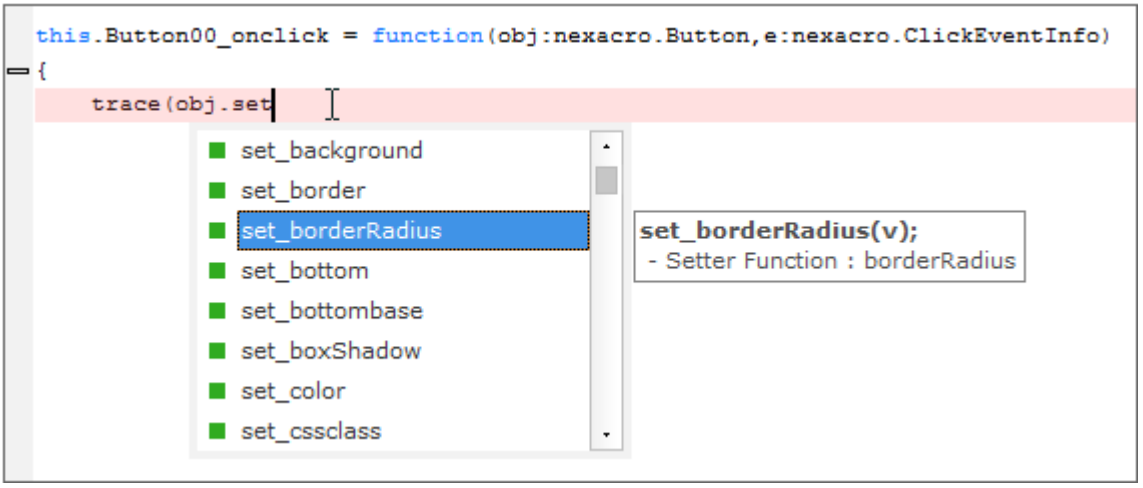
이벤트를 속성 창에서 생성하게 되면 아래와 같이 문자열로 설정됩니다.
<Button id="Button00" ... onclick="Button00_onclick"/>
해당 코드는 빌드 작업 후 변환된 자바스크립트 코드에서는 아래와 같이 this 지시자를 붙여서 생성됩니다.
this.Button00.addEventHandler("onclick", this.Button00_onclick, this);

8.2.2 오브젝트 타입

넥사크로 스튜디오에서 이벤트 핸들러 함수에 매개변수를 설정할 때 오브젝트의 타입을 지정할 수 있습니다. 타입은 자바스크립트 표준 문법은 아니며 넥사크로 스튜디오 내부에서 개발 편의성을 제공하기 위해 지원되는 형식입니다.

```
this.Button00_onclick = function(obj:nexacro.Button,e:nexacro.ClickEventInfo)
{
    trace(obj.text);
}
```

입력된 타입 값을 기준으로 넥사크로 스튜디오에서 인텔리센스 기능을 지원합니다.



해당 코드를 자바스크립트 코드로 변환할 때는 타입 값을 제거합니다.

```
this.Button00_onclick = function(obj,  e)
{
    trace(obj.text);
}
```

> 오브젝트 타입을 지정하지 않아도 앱이 실행하는데 영향을 미치지 않습니다.

> 이벤트 핸들러 함수 외 다른 함수에서는 오브젝트 타입을 지원하지 않으며 동작에 영향을 미칠 수 있습니다.

8.3 Setter

ECMAScript 5부터 지원하는 Setter/Getter와 별개로 넥사크로에서 set 메소드를 제공합니다.

8.3.1 set 메소드

예를 들어 Button 컴포넌트의 text 속성을 사용할 때는 아래와 같이 사용할 수 있습니다.

```
this.Button00.set_text("text");
trace(this.Button00.text);
```

아래와 같이 속성값에 직접 접근하는 방식은 더는 지원하지 않습니다.

```
this.Button00.text = "text";
```

사용자가 직접 추가한 오브젝트 속성에 접근할 때는 기존처럼 접근할 수 있습니다.

```
<Button myprop="333">

this.Button00.myprop = "333";
this.Button00.mytext = "text";
```

스타일 관련 속성에 접근하는 방식도 set 메소드를 사용합니다. 추가된 set 메소드에는 하위 속성까지 포함하고 있습니다. 기존 속성은 값을 가져올 때만 사용할 수 있습니다.

```
this.Button00.set_color('aqua');
trace(this.Button00.color); // 'aqua'
```

set 메소드를 호출하기 전에 스타일 속성값을 가져오려 하면 null 값이 나올 수 있습니다.

8.3.2 동적인 속성

속성값을 가져오는 것은 기존과 같지만, 예외적으로 동적으로 속성값이 변경되는 오브젝트는 별도의 GetMethod를 제공합니다. System 오브젝트의 curx, cury, screenwidth, screenheight 4개 속성은 추가된 메소드를 사용해야 합니다.

```
nexacro.System.getCursorX();
nexacro.System.getCursorY();
nexacro.System.getScreenWidth();
nexacro.System.getScreenHeight();
```

컴포넌트 타입에 따라 사용하지 않는 속성은 접근할 수 없습니다. 예를 들어 Calendar 컴포넌트의 타입이 spin이 아닌 경우에는 아래 속성에 접근할 수 없습니다.

```
this.Calendar.spindownbutton
this.Calendar.spinupbutton
```

Calendar 컴포넌트에서 spindownbutton 속성은 컨트롤 속성(Control Property)이라고 하며 속성 자체를 문자열로 출력하면 [object ButtonControl]라고 표시됩니다.

8.4 기타 변경 사항

8.4.1 nexacro 메소드

ECMAScript에서 지원하지 않는 기능을 넥사크로 자체적으로 수정해 사용하던 메소드는 지원하지 않습니다. 기존에 제공하던 기능은 nexacro 메소드로 별도 제공합니다.

예를 들어 Math 오브젝트에서 제공하던 메소드 중 2개의 인자를 지원하는 floor, ceil, round 메소드는 더는 제공되지 않습니다. 해당 메소드를 사용하려면 nexacro 메소드로 사용해야 합니다.

```
//Math.floor( v, digit );
nexacro.floor( v, digit );

//Math.ceil( v, digit );
```

```
nexacro.ceil( v, digit );

//Math.round( v, digit );
nexacro.round( v, digit );
```

또한, 자바스크립트에서 예약어로 사용하는 일부 오브젝트의 명칭이 변경되었습니다.

```
//new Image();
new nexacro.Image();
```

예약어와 중복되지 않는 나머지 컴포넌트도 TypeDefinition 내의 classname이 nexacro.Button과 같은 형식으로 변경되었습니다. 하지만 기존처럼 Button을 그대로 사용할 수 있습니다.

```
this.button00 = new Button();
or
this.button00 = new nexacro.Button();
```

> 속성이나 오브젝트도 예약어와 충돌되어 일부 변경되었습니다.
>
> Component.class○→ Component.cssclass
> ExcelExportObject.export() → ExcelExportObject.exportData()
> VirtualFile.delete() → VirtualFile.remove()

8.4.2 동작 방식 변경

자바스크립트에서 지원하지 않거나 다르게 동작하는 일부 항목이 수정되었습니다.

<> 비교 연산자 지원하지 않음

<> 비교 연산자를 더는 지원하지 않습니다. 다른 값을 비교할 때는 != 연산자를 사용하세요.

switch 문 내 문자열 처리 방식 변경

이전 버전에서는 switch 문 내에서 case "2" 와 case 2 가 같은 방식으로 처리되던 것을 별개의 값으로 처리합니다.

정규표현식 /g 옵션 적용 방식 변경

정규표현식에서 /g 옵션을 사용하지 않고 replace를 적용하게 되면 한 개의 항목만 변경되며 /g 옵션을 적용해야 전체 항목이 변경됩니다.

8.4.3 오브젝트 명 생성 시 제약

컨테이너의 멤버인 속성, 메소드명과 같은 ChildName을 만들 수 없습니다. 예를 들어 Form은 text라는 속성이 있는데 추가된 버튼 컴포넌트의 id를 text로 지정할 수 없습니다. 아래 같은 경우 버튼이 생성되지 않습니다.

```
<Form text="formtext">
    <Layouts>
        <Layout>
            <Button id="text">
```

Dataset과 같은 Invisible 오브젝트 역시 Array의 멤버로 처리되어 length와 같은 속성을 id로 지정할 수 없습니다. 아래 같은 경우 컬럼이 생성되지 않습니다.

```
<Objects>
    <Dataset id="Dataset00">
        <ColumnInfo>
            <Column id="length" type="STRING" size="256"/>
        </ColumnInfo>
```

8.4.4 변수명, 함수명 생성 시 제약

> 변수명이나 함수명 앞에 언더바(_)를 포함하는 경우 넥사크로 라이브러리에서 사용하는 변수나 함수와 충돌할 수 있습니다. 이로 인해 화면이 보이지 않거나 의도와 다른 방향으로 앱이 동작할 수 있습니다.

예를 들어 아래의 경우 로딩이 완료되지 않고 화면이 보이지 않습니다. 넥사크로 라이브러리에서 사용하는 _is_loading 변수와 같은 이름으로 변수명을 만들었습니다.

```
this._is_loading = true;
this.Button00_onclick = function(obj:Button,  e:nexacro.ClickEventInfo)
{
    trace(this._is_loading);
}
```

> 언더바(_)를 포함하지 않더라도 넥사크로 라이브러리에서 사용하는 일부 변수명이나 함수명과 충돌할 수 있습니다.

예를 들어 아래의 경우 컴포넌트 생성이 완료되지 않고 화면이 보이지 않습니다. 넥사크로 라이브러리에서 사용하는 createComponent 함수와 같은 이름으로 함수명을 만들었습니다.

```
this.createComponent = function()
{
    trace('createComponent ');
}
```

8.4.5 추가 모듈에서 window 오브젝트 사용 시 제약

> 외부 라이브러리 또는 자바스크립트 파일을 모듈로 등록해 사용하는 경우 window 오브젝트가 포함된 코드는 프로젝트 로딩 시 넥사크로 스튜디오에서 에러가 발생할 수 있습니다.

넥사크로 스튜디오에서는 프로젝트를 로딩하면서 모듈로 등록된 코드를 분석하고 처리하는데 이 과정에서 NRE에서 지원하지 않는 window 오브젝트를 처리하면서 에러가 발생합니다. window 오브젝트를 처리하지 않도록 아래와 같이 코드 상단에 분기처리문를 추가하면 에러가 발생하지 않습니다.

```
if( system.navigatorname != "nexacro DesignMode")
{
    if (typeof JSON !== 'object') {
        JSON = {};
    }

    (function () {
...
        })  (window)
    };
```

9.

Frame Tree

넥사크로는 하나의 Form만 가지는 간단한 구조로 화면을 구성할 수 있고 여러 기능을 복합적으로 수행하는 형태의 구조로 구성할 수도 있습니다. 이런 복잡한 구조에서는 각 화면을 구성하는 요소가 서로 어떻게 연결되는지 이해하는 것이 필요합니다.

이번 장에서는 이런 연결 구조에 대한 개념을 몇 가지 예제를 통해 살펴봅니다.

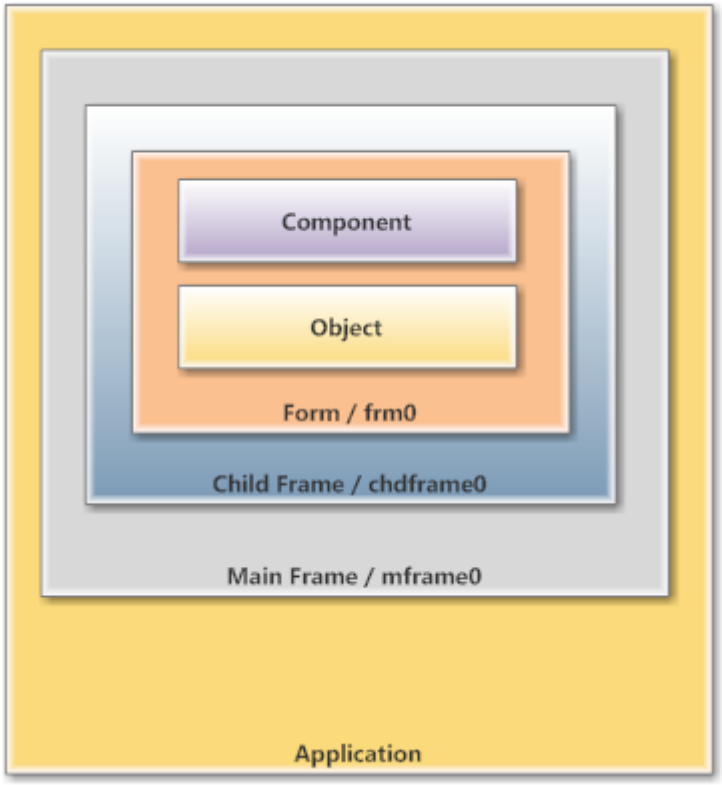
9.1 Application

```
// application -> mainframe
this.mainframe
this.id
this.all[n]
this.all['id']

// mainframe -> application
this.mainframe.parent

// mainframe -> childframe
this.mainframe.frame
this.mainframe.all[n]
this.mainframe.all['id']
this.mainframe.id

// childframe -> mainframe
this.mainframe.frame.parent
this.mainframe.frame.getOwnerFrame()

// childframe -> form
```

```
this.mainframe.frame.form

// form -> childframe
this.parent
this.getOwnerFrame()
nexacro.getApplication().mainframe.frame
```

form내의 component / bind / invisible object는 .getOwnerFrame()이나 .getOwnerForm()이 없습니다.

9.1.1 Script 예시 화면



9.1.2 application에서 form 오브젝트 접근

```
// frm0로의 접근
this.mainframe.frame.form.name =="frm0"
this.mframe0.frame.form.name == "frm0"
```

```
this.mframe0.chdframe0.form.name == "frm0"
this.all[0].all[0].form.name == "frm0"
this.all["mframe0"].all["chdframe0"].form.name == "frm0"
this.mainframe.frame.form.name == "frm0"
```

9.1.3 form에서 상위 오브젝트 접근

```
// frm0에서 application으로 접근
this.parent.name == "chdframe0"
this.parent.parent.name == "mframe0"
this.getOwnerFrame().getOwnerFrame().name == "mframe0"
nexacro.getApplication().mainframe.name == "mframe0"
```

9.2 Form

하나의 Form과 Object, Bind, Component의 관계는 1:N 구조입니다. 스크립트 코드상에서는 아래와 같은 형식으로 접근할 수 있습니다.

```
// Form -> Object
this.all[n]
this.all['id']
this.id
this.objects[n]
this.objects['id']

// Form -> Bind
this.all[n]
this.all['id']
this.id
this.binds[n]
this.binds['id']

// Form -> Component
this.all[n]
```

```
this.all['id']
this.id
this.components[n]
this.components['id']

// Object, Bind, Component -> Bind
this.all[n].parent
```

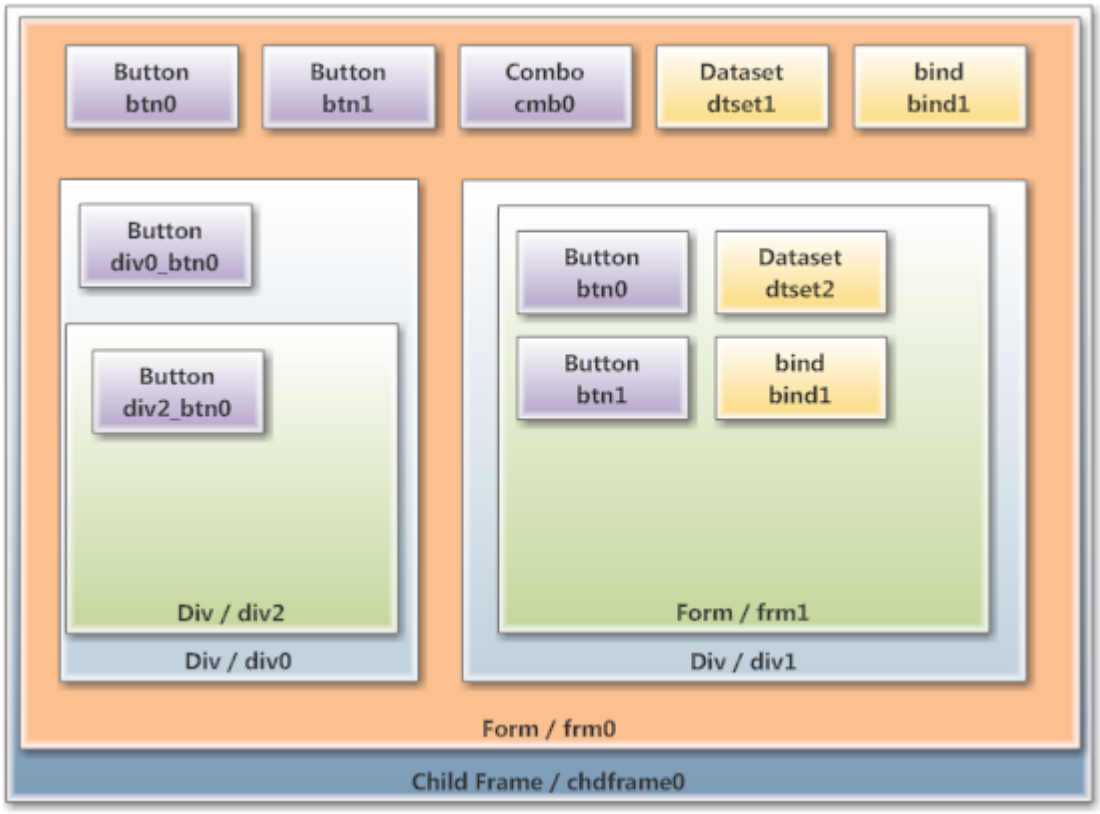
Div, PopupDiv, Tabpage와 같은 컨테이너 컴포넌트는 `url` 속성을 지정해 Form을 연결할 수 있습니다. 이런 경우 컨테이너 컴포넌트와 Form은 1:1 구조입니다. 스크립트 상에서 Form에 연결된 Object, Bind, Component는 아래와 같은 형식으로 접근할 수 있습니다.

```
// Container Component -> Object
this.Div.form.all[n]
this.Div.form.all['id']
this.Div.form.id
this.Div.form.objects[n]
this.Div.form.objects['id']

// Container Component -> Bind
this.Div.form.all[n]
this.Div.form.all['id']
this.Div.form.id
this.Div.form.binds[n]
this.Div.form.binds['id']

// Container Component -> Component
this.Div.form.all[n]
this.Div.form.all['id']
this.Div.form.id
this.Div.form.components[n]
this.Div.form.components['id']

// Object, Bind, Component -> Container Component
this.Div.form.all[n].parent.parent
```

`all`, `objects`, `binds`, `components` 속성은 `nexacro.Collection` 형식으로 제공됩니다. 컬렉션 오브젝트 내 인덱스 또는 id로 접근할 수 있습니다.

all 속성으로 접근하는 컬렉션 오브젝트 내 인덱스는 다음과 같은 순서로 지정됩니다.
1. Object
2. Component
3. Bind
그리고 같은 항목 내에서는 소스 코드 내 순서에 따라 지정됩니다.

9.2.1 Script 예시 화면



인덱스 순서는 dtset1, btn0, btn1, cmb0, div0, div1, bind1으로 가정합니다.
일반적인 폼에서 인덱스 순서는 Dataset이 가장 먼저 오고 나머지 컴포넌트는 배치된 순서대로 정해지며 Bind 오브젝트가 마지막에 지정됩니다.

한 form에서 같은 level에서만 id가 중복될 수 없습니다. 즉, btn0와 div0_btn0와 div2_btn은 동일 Level이 아니므로, 같은 id로 지정하여도 무방합니다.

9.2.2 component / object / bind

```
this.name == "frm0"
```

```
// btn0으로의 접근
this.all["btn0"].name == "btn0"
this.all[1].name == "btn0"
this.btn0.name == "btn0"
this.components[0].name == "btn0"
this.components["btn0"].name == "btn0"
```

```
// dtset1으로의 접근
this.all["dtset1"].name == "dtset1"
this.all[0].name == "dtset1"
this.dtset1.name == "dtset1"
this.objects[0].name == "dtset1"
this.objects["dtset1"].name == "dtset1"
```

```
// component / invisible object / bind의 개수
this.all.length == 7
this.components.length == 5 //btn0, btn1, comb0, div0, div1
this.binds.length == 1
```

9.2.3 container component

form을 연결하지 않은 container component도 동적으로 invisible object / bind를 가질 수 있습니다.

```
// div0내의 div0_btn0으로의 접근
this.div0.form.div0_btn0.name == "div0_btn0"
this.div0.form.all[0].name == "div0_btn0"
this.components[3].form.components[0].name == "div0_btn0"
```

```
// div2내의 div2_btn0으로의 접근
this.div0.form.div2.form.div2_btn0.name == "div2_btn0"
this.all["div0"].form.all["div2"].form.all["div2_btn0"].name == "div2_btn0"
```

```
this.components["div0"].form.components["div2"].form.components["div2_btn0"].name == "div2_btn
0"
```

9.2.4 form을 연결한 container component

```
// (frm1안의 script인 경우) frm1에 있는 btn0의 접근
this.parent.name == "div1"
this.btn0.name == "btn0"
this.all["btn0"].name == "btn0"
```

```
// (frm0안의 script에서 접근할 경우) frm1에 있는 btn0의 접근
this.name == "frm0"
this.div1.form.btn0.name == "btn0"
```

9.2.5 parent

```
this.div1.form.dtset2.parent.parent.name == "form" // frm0이 아닙니다.
this.div1.form.dtset2.parent.name == "div1"
```

파트 III.

참고

10.

앱 캐시

네트워크를 효율적으로 사용하기 위해 개발된 앱을 배포서버에서 매번 내려받지 않고 캐시 기술을 사용합니다.

10.1 캐시의 종류

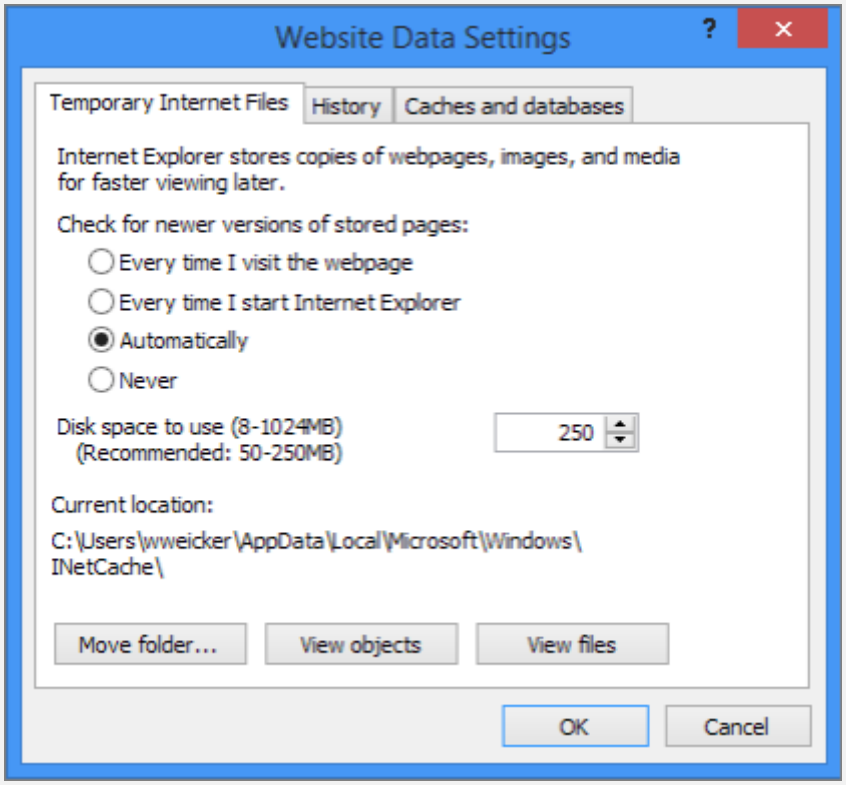
지원하는 캐시의 종류는 아래와 같습니다.

캐시 상태	설명
none	캐시 기능을 사용하지 않습니다.
dynamic	서버로부터 내려받은 파일이 갱신되었을 때만 수신합니다. 서버상의 파일이 갱신되지 않은 경우는 로컬 캐시(Local Cache) 파일을 재사용합니다.
session	엔진을 기동할 때, 단 한 번만 수신하고 해당 엔진이 종료될 때까지 로컬 캐시 파일만을 사용합니다. (엔진 기동 시 로컬 캐시 파일과 일치하면 재사용합니다.)
static	서버로부터 한 번이라도 내려받은 파일은 엔진 재실행과 상관없이 로컬 캐시 파일만 사용합니다. (단, Type Definition에 지정된 서비스 그룹의 버전이 변경된 경우에는 다시 내려받습니다.)

속성값이 static이면 캐시 파일을 로컬저장소에 저장하게 되는데 웹브라우져의 기능 제약으로 로컬저장소를 사용하지 못합니다. 그래서 웹브라우저에서 사용하는 경우에는 속성값이 static인 경우에도 session으로 동작합니다.

웹브라우저에서 사용하는 경우에는 속성값이 none이면 브라우저에서 캐시를 사용하지 않는 것이 정상입니다. Firefox를 사용하는 경우에는 브라우저 캐시를 사용하는 증상이 있는데 이 증상은 Firefox 버그입니다.
https://bugzilla.mozilla.org/show_bug.cgi?id=1129500

MS 인터넷 익스플로러에서 실행되는 경우 넥사크로 내부에서 이용하는 캐시메모리보다 인터넷 익스플로러의 캐시가 우선 동작하는 경우가 있습니다. 웹브라우저 캐시옵션이 "자동으로 확인(default)"일 때 서버에서 변경된 자바스크립트 파일(*.js)이 갱신되지 않는 현상은 제약사항입니다.



웹브라우저를 새로 실행하거나 새로고침하는 경우에는 제품에서 관리하는 메모리상의 캐시가 모두 제거되기 때문에 속성값을 session, dynamic, static으로 설정한 경우 설정한 값과 무관하게 브라우저 동작에 의존하게 됩니다.

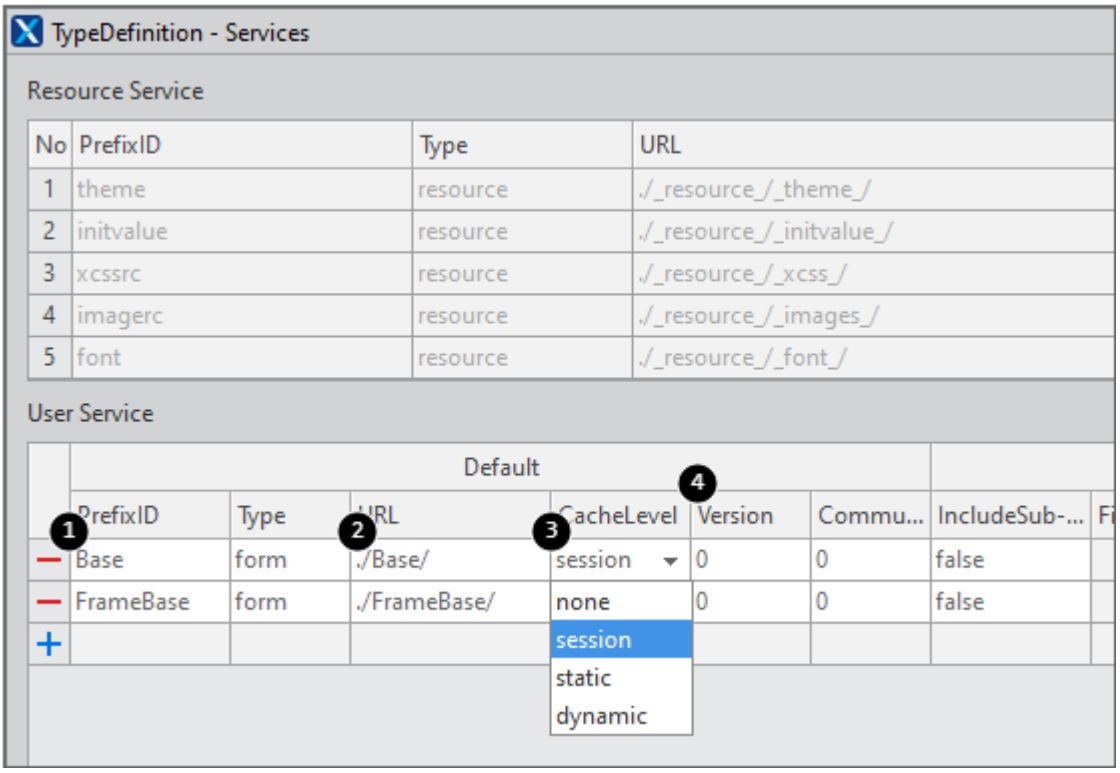
넥사크로 스튜디오에서 [Launch Project] 또는 [Quick View] 메뉴를 사용해 NRE를 실행하는 경우에는 typedefinition에 설정된 Service-cachelevel 값을 무시하고 none으로 동작합니다.
(cachelevel 속성의 설정값이 변경되는 것은 아닙니다)

넥사크로에서 이미지를 처리할 때는 이미지 표현 관련 기능(size, stretch 등)과 성능상의 이유로 사용 브라우저의 캐시 설정에 따른 동작을 하도록 설계됐습니다.
(프로젝트 service 그룹에 설정된 cachelevel과는 별개의 동작입니다)

> ExcelImportObject 또는 ExcelExportObject를 사용해 엑셀 파일을 가져오거나 내보낼 때에는 지정한 cachelevel 속성값과 관계없이 무조건 none으로 동작합니다.

10.2 캐시 적용 방법

다음은 Type Definition 편집기에 대한 설명입니다.



1. 캐시는 등록된 서비스 그룹에만 적용됩니다.
2. "./Base/' 폴더 아래 등록된 모든 파일에 대해 캐시가 적용됩니다.
3. 캐시의 종류를 선택합니다.
4. 서비스 버전을 설정하면 Static 상태에서 파일을 다시 내려받을 수 있습니다.

```
nexacro.getEnvironment().services["Base"].set_cachelevel("none" )
```

11.

Dataset XML Format

넥사크로는 서버모듈로 X-API를 제공합니다. 이 API를 사용하여 생성된 Dataset format을 넥사크로가 해석해 앱을 화면에 보여줍니다.

여기서는 X-API가 생성하는 Dataset format을 설명합니다. 이 format을 준수한 XML 파일은 넥사크로가 바로 적용할 수 있는 데이터로 간주해 사용할 수 있습니다. 즉, X-API를 사용하지 않아도 서버 프로그램을 개발할 수 있습니다.

11.1 Dataset XML layout

<table>
<tr><td colspan="6">XML Declaration</td></tr>
<tr><td colspan="6">Root</td></tr>
<tr><td rowspan="3"></td><td colspan="5">Cache</td></tr>
<tr><td colspan="5">Parameters</td></tr>
<tr><td></td><td colspan="4">Parameter (반복)</td></tr>
<tr><td rowspan="9"></td><td colspan="5">Dataset (반복)</td></tr>
<tr><td rowspan="3"></td><td colspan="4">ColumnInfo</td></tr>
<tr><td></td><td colspan="3">ConstColumn (반복)</td></tr>
<tr><td></td><td colspan="3">Column (반복)</td></tr>
<tr><td rowspan="5"></td><td colspan="4">Rows</td></tr>
<tr><td></td><td colspan="3">Row (반복)</td></tr>
<tr><td rowspan="3"></td><td colspan="2">Col (반복)</td></tr>
<tr><td colspan="2">OrgRow</td></tr>
<tr><td></td><td>Col (반복)</td></tr>
</table>

11.1.1 XML 선언

XML임을 명시하는 XML 선언문을 다음과 같이 정의합니다.

```
<?xml version="1.0" encoding="utf-8"?>
```

XML 선언문은 XML 문서의 가장 앞에 있어야 하며, 이 선언문 앞에는 어떠한 공백문자를 포함한 어떠한 문자도 올 수 없습니다.

11.1.2 XML 예

```
<?xml version="1.0" encoding="utf-8"?>
<Root xmlns="http://www.nexacroplatform.com/platform/dataset" ver="4000" >
    <Parameters>
        <Parameter id="service">stock</Parameter>
        <Parameter id="method">search</Parameter>
    </Parameters>
    <Dataset id="output">
        <ColumnInfo>
            <ConstColumn id="market" size="10" type="STRING" value="kse"/>
            <ConstColumn id="openprice" size="10" type="INT"
            value="15000"/>
            <Column id="stockCode" size="5" type="STRING"/>
            <Column id="currentprice" size="10" type="INT"/>
        </ColumnInfo>
        <Rows>
            <Row>
                <Col id="currentCode">10001</Col>
                <Col id="currentprice">5700</Col>
            </Row>
            <Row>
                <Col id="currentCode">10002</Col>
                <Col id="currentprice">14500</Col>
            </Row>
        </Rows>
    </Dataset>
</Root>
```

11.2 Dataset 요소

Dataset의 각 요소(element)들을 설명합니다.

11.2.1 Root

개요

Dataset을 나타내는 최상위 요소

자식 요소

Parameters, Dataset

반복 여부

반드시 1개가 있어야 합니다.

속성

```
<Root xmlns="http://www.nexacroplatform.com/platform/dataset">
<Root xmlns:nexacro="http://www.nexacroplatform.com/platform/dataset">
<Root xmlns="" version="1000">
```

속성 이름	설명
xmlns	네임스페이스, 만약 기본 네임스페이스를 사용하지 않는 경우라면 xmlns:nexacro로 명시합니다.
ver	Dataset Layout의 버전을 명시합니다.

11.2.2 Parameters

개요

통신 시 필요한 파라미터를 명시하기 위한 요소. 실제 파라미터값은 자식 요소인 Parameter가 가지고 있으며, Parameters 요소는 이들을 가지고 있는 집합의 의미입니다.

자식 요소

Parameter

반복 여부

없거나 1개가 있을 수 있습니다.

11.2.3 Parameters > Parameter

개요

파라미터의 값을 명시합니다.

자식 요소

없다.

반복 여부

없거나 Parameter 요소는 파라미터 개수만큼 반복될 수 있습니다.

속성

```
<Parameter id="ErrorMsg" type="STRING">SUCC</ Parameter >
```

속성 이름	설명
id	파라미터 이름
type	파라미터값의 타입

비고

넥사크로에서는 다음과 같은 필수 파라미터를 사용하도록 정의하고 있습니다. 넥사크로와의 연동을 위해서는 파라미터를 설정하여야 합니다.

```
<Parameter id="ErrorCode">1</Parameter>
<Parameter id="ErrorMsg">SUCC</Parameter>
<Parameter id="cachetype">Session</Parameter>
```

id	설명
ErrorCode	Error Code. -Transaction 함수 호출 시 0보다 작은 경우에는 사용자가 정의한 에러로 InputDataset에 UpdateStatus를 반영하지 않고 실패 처리됩니다. 0 이상일 경우에는 사용자가 정의한 정상 상태로 InputDataset에 UpdateStatus를 반영하며 성공 처리됩니다. -Load 함수 호출 시

id	설명
	0보다 작은 경우에는 사용자가 정의한 에러로 실패 처리됩니다. 0 이상일 경우에는 사용자가 정의한 정상 상태로 성공 처리됩니다. 미지정 시 : 0
ErrorMsg	Error Message 미지정 시 : ErrorCode가 미지정 또는 0일 때 SUCCESS, 그 외는 FAILED
CacheType	Cache 방법의 Type을 지정한다. Type은 Session, Dynamic, Static, None 로 구분됩니다. 미지정 시 : Session

type에 올 수 있는 데이터 타입은 다음과 같습니다.

- STRING: 문자열. 컬럼의 크기와 관계없이 길이 제한이 없습니다. (단 시스템에서 한 번에 할당 가능한 크기인 2 GB 정도 제한됩니다)
  - 빈 문자열인 경우
    <Parameter id="ErrorMsg" type="STRING"></Parameter>
  - Null인 경우
    <Parameter id="ErrorMsg" type="STRING"/>로 구분한다.
- INT: 정수(-2^{31} ~ $2^{31}-1$)
- FLOAT, DECIMAL: ($\pm 2.2X10^{-308}$ ~$\pm 1.7X10^{308}$)
- BIGDECIMAL: 실수($\pm 10^{-1056}$ ~ $\pm 10^{1056}$)
- DATE: YYYYMMDD(-8192/01/01 ~ 8191/12/31)
- DATETIME : YYYYMMDDHHmmssuuu (msec포함가능)
- TIME : 6자리 HHmmssuuu(msec포함가능)
- BLOB: 이진 데이터

11.2.4 Dataset

개요

하나의 Dataset의 내용의 값을 가지고 있는 요소. 자식 요소에는 Dataset의 구조를 표현하기 위한 ColumnInfo와 Dataset의 실제 데이터를 가진 Rows가 있습니다.

자식 요소

ColumnInfo, Rows

반복 여부

없거나 xml이 전달하는 Dataset의 개수만큼 반복될 수 있습니다.

속성

```
<Dataset id="History">
```

속성 이름	설명
id	Dataset 이름

11.2.5 Dataset > ColumnInfo

개요

Dataset의 스키마를 표현하기 위한 요소. 실제 스키마의 정보는 자식 요소인 ConstColumn 과 Column이 표현하고 ColumnInfo는 이들을 포함하는 집합의 의미가 있습니다. 고정된 값의 컬럼의 스키마는 ConstColumn 이 비 고정값의 컬럼의 스키마는 Column이 표현합니다.

자식 요소

ConstColumn, Column

반복 여부

없거나 1개가 있을 수 있습니다.

> ColumnInfo의 자식 엘리먼트들은 ConstColumn, Column 순으로 나열되어야 하며 ConstColumn이 항상 먼저 나열되어야 합니다.
> ConstColumn, Column이 섞여서 나열되는 경우에는 앱에서 데이터를 처리하지 못할 수 있습니다.

```
(X) ConstColumn이 먼저 나열되지 않은 경우
<ColumnInfo>
    <Column id="stockCode" size="5" type="STRING"/>
    <Column id="currentprice" size="10" type="INT"/>
    <ConstColumn id="market" size="10" type="STRING" value="kse"/>
    <ConstColumn id="openprice" size="10" type="INT" value="15000"/>
</ColumnInfo>
...
```

```
(X) ConstColumn, Column이 섞여있는 경우
<ColumnInfo>
    <ConstColumn id="market" size="10" type="STRING" value="kse"/>
    <Column id="currentprice" size="10" type="INT"/>
```

```
    <ConstColumn id="openprice" size="10" type="INT" value="15000"/>
    <Column id="stockCode" size="5" type="STRING"/>
</ColumnInfo>
...
```

11.2.6 Dataset > ColumnInfo > ConstColumn

개요

항상 고정된 값을 갖는 컬럼의 스키마를 표현합니다.

자식 요소

없다.

반복 여부

없거나 고정값 컬럼의 개수만큼 있을 수 있습니다.

속성

```
<ConstColumn id="systemName"
      size="255"
      type="STRING"
      value="Production"/>
```

속성 이름	설명
id	컬럼 이름
size	최대 데이터 크기
type	컬럼의 데이터 타입
value	컬럼의 고정값

비고

type에 올 수 있는 데이터 타입은 다음과 같습니다.

- STRING: 문자열. 컬럼의 크기와 관계없이 길이 제한이 없습니다.(단 시스템에서 한 번에 할당 가능한 크기인 2GB 정도로 제한됩니다)
- INT: 정수(-2^{31} ~ $2^{31}-1$)
- FLOAT, DECIMAL: ($\pm 2.2 \times 10^{-308}$ ~$\pm 1.7 \times 10^{308}$)
- BIGDECIMAL: 실수($\pm 10^{-1056}$ ~ $\pm 10^{1056}$)
- DATE: YYYYMMDD(-8192/01/01 ~ 8191/12/31)

- DATETIME : YYYYMMDDHHmmssuuu (msec포함가능)
- TIME : 6자리 HHmmssuuu(msec포함가능)
- BLOB: 이진 데이터

11.2.7 Dataset > ColumnInfo > Column

개요

파라미터의 값을 명시합니다.

자식 요소

없다.

반복 여부

없거나 비 고정값 컬럼의 개수만큼 있을 수 있습니다.

속성

```
<Column id="department"
      size="255"
      type="STRING"
      prop="text"
      sumtext="소계"/>
```

속성 이름	설명
id	컬럼 이름
size	최대 데이터 크기
type	컬럼의 데이터 타입
Prop	Summury시 type을 정의
Sumtext	porp값이 text일 때 문자열

비고

type에 설정되는 값은 ColumnInfo에서의 type과 같습니다.

Prop은 다음과 같은 값을 갖습니다.

- count : 컬럼의 row 개수를 summary 표시합니다.
- sum: 컬럼의 row값의 합을 summary에 표시합니다.
- max: 컬럼의 row들 중 Max값을 summary에 표시합니다.
- min: 컬럼의 row들 중 Min값을 summary에 표시합니다.

- avg: 컬럼의 row값의 평균을 summary에 표시합니다.
- Text: summary에 sumtext 속성값을 표시합니다.

11.2.8 Dataset > Rows

개요

Dataset의 각 Row를 포함하기 위한 집합 의미의 요소. 실제 데이터를 표현하는 Row를 자식 요소로 갖습니다.

자식 요소

Row

반복 여부

없거나 1개 있을 수 있습니다.

11.2.9 Dataset > Rows > Row

개요

Dataset의 하나의 Row의 데이터를 표현하기 위한 요소. 실제 각 컬럼의 값을 표현하는 Col을 자식 요소로 갖습니다. 그리고 type이 “update"일 경우 변경되기 전의 Row의 값을 가지고 있는 Org_Row 자식 요소가 있습니다.

자식 요소

Col, Org_Row

반복 여부

없거나 Dataset의 Row 개수만큼 있을 수 있습니다.

속성

```
<Row type="insert">
    <Col id="currentCode">10001</Col>
    <Col id="currentprice">13400</Col>
</Row>

<Rows>
    <Row type="update">
        <Col id="currentCode">10001</Col>
        <Col id="currentprice">13400</Col>
```

```
        <OrgRow>
            <Col id="currentCode">10001</Col>
            <Col id="currentprice">13700</Col>
        </OrgRow>
    </Row>
</Rows>

<Row type="delete">
    <Col id="currentCode">10001</Col>
    <Col id="currentprice">13400</Col>
</Row>
```

속성 이름	설명
type	insert : 원본 Dataset에 추가된 Row.
	update: 원본 Dataset에 변경된 Row. 자식 요소로 Org_Row를 포함합니다. Org_Row는 변경 전의 원본 Row입니다.
	delete: 원본 Dataset에 삭제된 Row를 의미합니다.

11.2.10 Dataset > Rows > Row > Col

개요

Dataset의 각 컬럼값을 표현합니다.

자식 요소

없습니다.

반복 여부

없거나 컬럼의 개수만큼 있을 수 있습니다.

속성

```
<Col id="department">management</Col>
```

속성이름	설명
Id	컬럼 이름. ColumnInfo 의 자식 요소(ConstColumn , Column)에서 설정한 이름과 같습니다.

비고

빈 문자열인 경우

```
<Col id="department"></Col>
or
<Col id="department"/>
```

Null인 경우 Tag가 없는 것으로 구분합니다.

11.2.11 Dataset > Rows > Row > OrgRow

개요

Dataset의 Row의 값이 변경되었을 때, 원래 값을 가지고 있는 요소. 자식 요소로 실제 값을 가지고 있는 Col이 있습니다.

자식 요소

Col

반복 여부

없거나 1개 있을 수 있습니다.

비고

부모 요소인 Row와 유사하나 속성이 없습니다.

11.2.12 Dataset > Rows > Row > OrgRow > Col

개요

변경되기 전의 Column 값을 표현합니다.

자식 요소

없습니다.

반복 여부

없거나 컬럼의 개수만큼 있을 수 있습니다.

속성

```
<Col id="department">management</Col>
```

속성 이름	설명
id	컬럼 이름. ColumnInfo 의 자식 요소(ConstColumn , Column)에서 설정한 이름과 같습니다.

12.

Dataset SSV Format

Dataset 오브젝트 데이터를 SSV(Simple Separated Values) 형태로 처리하기 위한 레이아웃을 설명합니다.

> 본문에서 주요 용어는 아래와 같이 줄여서 표시합니다.
>
> - (RS): RS(Record Seperator) / 0x1E(30)
> - (US): US(Unit Separator) / 0x1F(31)

> 운영 환경에 따라 (RS), (US)는 넥사크로 스튜디오에서 다른 코드로 변경할 수 있습니다.
>
> - Environment.ssvrecordseparator
> - Environment.ssvunitseparator

12.1 Dataset SSV layout

Dataset SSV는 아래와 같은 형식으로 구성합니다.

항목	필수
Stream Header	O
Variables	X
Datasets	X

12.1.1 Stream Header

```
SSV{:CodePage}(RS)
```

Stream Header는 "SSV"로 시작하고 필요한 경우 콜론(:) 다음에 CodePage 값을 추가할 수 있습니다. (RS)로 다음 항목(Variables 또는 Datasets)과 구분합니다.

항목	필수	설명
SSV	O	"SSV"로 고정된 값입니다.
CodePage	X	":" 문자로 시작하며 인코딩 정보입니다.

```
SSV(RS)
SSV:en_US(RS)
SSV:utf-8(RS)
```

12.1.2 Variables

생략하거나 1개 이상의 Variable을 가질 수 있습니다. n개의 Variable은 (RS)로 구분하며 (RS)로 Datasets와 구분합니다.

Variable

```
Variable ID{:Type(Length)}{=Value}(RS)
Variable ID{:Type}{=Value}(RS)
```

항목	필수	설명
Variable ID	O	고유 ID로 중복된 값이 들어오면 마지막에 들어온 값만 처리합니다.
Type	X	":" 문자로 시작하며 Length 정보를 포함할 수 있습니다. 생략한 경우 "STRING"으로 처리합니다.
Length	X	"("문자와 ")"문자 사이의 값으로 처리합니다. 생략한 경우 Type에 따른 임계치로 처리합니다. 단, Type이 "STRING"인 경우에는 255로 처리합니다.
Value	X	"="문자로 시작하며 (RS)가 나오기 전까지 Value로 처리합니다.

```
name1=value(RS)
name1=value1(RS)name2=value2(RS)
name1:STRING=value1(RS)name2=value2(RS)
name3=value3(RS)name4:STRING(10)=value4(RS)
```

12.1.3 Datasets

생략하거나 1개 이상의 Dataset을 가질 수 있습니다.

Dataset

Dataset은 아래와 같은 형식을 가집니다.

항목	필수
Dataset Header	O
Const Column Infos	X
Column Infos	O
Records	X
Null Record	O

> Const Column Infos가 있을 경우 Column Info보다 먼저 나열해야 합니다.
> 순서가 바뀌거나 Column Info 뒤에 다시 Const Column Infos 정보를 나열하는 경우 정상적인 동작을 보장하지 않습니다.

12.2 Dataset 형식 상세

Dataset의 각 항목을 설명합니다.

12.2.1 Dataset Header

```
Dataset:Dataset ID(RS)
```

Dataset Header는 "Dataset"로 시작하고 콜론(:) 다음에 Dataset ID를 작성합니다. (RS)로 다음 항목과 구분합니다.

12.2.2 Const Column Infos

```
_Const_(US)Const Column ID{:Type(Length)}{=Value}(RS)
_Const_(US)Const Column ID{:Type}{=Value}(RS)
```

"_Const_"로 시작하고 1개 이상의 Const Column Info를 가질 수 있습니다. n개의 Const Column Info는 (US)로 구분하며 (RS)로 다음 항목과 구분합니다.

항목	필수	설명
Const	O	"_Const_"로 고정된 값입니다.
Const Column ID	O	Dataset에서 Const Column ID로 사용할 ID 값입니다.
Type	X	":" 문자로 시작하며 Length 정보를 포함할 수 있습니다. 생략한 경우 "STRING"으로 처리합니다.
Length	X	"("문자와 ")"문자 사이의 값으로 처리합니다. 생략한 경우 Type에 따른 임계치로 처리합니다. 단, Type이 "STRING"인 경우에는 255로 처리합니다.
Value	X	"="문자로 시작하며 (US) 또는 (RS)가 나오기 전까지 Value로 처리합니다.

12.2.3 Column Infos

```
_RowType_(US)Column ID{:Type(Length)}{:Sum Type}{:Sum Text}(RS)
_RowType_(US)Column ID{:Type}{:Sum Type}{:Sum Text}(RS)
```

"_RowType_"로 시작하고 1개 이상의 Column Info를 가질 수 있습니다. n개의 Column Info는 (US)로 구분하며 (RS)로 다음 항목과 구분합니다.

항목	필수	설명
RowType	O	"_RowType_"로 고정된 값입니다.
Column ID	O	Dataset에서 Column ID로 사용할 ID 값입니다.
Type	X	":" 문자로 시작하며 Length 정보를 포함할 수 있습니다. 생략한 경우 "STRING"으로 처리합니다.
Length	X	"("문자와 ")"문자 사이의 값으로 처리합니다. 생략한 경우 Type에 따른 임계치로 처리합니다. 단, Type이 "STRING"인 경우에는 255로 처리합니다.
Sum Type	X	":"문자로 ColumnInfo 오브젝트의 prop 속성값을 설정합니다.
Sum Text	X	":"문자로 ColumnInfo 오브젝트의 sumtext 속성값을 설정합니다.

12.2.4 Records

```
RowType(US)Column Value(RS)
```

첫 번째 필드값은 RowType이고 그 이후 필드값은 Column Infos에서 설정한 Column 순서대로 작성합니다. 각 필드값은 (US)로 구분합니다.

항목	필수	설명
RowType	O	레코드 형태를 작성합니다. N: Normal Record I: Inserted Record U: Updated Record D: Deleted Record O: Original Record (Update Record의 원본 레코드입니다. Update Record가 있는 경우 사용할 수 있습니다).
Column Value	O	Dataset에서 Column Value로 사용할 값입니다. Column Value를 undefined로 처리하고자 한다면 "ETX (End of Text) / 0x03(03)"을 사용합니다.

12.2.5 Null Record

(RS)로 Dataset의 끝을 구분합니다.

12.3 참고

12.3.1 Type

Variable, Const Column Infos, Column Infos에서 Type 항목에 사용할 수 있는 값은 아래와 같습니다.

Type	설명
STRING	문자열. 최대값은 system에서 한번에 할당 가능한 크기인 2GB로 제한됨 Column의 size와 관계없이 최대치까지 사용될 수 있다.
INT	정수형 (-2^{31} ~ $2^{31}-1$)
FLOAT, DECIMAL	실수형 ($\pm 2.2 \times 10^{-308}$ ~ $\pm 1.7 \times 10^{308}$)
BIGDECIMAL	실수형(문자열로 저장) ($\pm 10^{-1056}$ ~ $\pm 10^{+1056}$)

DATE	날짜형 YYYYMMDD (-8192/01/01 ~ 8191/12/31)
DATETIME	YYYYMMDDHHmmssuuu (msec포함가능)
TIME	6자리 HHmmssuuu(msec포함가능)
BLOB	이진 데이터

12.3.2 예제

- (RS), (US)는 아래 기호로 표시했습니다.
  ▼ = Record Seperator(RS)
  • = Unit Seperator(US)
- 쉽게 읽기 위해 줄바꿈을 추가했습니다. 실제 SSV 구성 시에는 줄바꿈을 추가하지 않습니다.

Dataset이 1개 존재하는 경우

```
SSV:utf-8▼
Dataset:dataset0▼
_RowType_•Col1:String(20)•Col2:Int:SUM•Col3:Decimal:AVG▼
N•Test•0•1•1▼
I•Abc•1•2•2▼
U•Def•2•3•3▼
O•Chk•2•3•3▼
D•Ghi•3•4•4▼
▼
```

Const Column이 존재하는 Dataset이 1개 존재하는 경우

```
SSV:utf-8▼
Dataset:dataset0▼
_Const_•ConstCol1:STRING(20)=Name•ConstCol2:INT=1•ConstCol3:DECIMAL=0.8▼
_RowType_•Col1:STRING(20)•Col2:INT:SUM•Col3:DECIMAL:AVG▼
N•Test•0•1•1▼
I•Abc•1•2•2▼
U•Def•2•3•3▼
O•Chk•2•3•3▼
D•Ghi•3•4•4▼
▼
```

Dataset이 여러 개 존재하는 경우

```
SSV:utf-8▼
Dataset:dataset0▼
_Const_ • ConstCol1:STRING(20)=Name • ConstCol2:INT=1 • ConstCol3:DECIMAL=0.8▼
_RowType_ • Col1:STRING(20) • Col2:INT:SUM • Col3:DECIMAL:AVG▼
N • Test • 0 • 1 • 1▼
I • Abc • 1 • 2 • 2▼
U • Def • 2 • 3 • 3▼
O • Chk • 2 • 3 • 3▼
D • Ghi • 3 • 4 • 4▼
▼
Dataset:dataset1▼
_RowType_ • Col1:INT(4):Summ,Col2:STRING(30):Text▼
N • 0 • test▼
I • 1 • test1▼
U • 2 • test3▼
O • 2 • test3-1▼
D • 3 • test4▼
▼
```

Variable만 존재하는 경우

```
SSV:utf-8▼
Var4=10▼Var5=20▼
▼
```

Variable과 Dataset이 존재하는 경우

```
SSV:utf-8▼
Var4=10▼Var5=20▼
Dataset:dataset0▼
_Const_ • ConstCol1:STRING(20)=Name • ConstCol2:INT=1 • ConstCol3:DECIMAL=0.8▼
_RowType_ • Col1:STRING(20) • Col2:INT:SUM • Col3:DECIMAL:AVG▼
N • 0 • test▼
I • 1 • test1▼
U • 2 • test3▼
O • 2 • test3-1▼
D • 3 • test4▼
▼
```

13.

Dataset JSON Format

Dataset 오브젝트 데이터를 JSON(JavaScript Object Notation) 형태로 처리하기 위한 레이아웃을 설명합니다.

13.1 Dataset JSON layout

Dataset JSON는 아래와 같은 형식으로 구성합니다.

항목	필수
version	O
Parameters	X
Datasets	X

13.1.1 version

Dataset Layout 의 버전을 명시합니다.

```
"version" : "1.0",
```

13.1.2 Parameters

파라미터의 값을 명시합니다.

```
"Parameters":
[
    {"id": "ErrorCode", "value":0},
```

```
    {"id": "ErrorMsg", "value":""},
    {"id": "param1", "value":0},
    {"id": "param2", "value":"0", "type":"string"}
]
```

속성명	필수	설명
id	O	파라미터명을 설정합니다. "ErrorCode", "ErrorMsg" 항목은 예약된 키워드입니다. "ErrorCode" 항목을 지정하지 않은 경우 value는 0으로 처리합니다. "ErrorMsg" 항목을 지정하지 않은 경우 "ErrorCode" value가 0이면 "SUCCESS", 그 외의 값이면 "FAILED:로 처리합니다.
value	X	Integer, Float, String
type	X	데이터 타입을 설정합니다. type 항목을 참고하세요.

- "ErrorCode", "ErrorMsg" 항목은 예약된 키워드입니다.
- "ErrorCode" 항목을 지정하지 않은 경우 0으로 처리합니다.
- "ErrorMsg" 항목을 지정하지 않은 경우 "ErrorCode" 항목값이 0이면 "SUCCESS", 그 외의 값이면 "FAILED:로 처리합니다.

13.1.3 Datasets

Dataset 오브젝트 정보를 포함하는 요소로 배열 내 오브젝트 형태로 Dataset을 가지고 있습니다.

Dataset

하나의 Dataset 오브젝트 정보를 가지고 있는 요소로 Dataset 오브젝트의 구조를 표현하기 위한 ColumnInfo와 Dataset 오브젝트의 실제 데이터를 가지고 있는 Rows가 포함됩니다.

항목		필수
id		O
ColumnInfo		O
	ConstColumn	X
	Column	O
Rows		O

13.2 Dataset 형식 상세

Dataset의 각 항목을 설명합니다.

13.2.1 id

```
"id":"inData"
```

속성명	필수	설명
id	O	Dataset 오브젝트 id 속성값입니다.

13.2.2 ColumnInfo > ConstColumn

```
"ColumnInfo":
{
    "ConstColumn":
    [
        {"id":"ConstCol1", "value":10},
        {"id":"ConstCol2", "type":"string", "size":"256", "value":10},
        {"id":"ConstCol3"}
    ]
}
```

속성명	필수	설명
id	O	Dataset 오브젝트에서 Const Column ID로 사용할 ID 값입니다.
type	X	데이터 타입을 설정합니다. type 항목을 참고하세요.
size	X	최대 데이터 크기를 설정합니다. 생략한 경우 Type에 따른 임계치로 처리합니다. 단, type이 "STRING"인 경우에는 255로 처리합니다.
value	X	고정값을 설정합니다. 설정하지 않으면 undefined로 처리됩니다.

13.2.3 ColumnInfo > Column

```
"ColumnInfo":
{
    "Column" :
    [
        {"id":"Column0"},
        {"id":"Column1", "type":"string", "size":"256"},
        {"id":"Column2", "type":"string", "size":"256"}
    ]
}
```

속성명	필수	설명
id	O	Dataset에서 Column ID로 사용할 ID 값입니다.
type	X	데이터 타입을 설정합니다. 생략한 경우 "STRING"으로 처리합니다. type 항목을 참고하세요.
size	X	최대 데이터 크기를 설정합니다. 생략한 경우 Type에 따른 임계치로 처리합니다. 단, type이 "STRING"인 경우에는 255로 처리합니다.
prop	X	ColumnInfo 오브젝트의 prop 속성값을 설정합니다.
sumtext	X	ColumnInfo 오브젝트의 sumtext 속성값을 설정합니다.

13.2.4 Rows

```
"Rows":
[
    {"_RowType_":"U", "Column0":"", "Column1":"zzz", "Column2":""},
    {"_RowType_":"O", "Column0":"", "Column2":""},
    {"_RowType_":"N", "Column0":"A", "Column1":"B", "Column2":""},
    {"_RowType_":"D", "Column0":"a", "Column1":"b", "Column2":"c"},
    {"_RowType_":"I", "Column0":"", "Column1":"", "Column2":""}
]

"Rows":
[
    {"Column0":"A", "Column1":"B"},
```

```
    {"Column0":"a", "Column1":"b", "Column2":"c"},
    {"Column0":"", "Column1":"", "Column2":""}
]
```

속성명	필수	설명
RowType	X	레코드 형태를 설정합니다. N: Normal Record (Default) I: Inserted Record U: Updated Record D: Deleted Record O: Original Record (Update Record의 원본 레코드입니다)
<ConstColumn id>	X	ConstColumn 값을 설정합니다.
<Column id>	O	Column 값을 설정합니다.

- Row 데이터는 순서에 의미를 가집니다.
- _RowType_ 속성값이 “O” 인 경우는 바로 앞 Row 의 Original row 임을 의미합니다.
  바로 앞의 _RowType_ 속성값이 “U” 인 경우에만 정상 동작합니다.
- _RowType_ 속성값이 “O”이고 바로 앞 Row 의 _RowType_ 속성값이 "U"가 아닌 경우 해당 Row는 무시합니다. 또한 첫번째 row의 _RowType_ 속성값이 “O” 인 경우에도 Row는 무시됩니다.

13.3 참고

13.3.1 type

Parameters, ConstColumn, Column에서 type 항목에 사용할 수 있는 값은 아래와 같습니다.

> Parameters, ConstColumn의 경우 설정하지 않으면 value에 따라 Integer, Float의 경우에는 type이 "INT", "FLOAT"로 처리되고 그 외에는 "STRING"으로 처리됩니다.

Type	설명
STRING	문자열. 최대값은 system에서 한번에 할당 가능한 크기인 2GB로 제한됨 Column의 size와 관계없이 최대치까지 사용될 수 있다.
INT	정수형 (-2^{31} ~ $2^{31}-1$)
FLOAT, DECIMAL	실수형 ($\pm 2.2 \times 10^{-308}$ ~ $\pm 1.7 \times 10^{308}$)
BIGDECIMAL	실수형(문자열로 저장) ($\pm 10^{-1056}$ ~ $\pm 10^{+1056}$)

DATE	날짜형 YYYYMMDD (-8192/01/01 ~ 8191/12/31)
DATETIME	YYYYMMDDHHmmssuuu (msec포함가능)
TIME	6자리 HHmmssuuu(msec포함가능)
BLOB	이진 데이터

13.3.2 JSON 예

```
{
    "version" : "1.0",
    "Parameters":
    [
        {"id":"ErrorCode", "value":0},
        {"id":"ErrorMsg", "value":""},
        {"id":"param1", "value":0},
        {"id":"param2", "value":"0", "type":"string"}
    ],
    "Datasets" :
    [
        {
            "id":"indata",
            "ColumnInfo":
            {
                "ConstColumn":
                [
                    {"id":"ConstCol1", "value":10},
                    {"id":"ConstCol2", "type":"string", "size":"256", "value":10},
                    {"id":"ConstCol3"}
                ],
                "Column" :
                [
                    {"id":"Column0"},
                    {"id":"Column1", "type":"string", "size":"256"},
                    {"id":"Column2", "type":"string", "size":"256"}
                ]
            },
            "Rows":
            [
                {"_RowType_":"U", "Column0":"", "Column1":"zzz", "Column2":""},
                {"_RowType_":"O", "Column0":"", "Column2":""},
```

```
                {"_RowType_":"N", "Column0":"A", "Column1":"B", "Column2":""},
                {"_RowType_":"D", "Column0":"a", "Column1":"b", "Column2":"c"},
                {"_RowType_":"I", "Column0":"", "Column1":"", "Column2":""}
            ]
        },
        {
            "id": "indata2",
            "ColumnInfo":
            {
                "Column":
                [
                    {"id":"Column0"},
                    {"id":"Column1", "type":"string", "size":"256"},
                    {"id":"Column2", "type":"string", "size":"256"}
                ]
            },
            "Rows":
            [
                {"Column0":"A", "Column1":"B"},
                {"Column0":"a", "Column1":"b", "Column2":"c"},
                {"Column0":"", "Column1":"", "Column2":""}
            ]
        }
    ]
}
```