

# 초보자를 위한 자습서

---

17.0.0.800



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 | 2018/08/06

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

전화 | 02-2140-7700

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

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

제품기술문의 | 1588-7895 (오전 10시부터 오후 5시까지)

# 변경 이력

---

버전	변경일	내용
17.0.0.100	2017-10-13	17.0.0.100 공개로 전환
17.0.0.500	2018-02-22	Generate, Deploy 메뉴, 기능 변경에 따른 수정 반영
17.0.0.500.2	2018-03-20	<a href="#">X-API</a> 설치 설명을 보완했습니다.
17.0.0.800	2018-06-08	오류 메시지 관련 설명을 보완했습니다.

# 차례

---

저작권 및 면책조항	ii
변경 이력	iii
차례	iv
1. 넥사크로 스튜디오 설치	1
1.1 넥사크로 스튜디오 설치	1
1.2 설치 파일	2
1.3 버전 확인	3
1.4 실행 환경 설정	4
1.5 라이선스 인증	5
2. Hello nexacro platform	7
2.1 넥사크로 스튜디오 실행하기	7
2.2 넥사크로플랫폼 프로젝트 만들기	8
2.3 넥사크로플랫폼 애플리케이션 만들기	11
2.3.1 폼 만들기 마법사	11
2.3.2 컴포넌트 배치	13
2.3.3 Generate	15
2.3.4 이벤트 추가	17
3. 테스트 서버 환경 설정	20
3.1 로컬 웹서버 사용하기	21
3.2 경량 웹서버 사용하기	21
3.2.1 몽구스(mongoose)	21
내려받기	22
웹서버 실행	22
넥사크로플랫폼 애플리케이션 확인	23
3.3 웹 애플리케이션 서버 사용하기	24
3.3.1 JDK(Java SE Development Kit)	24
내려받기	24
설치 확인	25

3.3.2	아파치 톰캣(Apache Tomcat)	25
	내려받기	25
	설치 확인	26
	컨텍스트 설정	27
	컨텍스트 파일 생성	27
	넥사크로플랫폼 애플리케이션 확인	29
<b>4.</b>	<b>화면 만들기 (고객사 목록 조회)</b>	<b>30</b>
4.1	화면 구성	30
4.1.1	프로젝트(Project)	30
4.1.2	폼(Form)	30
4.1.3	데이터셋(Dataset)	31
4.1.4	데이터셋 콘텐츠 에디터(Dataset Contents Editor)	32
4.1.5	컴포넌트 배치	34
4.2	그리드	35
4.2.1	데이터 바인딩	35
4.2.2	그리드 콘텐츠 에디터(Grid Contents Editor)	37
4.3	데이터 테스트	41
4.3.1	버튼 클릭 이벤트	41
4.3.2	Generate	42
4.3.3	Quick View	43
<b>5.</b>	<b>화면 만들기 (트랜잭션)</b>	<b>45</b>
5.1	그리드 콤보	45
5.1.1	데이터셋(Dataset)	45
5.1.2	그리드(Grid)	47
5.1.3	Generate	48
5.2	트랜잭션	49
5.2.1	sample.xml	49
5.2.2	버튼 클릭 이벤트	51
5.2.3	콜백 함수	52
5.3	로그	53
5.3.1	구글 크롬	53
5.3.2	파이어폭스	53
5.3.3	인터넷 익스플로러	54
<b>6.</b>	<b>화면 만들기 (X-API)</b>	<b>55</b>
6.1	서비스	55
6.2	X-API	56
6.2.1	배포 파일	56
6.2.2	설치	57

6.2.3	전용객체	60
6.3	화면에 Button 컴포넌트 추가	60
6.4	initdata.jsp	61
6.4.1	pseudo code	61
6.4.2	코드 구현	61
	자바 라이브러리 지정	61
	MIME 타입 정의	61
	기본객체(PlatformData) 생성하기	62
	Dataset을 생성하여 File로 저장하기	62
	ErrorCode, ErrorMsg 처리하기	63
	결과값 Client에게 보내기	63
6.4.3	전체 코드	64
6.4.4	데이터 초기화 이벤트	66
6.4.5	저장된 파일	68
6.5	search.jsp	68
6.5.1	pseudo code	68
6.5.2	코드 구현	69
	자바 라이브러리 지정	69
	MIME 타입 정의	69
	기본객체(PlatformData) 생성하기	69
	File의 내용을 읽어서 Dataset 생성하기	69
	ErrorCode, ErrorMsg 처리하기	70
	결과값 Client에게 보내기	70
6.5.3	전체 코드	72
6.5.4	데이터 조회 이벤트	73
6.5.5	데이터베이스 연결	74
6.6	save_list.jsp	76
6.6.1	pseudo code	76
6.6.2	코드 구현	77
	자바 라이브러리 지정	77
	MIME 타입 정의	77
	기본객체(PlatformData) 생성하기	77
	클라이언트 요청받기	77
	클라이언트가 보낸 데이터 추출 후 파일로 저장하기	78
	ErrorCode, ErrorMsg 처리하기	78
	결과값 Client에게 보내기	79
6.6.3	전체 코드	79
6.6.4	데이터 저장 이벤트	81

# 1.

## 넥사크로 스튜디오 설치

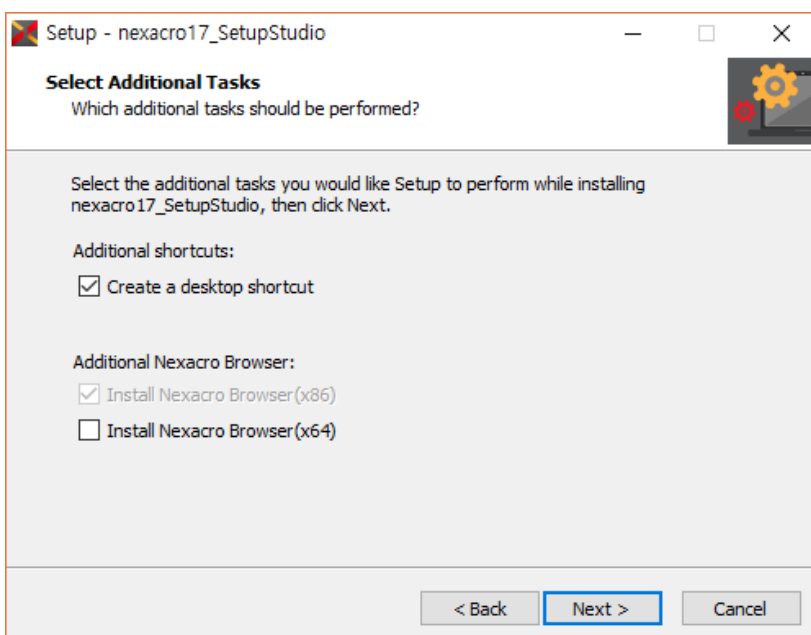
### 1.1 넥사크로 스튜디오 설치

넥사크로 스튜디오는 넥사크로플랫폼 설치 패키지에 포함되어 배포됩니다. 배포된 Setup 파일(nexacro17\_SetupStudio.exe)을 실행하면 설치 과정이 진행됩니다.

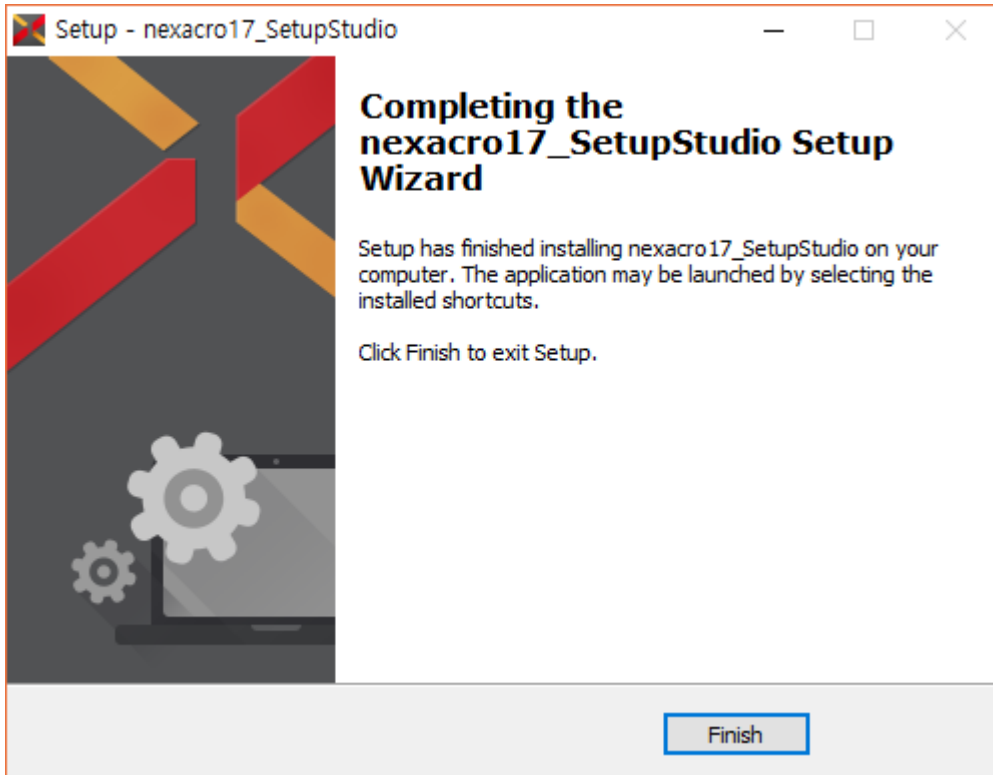


Windows Vista 이상의 운영 체제는 보안경고창이 나타날 수 있습니다. 반드시 '실행'을 선택해 주시기 바랍니다.

바탕화면에 단축 아이콘을 만들고자 한다면 [Create a desktop shortcut] 항목을 체크합니다. 64비트 운영체제를 사용하고 있다면 64비트를 지원하는 넥사크로 브라우저를 설치할 수 있습니다. 64비트 넥사크로 브라우저를 설치하려면 [Install Nexacro Browser(x64)] 항목을 체크합니다.



[Finish] 버튼을 클릭하여 프로그램 설치를 완료합니다.



## 1.2 설치 파일

넥사크로 스튜디오를 설치한 폴더 내 생성되는 주요 파일과 폴더는 아래와 같습니다.

폴더명	설명
help	- 넥사크로플랫폼 도움말 폴더 컴파일된 HTML 도움말 파일(compiled html help, CHM)로 제공합니다. 넥사크로 스튜디오는 운영체제 언어 설정에 따라 해당하는 한국어, 영어, 일본어 도움말 파일을 실행합니다.
nexacro17lib	- 넥사크로플랫폼 라이브러리 폴더 설치된 버전의 라이브러리 파일이 포함된 폴더입니다. metainfo 파일을 포함하고 있습니다. metainfo 파일은 넥사크로 스튜디오에서만 사용하는 파일이며 배포 대상은 아닙니다.
license	- 라이선스 관련 문서 폴더 제품 라이선스, 오픈소스 라이선스 관련 문서 파일이 포함된 폴더입니다.
theme	- 테마 폴더 넥사크로 스튜디오 테마, 운영체제별 기본 테마 파일이 포함된 폴더입니다.
Embedded	- 넥사크로 스튜디오 내에서 사용하는 넥사크로플랫폼 애플리케이션 그리드 콘텐츠 에디터나 에뮬레이터처럼 넥사크로플랫폼을 사용해 만든 기능 관련 소스



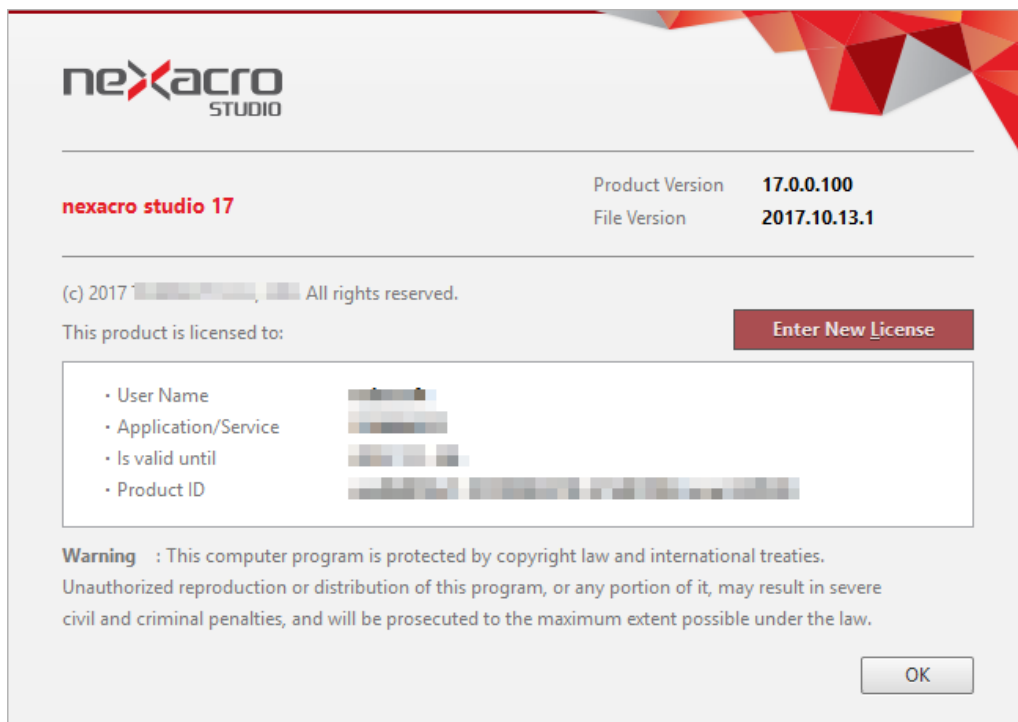
폴더명	설명
	코드가 저장된 폴더입니다.
파일명	설명
nexacro.exe	- 넥사크로 런타임 브라우저 실행 파일
nexacrodeploy17.exe	- 프로젝트 제너레이트, 난독화 실행 파일
nexacromigrator17.exe	- 이전 버전 프로젝트 마이그레이션 실행 파일
ResourceUpdater_x64.exe ResourceUpdater_x86.exe	- 윈도우 런타임 배포 파일 생성 실행 파일
nexacroemulator17.exe	- 에뮬레이터 실행 파일
nexacrostudio17.exe	- 넥사크로 스튜디오 실행 파일
unins000.exe	- 넥사크로 스튜디오 삭제 실행 파일 제어판에서 해당 프로그램을 찾아 삭제할 수도 있습니다.



설치하는 제품 버전에 따라 설치된 파일 목록이 일부 변경될 수 있습니다.

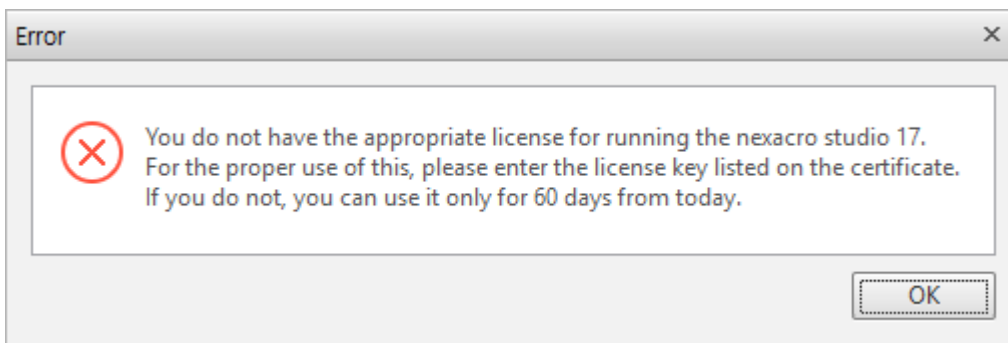
## 1.3 버전 확인

넥사크로 스튜디오의 메뉴[Help > About nexacro studio]를 선택하면 설치된 넥사크로 스튜디오의 버전과 등록된 라이선스 정보를 확인할 수 있습니다.

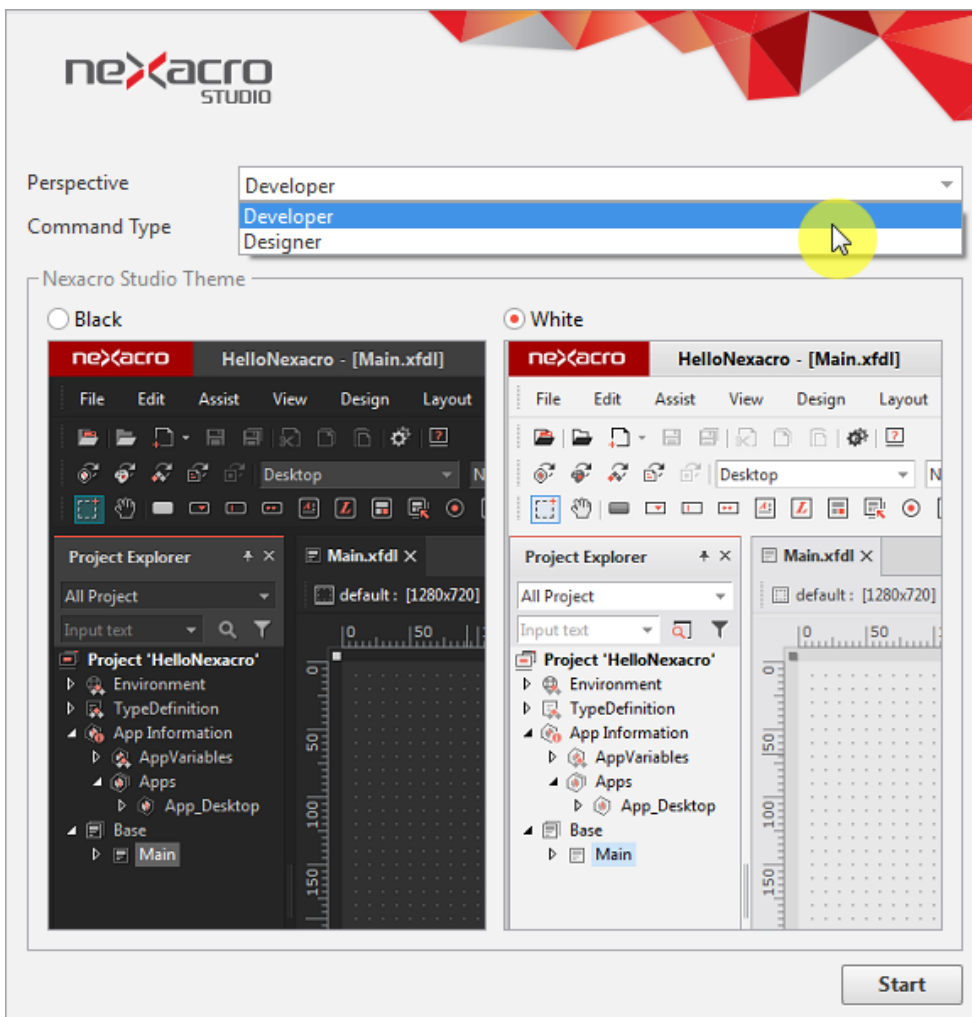


## 1.4 실행 환경 설정

넥사크로 스튜디오를 처음 설치하면 60일 동안 사용할 수 있는 체험판으로 설정됩니다. 최초 실행 시 경고 팝업 창이 표시되는데 [OK] 버튼을 클릭하면 다음 단계로 진행할 수 있습니다. 제품을 구매한 경우에는 발급받은 Product Key를 입력해 사용할 수 있습니다.

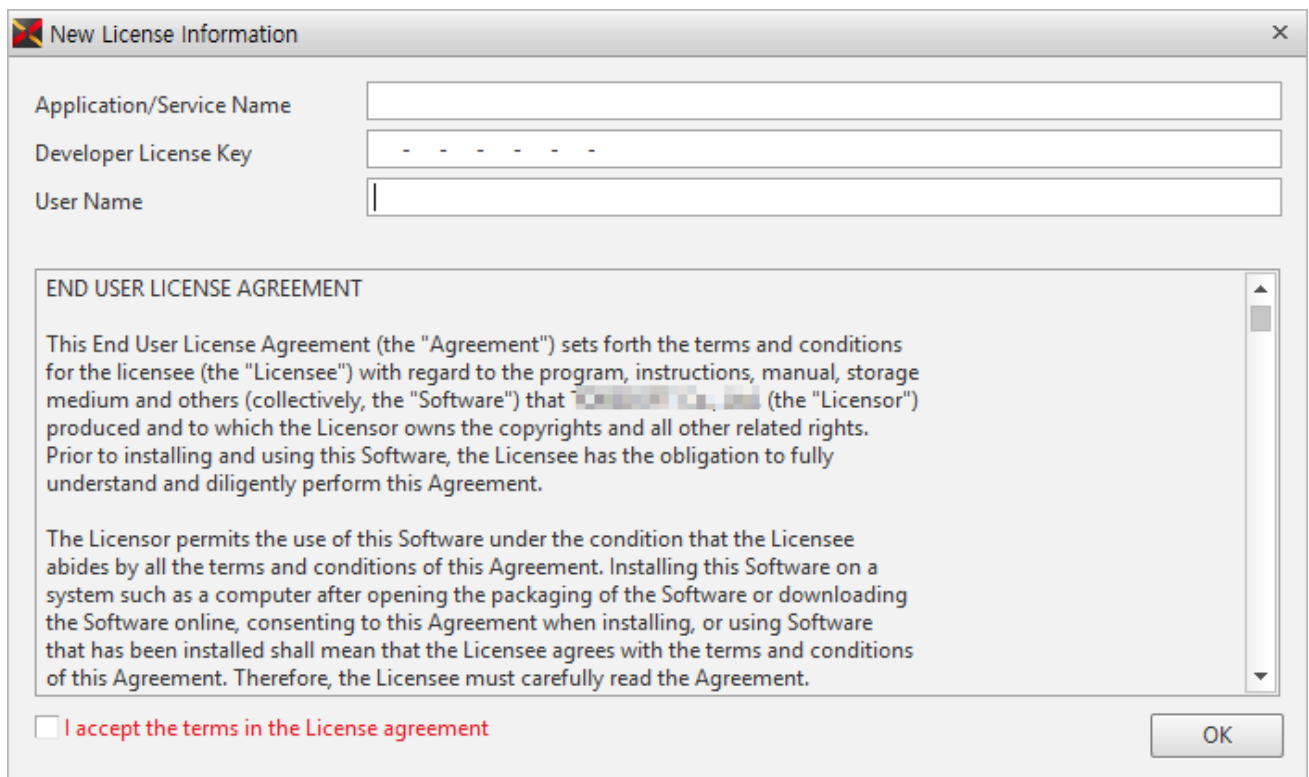


넥사크로 스튜디오 실행 환경을 설정합니다. Perspective, Command Type, Theme를 설정할 수 있습니다. 설정한 항목은 메뉴[Options > Environment > General > Development Tools]에서 변경할 수 있습니다.



## 1.5 라이선스 인증

넥사크로 스튜디오의 메뉴[Help > About nexacro studio]를 클릭하고 버전 확인창에서 'Enter New License' 버튼을 클릭합니다. 라이선스 입력 창에는 'Application/Service Name'과 'Developer License Key' 2개 항목을 입력합니다.



**New License Information**

Application/Service Name:

Developer License Key:

User Name:

**END USER LICENSE AGREEMENT**

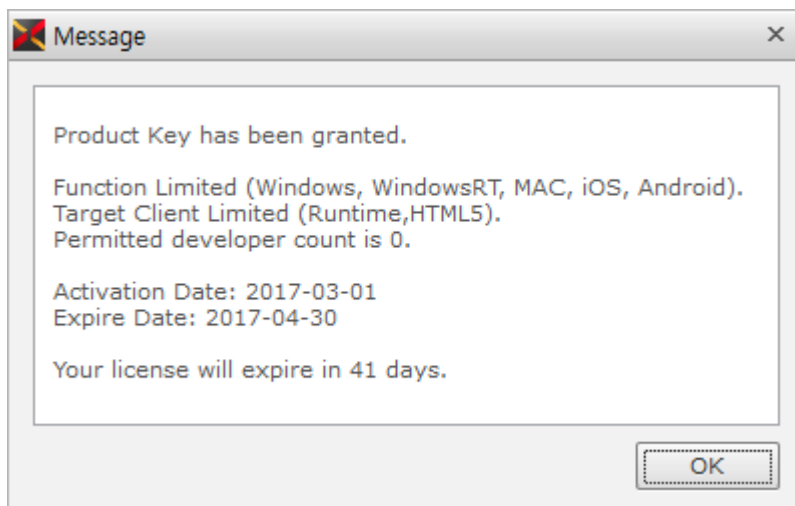
This End User License Agreement (the "Agreement") sets forth the terms and conditions for the licensee (the "Licensee") with regard to the program, instructions, manual, storage medium and others (collectively, the "Software") that [REDACTED] (the "Licensor") produced and to which the Licensor owns the copyrights and all other related rights. Prior to installing and using this Software, the Licensee has the obligation to fully understand and diligently perform this Agreement.

The Licensor permits the use of this Software under the condition that the Licensee abides by all the terms and conditions of this Agreement. Installing this Software on a system such as a computer after opening the packaging of the Software or downloading the Software online, consenting to this Agreement when installing, or using Software that has been installed shall mean that the Licensee agrees with the terms and conditions of this Agreement. Therefore, the Licensee must carefully read the Agreement.

☐ I accept the terms in the License agreement

OK

정상적인 값이 입력되었다면 아래와 같은 메시지가 뜨면서 라이선스 인증을 처리합니다.



**Message**

Product Key has been granted.

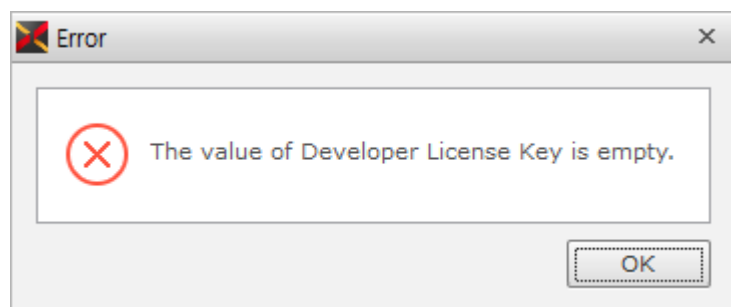
Function Limited (Windows, WindowsRT, MAC, iOS, Android).  
Target Client Limited (Runtime,HTML5).  
Permitted developer count is 0.

Activation Date: 2017-03-01  
Expire Date: 2017-04-30

Your license will expire in 41 days.

OK

잘못된 값을 입력하거나 이미 인증된 코드를 입력하면 아래와 같은 메시지가 뜨면서 라이선스 인증이 처리되지 않습니다.



## 2.


---

# Hello nexacro platform

넥사크로플랫폼 애플리케이션 개발 환경을 이해하고 넥사크로 스튜디오를 다루는 기본적인 방법을 배우기 위해 간단한 애플리케이션을 작성해보겠습니다.

이번 장에서는 아래와 같은 내용을 다룹니다. 화면에 'Hello, nexacro platform!'이라는 텍스트를 보여주고 텍스트를 클릭했을 때 경고창에 'nexacro platform 17'이라는 텍스트를 보여주는 애플리케이션을 만듭니다.

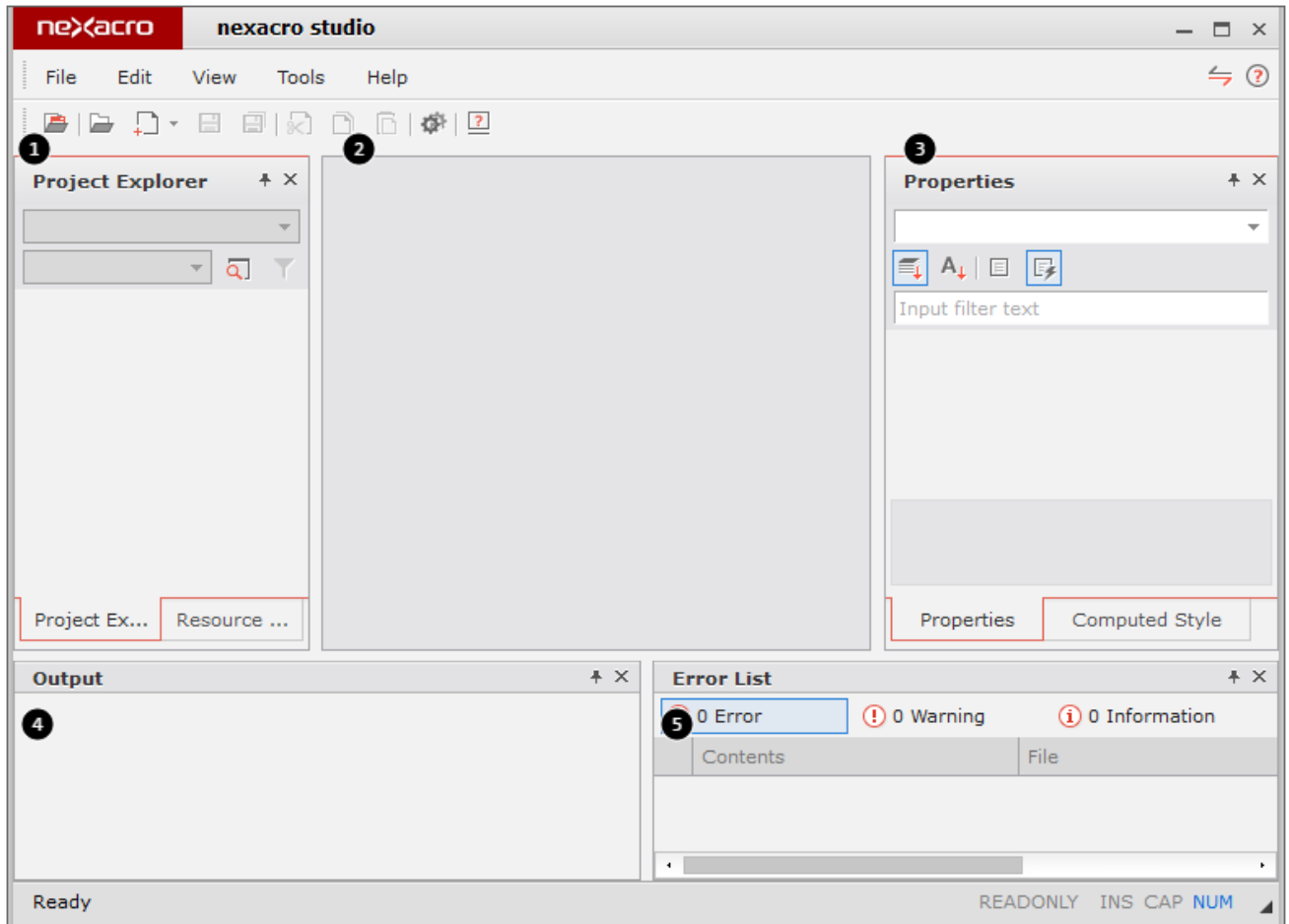
## 2.1 넥사크로 스튜디오 실행하기

PC에 설치된 넥사크로 스튜디오를 실행합니다. 설치 과정에서 바탕화면에 아이콘 을 생성했다면 해당 아이콘을 더블 클릭해 바로 실행할 수 있습니다.

단축 아이콘을 생성하지 않았다면 넥사크로 스튜디오가 설치된 폴더에서 실행 파일을 더블 클릭해 실행할 수 있습니다. 넥사크로 스튜디오가 설치된 경로는 아래와 같습니다.

`C:\[Program Files]\nexacro\17\nexacrostudio17.exe`

넥사크로 스튜디오 설치 후 최초 실행 시에는 어떤 프로젝트도 열리지 않은 상태입니다. 넥사크로 스튜디오의 화면 구성은 아래와 같습니다.



구성요소	설명
① 프로젝트 탐색기 (Project Explorer)	작업 중인 프로젝트의 구성 요소를 표시합니다. 최초 실행 시에는 아무것도 보이지 않습니다.
② Form Design	넥사크로플랫폼 애플리케이션 화면 디자인, 스크립트 편집 작업을 수행하는 공간입니다.
③ 속성 창 (Properties)	폼, 컴포넌트, Dataset 컴포넌트의 속성을 보여주고 편집할 수 있는 공간입니다.
④ Output	오류 메시지나 Generate 메시지, trace() 메소드로 설정한 메시지를 보여주는 공간입니다.
⑤ Error List	스크립트 작성 중 오류를 실시간으로 확인하고 표시합니다.

## 2.2 넥사크로플랫폼 프로젝트 만들기

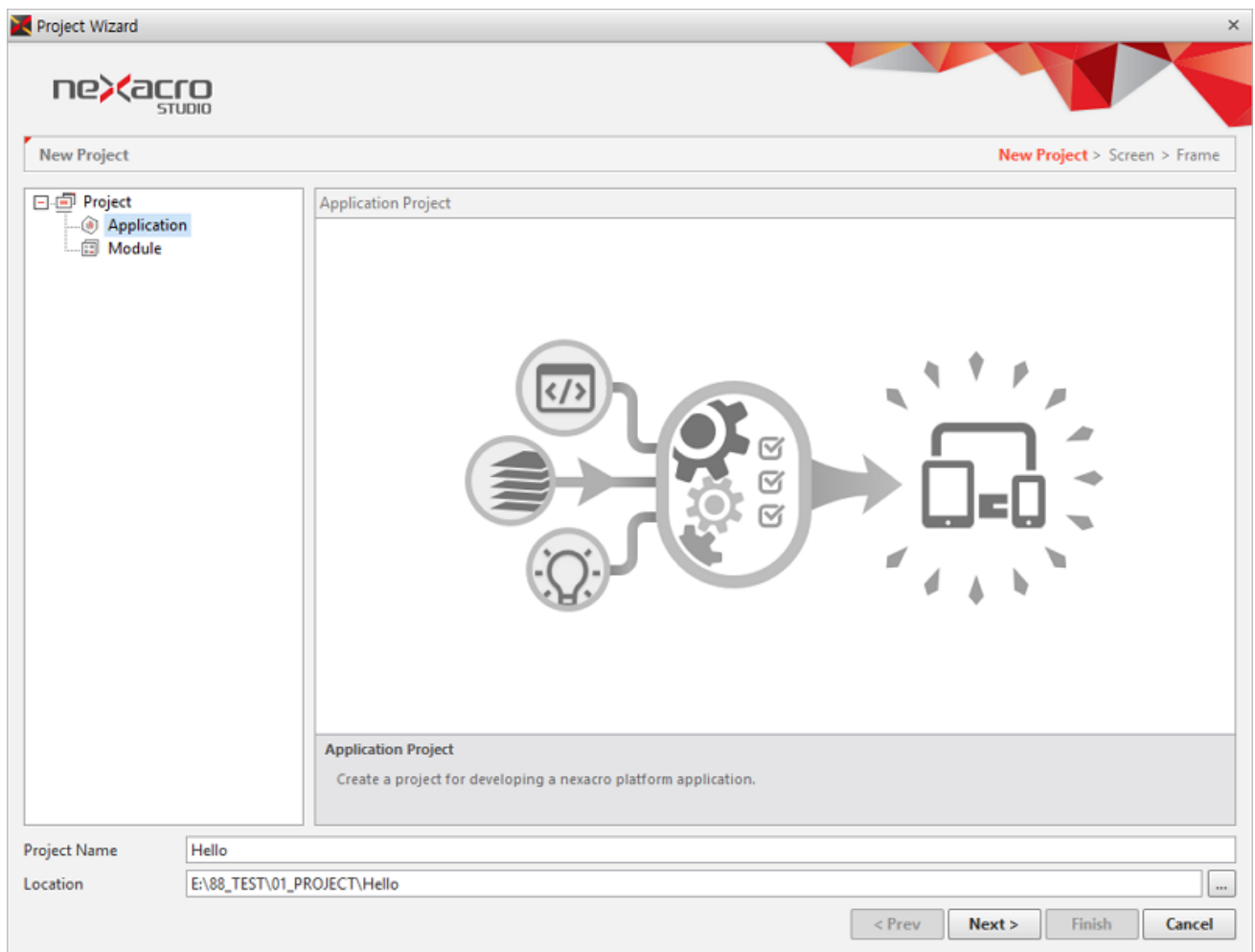
넥사크로플랫폼 애플리케이션을 만들기 위해서는 프로젝트를 먼저 만들어야 합니다. 모든 넥사크로플랫폼 애플리케이션은 프로젝트에서 지정된 속성을 기반으로 사용자 화면에 보이고 동작하게 됩니다.

프로젝트는 새로 만들거나 이미 만들어진 템플릿 프로젝트를 사용할 수 있습니다. 이번 장에서는 새로운 프로젝트를 만듭니다.

모든 작업은 메뉴 또는 툴바, 단축키로 실행할 수 있습니다. 이번 장에서 컴포넌트 배치를 제외한 모든 작업은 메뉴를 기준으로 설명합니다.

넥사크로플랫폼 프로젝트를 만들기 위한 메뉴는 아래와 같습니다. New Project Wizard가 실행되며 순서에 따라 새로운 프로젝트를 만듭니다.

[Menu] File > New > Project

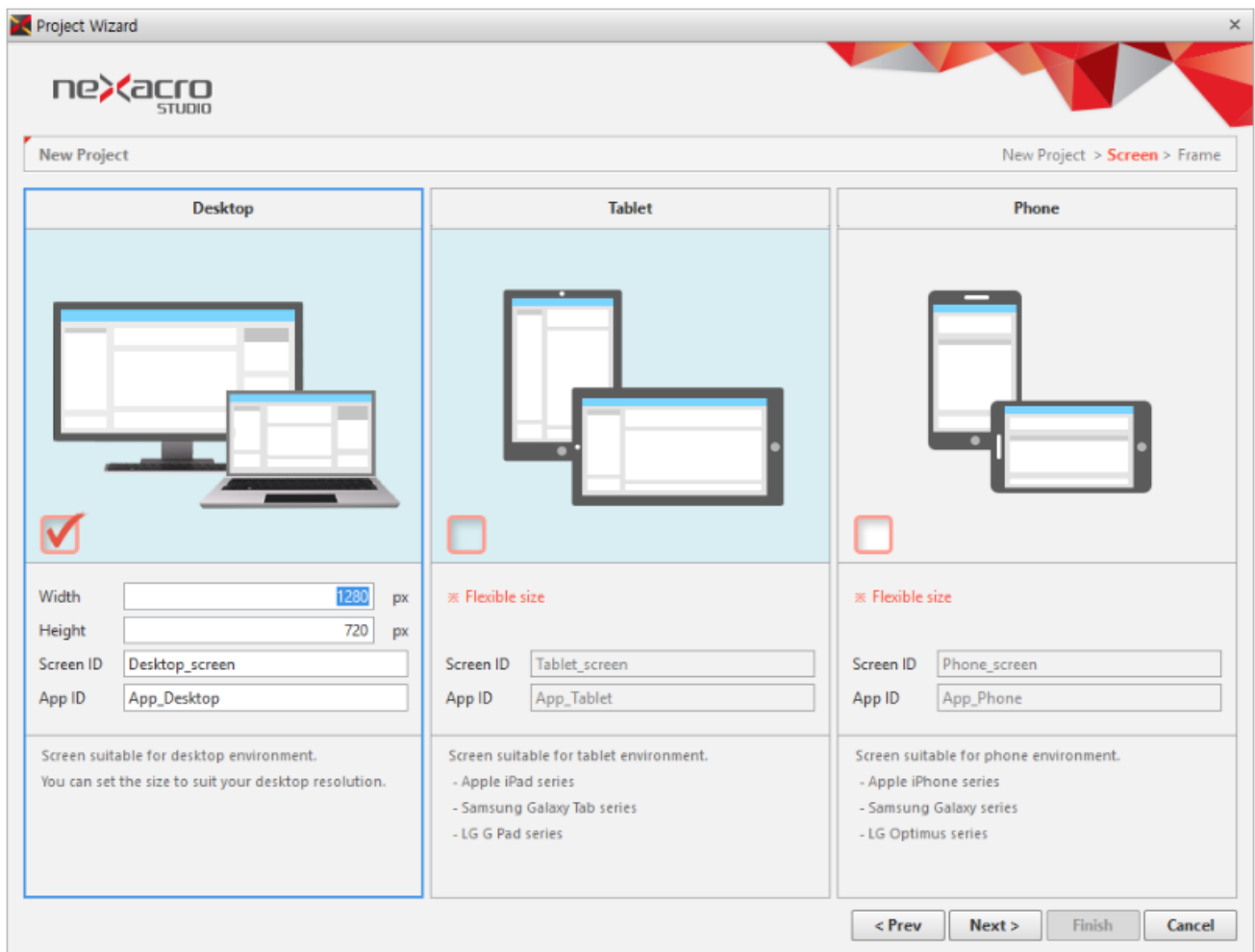


새로운 프로젝트를 만들기 위해서는 프로젝트 이름과 프로젝트 파일을 저장할 위치를 지정해주어야 합니다. ❶ 'Name' 항목에 프로젝트 이름을 지정하고 ❷ 'Location' 항목에 저장할 위치를 지정합니다.

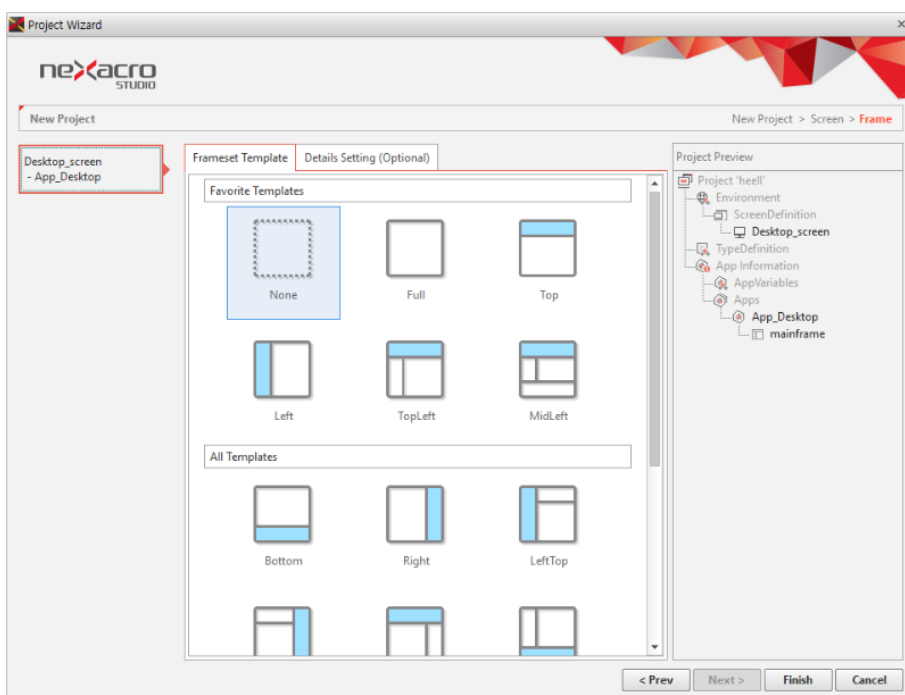
이번 예제에서는 아래와 같이 설정합니다.

	설명	설정값
❶ Name	프로젝트 이름	Hello
❷ Location	프로젝트 파일 저장 위치	[원하는 폴더]wHello

Screen 항목과 Screen Detail 항목을 설정합니다. 이번 장에서는 기본값을 그대로 사용합니다. Screen 항목은 'Desktop'을 선택하고 자동으로 생성되는 Screen 정보도 그대로 사용합니다.



Frame 항목도 기본값을 그대로 사용합니다. Screen 관련 설정이나 Frame 설정은 프로젝트 생성 후에 수정할 수 있습니다. [Finish] 버튼을 클릭해 프로젝트를 생성합니다.





프로젝트가 만들어지면서 프로젝트 탐색창에 프로젝트 구성 요소가 표시됩니다. 각 구성 요소에 대한 세부적인 설명은 이번 예제에서 다루지 않고 넥사크로플랫폼 애플리케이션 만들기로 넘어갑니다.



이번 장에서 설명하는 경로는 윈도우 10 운영체제를 기준으로 설명합니다. 운영체제에 따라 설정된 경로가 달라질 수 있습니다.



Location 항목을 따로 지정해주지 않으면 아래 경로로 자동 지정됩니다.  
C:\Users\[User]\Documents\nexacro\17\projects\

## 2.3 넥사크로플랫폼 애플리케이션 만들기

넥사크로플랫폼 애플리케이션은 폼(Form)을 기반으로 동작합니다. 프로젝트가 애플리케이션을 위한 무대를 만든 것이라면 무대를 꾸미고 동선을 배치하고 효과를 추가하는 작업은 폼에서 담당합니다.

### 2.3.1 폼 만들기 마법사

넥사크로플랫폼 폼을 만들기 위한 메뉴는 아래와 같습니다. New Form Wizard가 실행되며 순서에 따라 새로운 폼을 만듭니다.

[Menu] File > New > Form

**Form Wizard**

Information > Position

**New Form**

Name: ① frm\_hello

Location: ② Base

Layout: default

Layout Information	
name	default
screenid	
width	1280 px
height	720 px
stepcount	
stepindex	
description	
mobileorientation	landscape

③ < Prev    Next >    ④ Finish    Cancel

새로운 폼을 만들기 위해서는 폼 이름과 서비스 그룹을 저장할 위치를 지정해주어야 합니다. ① 'Name' 항목에 만들려는 폼 이름을 지정하고 ② 'Location' 항목에 서비스 그룹을 지정합니다. 서비스 그룹은 기본값으로 Base가 생성되어 있습니다.

이번 예제에서는 아래와 같이 설정합니다.

	설명	설정값
① Name	폼 이름	frm_hello
② Location	서비스 그룹 (기본값은 Base)	Base

Layout 항목은 기본값 그대로 설정합니다. ③ [Next] 버튼을 클릭하면 Position 항목이 나오지만, 이번 예제에서는 필요 없는 항목이기 때문에 ④ [Finish] 버튼을 클릭해 바로 폼을 생성합니다.

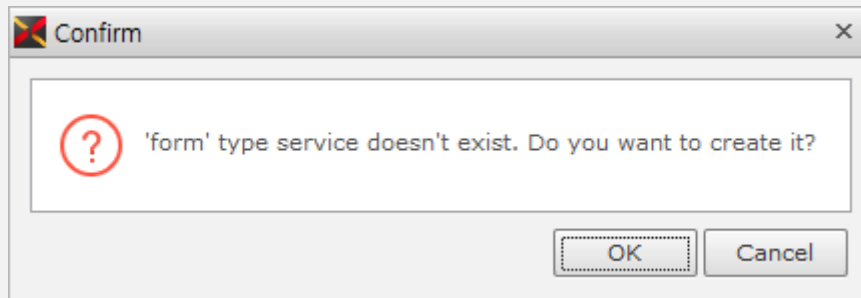


폼의 기본 크기는 옵션에서 지정할 수 있습니다. 메뉴 [Tools > Options > Form Design > General] 항목에서 'Default Width', 'Default Height' 항목을 수정하면 기본 생성되는 폼의 크기를 수정할 수 있습니다.



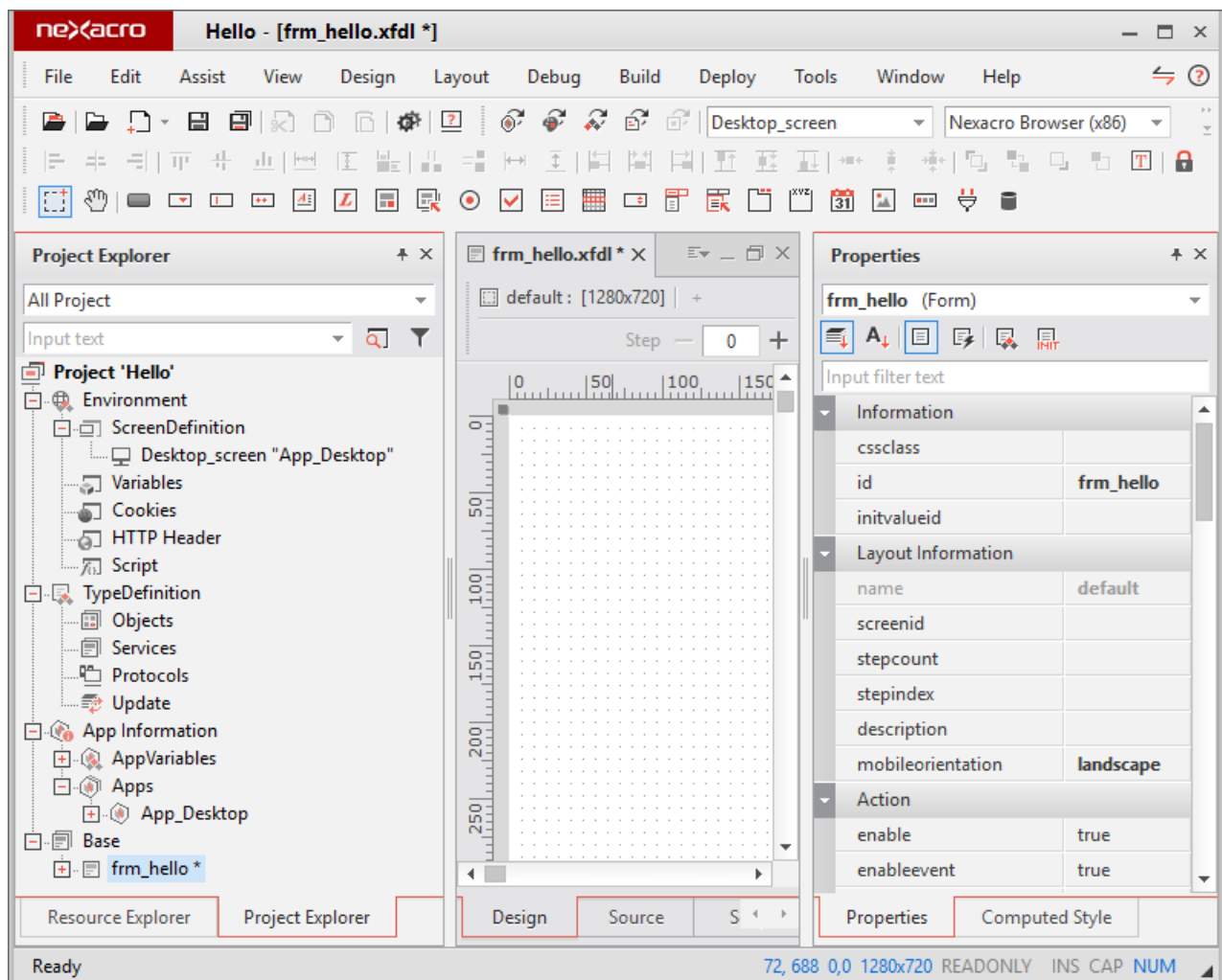
프로젝트를 만들고 설정이 완료되기 위해서는 약간의 시간이 필요합니다. 너무 빨리 폼을 만들려고 하는 경우에는 아래와 같이 서비스가 없다는 메시지가 보일 수 있습니다.

TypeDefinition에서 서비스를 모두 지운 경우에도 같은 메시지가 보입니다.



## 2.3.2 컴포넌트 배치

새로운 폼이 만들어지면 Form Design 창이 활성화됩니다.




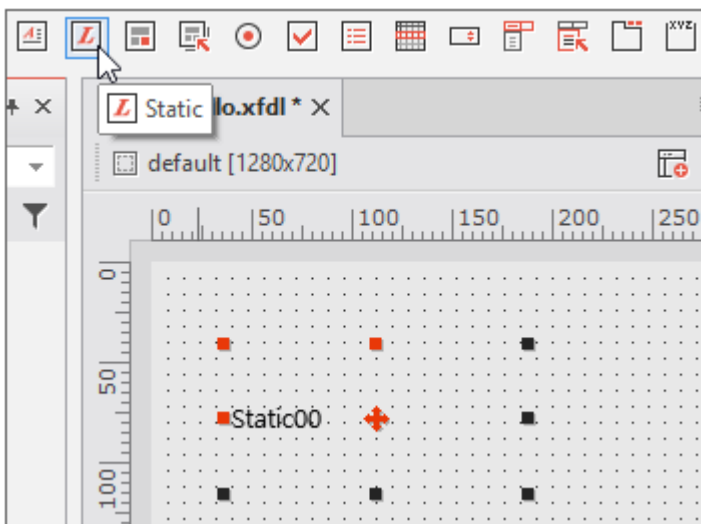


화면에 보이는 툴바는 설정에 따라 달라질 수 있습니다. 화면에 나타나는 툴바는 [View > Toolbars] 메뉴에서 설정할 수 있습니다.

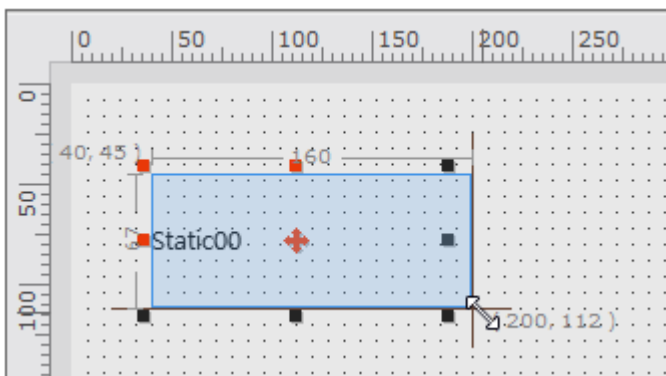


넥사크로 스튜디오 메뉴는 리본 메뉴와 커맨드 메뉴 중 하나를 선택해 사용할 수 있습니다. 이 문서에서는 커맨드 메뉴로 설정된 상태에서 설명합니다.

넥사크로플랫폼 애플리케이션에서 글자를 표기할 때는 Static 컴포넌트를 사용합니다. 'Objects' 툴바에서 Static 컴포넌트 를 마우스로 선택하고 Form Design 창에서 배치하기 원하는 위치를 마우스로 클릭하면 기본값으로 정해진 크기로 Static 컴포넌트가 배치됩니다.



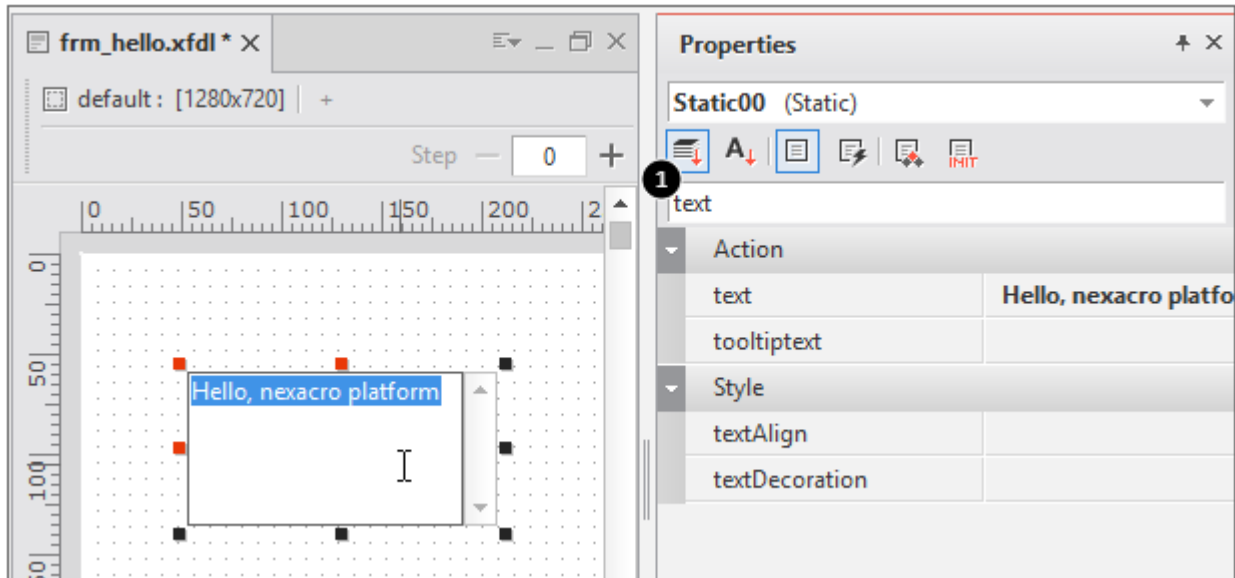
컴포넌트를 Form Design 창에 배치할 때 크기를 조정하고자 한다면 'Objects' 툴바에서 컴포넌트를 마우스로 선택하고 컴포넌트를 그리기 시작할 위치에서 마우스 왼쪽 버튼을 누른 상태에서 마우스를 드래그하면 원하는 크기로 컴포넌트를 배치할 수 있습니다. 컴포넌트를 배치한 이후에도 마우스로 컴포넌트를 선택하고 컴포넌트의 크기를 조정할 수 있습니다.



Static 컴포넌트의 텍스트를 수정할 때는 컴포넌트를 선택한 후 단축키(F2)를 누르면 편집 모드로 전환되며 직접 수정할 수 있습니다. Form Design 창이 아닌 속성 창에서도 해당 속성을 수정할 수 있습니다. Static 컴포넌트를 선택하

고 속성 창에서 text 속성 항목을 수정합니다.

수정할 속성명을 알고 있는 경우에는 찾기 원하는 문자열을 ❶ 번 창에 입력하면 원하는 항목을 바로 찾을 수 있습니다. 이번 예제에서는 'text'라는 문자열을 입력했습니다.



컴포넌트의 텍스트를 Form Design 창에서 직접 수정할 때 컴포넌트를 클릭해서 선택하고 다시 클릭해서 텍스트 편집 상태로 들어갑니다. 클릭 사이의 간격이 짧으면 더블클릭으로 인식되어 onclick 이벤트 스크립트 편집창으로 이동합니다.

### 2.3.3 Generate

수정된 넥사크로플랫폼 애플리케이션을 확인하고 싶다면 Quick View를 사용합니다. Quick View를 실행하는 메뉴는 아래와 같습니다.

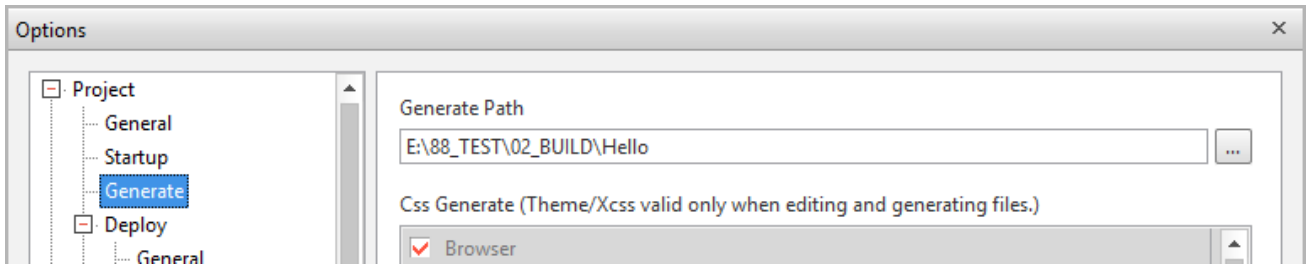
[Menu] Generate > Quick View

넥사크로 스튜디오에서 작성된 코드는 바로 실행하지 않고 자바스크립트 코드로 변환(Generate)하는 과정이 필요합니다. Generate는 Form을 생성하거나 수정 후 저장할 때 자동으로 처리됩니다. 아래 메뉴를 사용해 현재 열린 프로젝트를 Generate 할 수 있습니다.

[Menu] Generate > Application

Generate Path는 아래 메뉴에서도 등록할 수 있습니다. Generate Path에는 변환된 자바스크립트 파일이 만들어질 폴더를 지정합니다. 지정된 폴더 아래 프로젝트 이름으로 새로운 폴더가 만들어지면서 자바스크립트 파일이 만들어집니다.

[Menu] Tools > Options > Project > Generate > Generate Path



Working Folder, Generate Path, Base Lib Path 기본 설정은 아래와 같습니다.

**Working Folder**

C:\Users\[User]\Documents\nexacro\17\projects

**Generate Path**

C:\Users\[User]\Documents\nexacro\17\outputs

**Base Lib Path**

C:\[Program Files]\nexacro\17\nexacro17lib

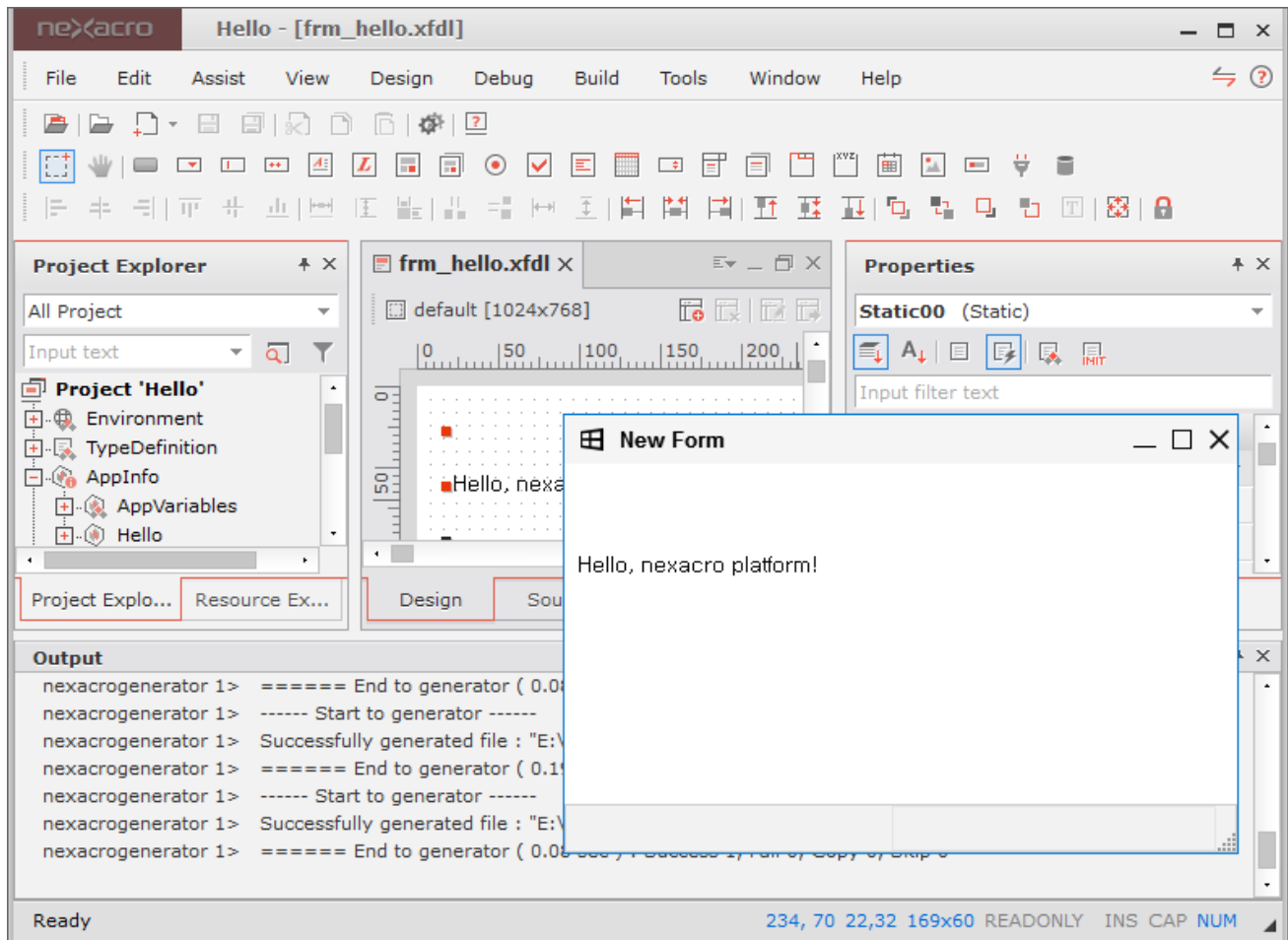
넥사크로 스튜디오에서 내부적으로 코드 변환 과정을 실행하며 앞에서 지정한 Generate Path에 필요한 자바스크립트 파일을 만듭니다. 해당 폴더에는 아래와 같은 파일이 생성됩니다.

```
[_resource/_theme_]
[Base]
[nexacro17lib]
environment_xml.js
App_Desktop.xadl.js
App_Desktop.xadl.quickview.js
index.html
launch.html
popup.html
quickView.html
start.json
```



넥사크로플랫폼 버전에 따라 생성되는 파일명이나 폴더명은 변경될 수 있습니다.

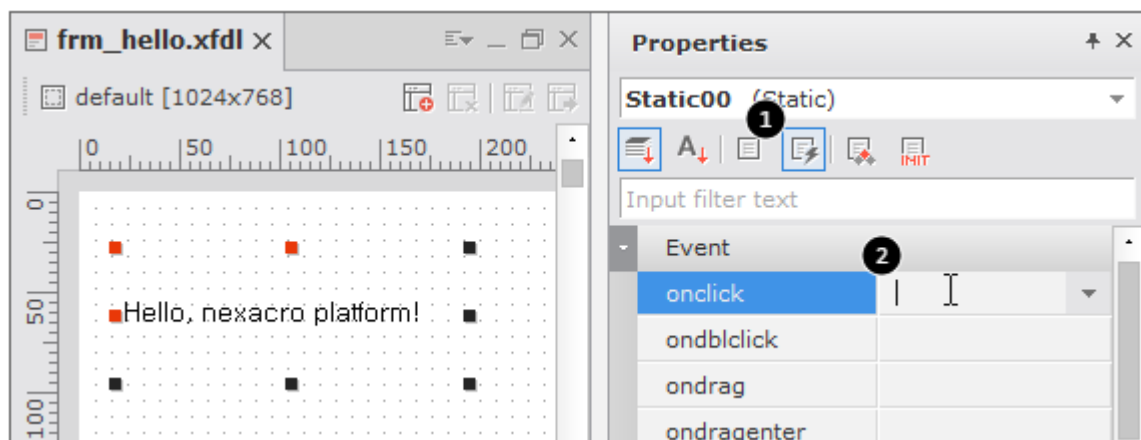
변환된 자바스크립트 파일이 만들어지면서 Quick View가 실행되는 것을 확인할 수 있습니다.



## 2.3.4 이벤트 추가

Static 컴포넌트를 더블클릭했을 때 경고창을 보여주는 기능을 추가합니다. 사용자의 특정 행위나 조건에 따라 다른 동작이 발생하는 것을 이벤트라고 합니다.

넥사크로 스튜디오에서 Static 컴포넌트를 선택하고 속성창에서 ❶ [Event] 아이콘 선택 후 onclick 항목 ❷ 입력창을 더블클릭합니다.



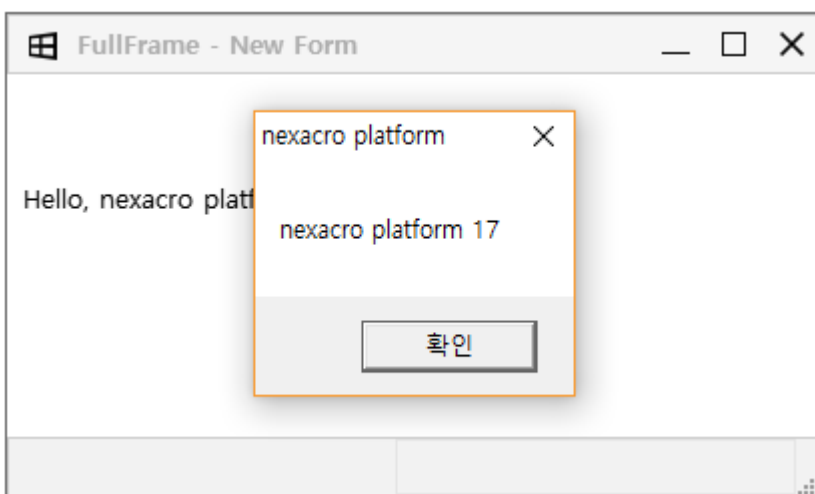
넥사크로 스튜디오에서 클릭 이벤트를 추가할 때는 Form Design 창에서 해당 컴포넌트를 더블클릭해주면 됩니다. 작업 창에 Form Script 창으로 전환되면서 이벤트를 지정하는 함수가 자동으로 만들어집니다.

```
this.Static00_onclick = function(obj:nexacro.Static,e:nexacro.ClickEventInfo)
{

};
```

만들어진 함수에는 해당 컴포넌트를 클릭했을 때 실행될 동작을 지정할 수 있습니다. 경고창에 특정 문구를 보여주는 코드를 추가합니다.

```
this.alert("nexacro platform 17");
```



이번 예제에서 생성된 폼(frm\_hello.xfdl)의 전체 소스 코드는 아래와 같습니다. 넥사크로플랫폼 애플리케이션을 개발하는 과정에서 직접 소스 코드를 수정하지는 않지만, 소스 코드를 보면서 어떻게 애플리케이션이 구성되었는지 확인할 수 있습니다.

frm\_hello.xfdl

```
<?xml version="1.0" encoding="utf-8"?>
<FDL version="2.0">
  <Form id="frm_hello" width="1024" height="768" titletext="New Form">
    <Layouts>
      <Layout height="768" mobileorientation="landscape" width="1024">
        <Static id="Static00" taborder="0" text="Hello, nexacro platform!" left="40" top="40"
width="170" height="60" onclick="Static00_onclick"/>
      </Layout>
    </Layouts>
    <Script type="xscript5.1"><![CDATA[
```



```
this.Static00_onclick = function(obj:nexacro.Static,e:nexacro.ClickEventInfo)
{
    this.alert("nexacro platform 17");
};
]]></Script>
</Form>
</FDL>
```

### 3.

## 테스트 서버 환경 설정

넥사크로플랫폼 애플리케이션을 웹브라우저에서 직접 실행하기 위해서는 실행 환경을 만들어주어야 합니다. 넥사크로 스튜디오에서 Windows Runtime으로 테스트해볼 수 있지만 다양한 웹브라우저에서 실제 동작하는 내용을 점검해보고 싶다면 각 개발환경에서 별도의 실행 환경을 만들어주어야 합니다.

넥사크로플랫폼 애플리케이션이 동작하기 위해서는 넥사크로 스튜디오에 포함된 로컬 웹서버를 사용하거나 직접 웹서버만 설치해도 무관하지만, 데이터 트랜잭션까지 테스트해보고자 한다면 웹 애플리케이션 서버(WAS, Web Application Server)를 설정해야 합니다. 아래에서는 간단한 테스트를 위해 경량 웹서버인 몽구스(mongoose)를 사용하는 방법과 WAS 설치를 위한 JDK 설치와 아파치 톰캣 설정에 대해 설명합니다.

로컬 웹서버, 몽구스와 아파치 톰캣은 아래와 같은 차이가 있습니다. 사용하려는 목적에 맞게 선택하세요.

	로컬 웹서버	mongoose	Apache Tomcat
설치파일	없음 (넥사크로 스튜디오 설치에 포함)	없음 (별도 설치 파일 없이 실행파일만 제공되며 더블클릭으로 바로 실행)	설치 파일 제공 (바이너리 소스로 받아서 직접 배치 파일을 실행시킬 수도 있음)
JDK 설치	필요 없음	필요 없음	JDK 설치 후 사용 가능
Context 설정	없음 (Generate 폴더가 웹서버 root로 지정)	없음 (실행파일이 실행되는 폴더가 웹서버 root로 지정)	정해진 절차에 따라 컨텍스트를 추가하거나 war 파일을 디플로이해서 컨텍스트 생성
X-API 지원	사용할 수 없음	사용할 수 없음	사용 가능 (별도 설정)



로컬 웹서버는 넥사크로 스튜디오를 실행하는 중에만 사용할 수 있습니다. 넥사크로 스튜디오를 실행하지 않고 애플리케이션을 테스트하려면 직접 웹서버나 WAS를 설치해야 합니다.

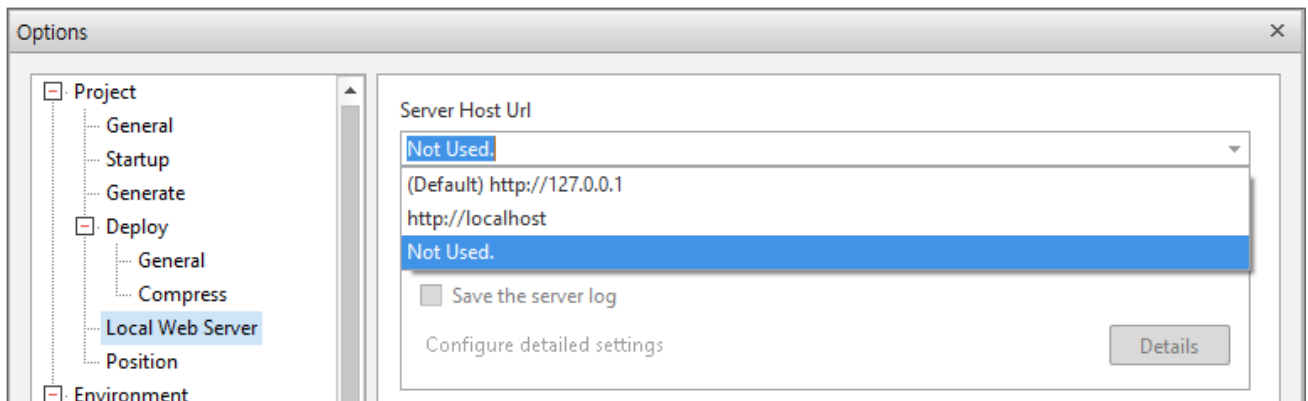


이번 장에서 설명하는 각 서비스에 대한 문제나 세부적인 사용 방법은 해당 제품 프로젝트 사이트를 참조하세요.

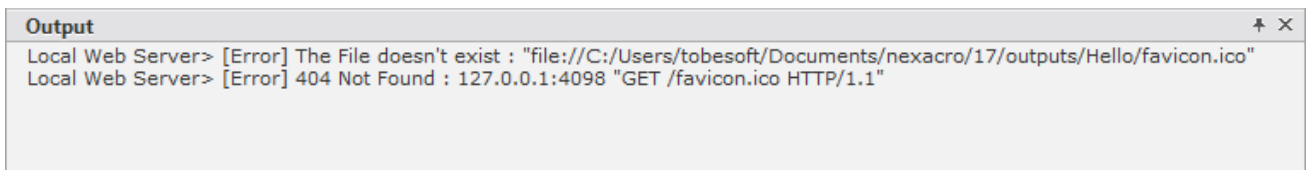
## 3.1 로컬 웹서버 사용하기

넥사크로 스튜디오 설치 후 기본 옵션은 로컬 웹서버를 사용하도록 설정됩니다. 로컬 웹서버를 사용하는 경우에는 별도의 설정 없이 넥사크로 스튜디오에서 "Launch" 또는 "Quick View" 실행 시 웹 브라우저를 선택한 경우에는 로컬 웹서버가 자동으로 실행되고 넥사크로 스튜디오를 종료하면 자동으로 로컬 웹서버도 종료됩니다.

로컬 웹서버를 사용하지 않을 경우에는 메뉴[Tools > Options > Project > Local Web Server] 항목에서 "Not Used"로 설정을 변경해야 합니다.



로컬 웹서버 실행 시 서버에서 출력하는 메시지는 넥사크로 스튜디오 Output 창에서 확인할 수 있습니다.



## 3.2 경량 웹서버 사용하기

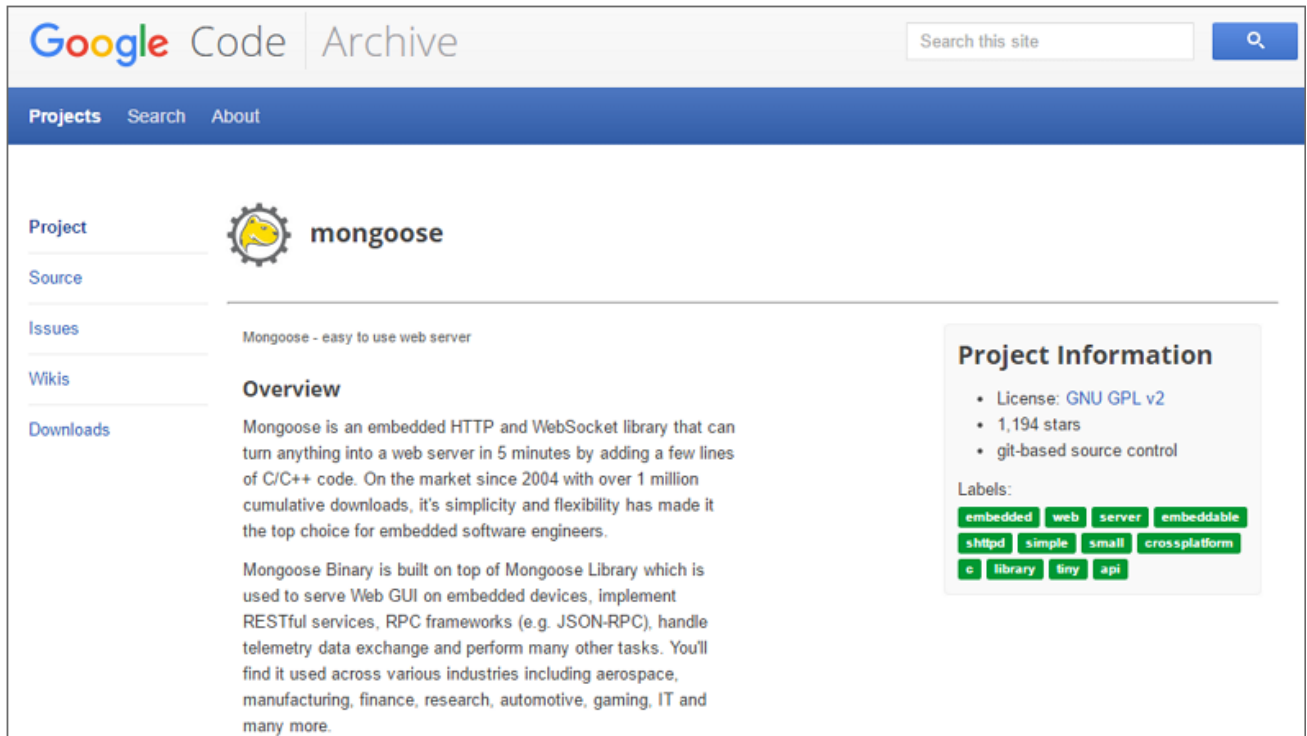
### 3.2.1 몽구스(mongoose)

넥사크로플랫폼 애플리케이션을 PC에서 간단하게 테스트해보기 위해 경량 웹서버인 몽구스를 사용합니다. 몽구스는 별도의 설치 과정 없이 웹서버 root로 동작할 위치에 실행 파일을 복사해서 실행하기만 하면 됩니다.

## 내려받기

아래 사이트에서 시스템 환경에 맞는 실행파일을 내려받을 수 있습니다. 윈도우 사용자라면 exe 확장자를 가진 파일을 내려받으면 됩니다.

<https://code.google.com/p/mongoose/>



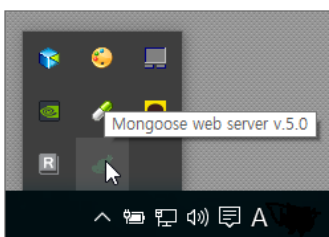
내려 받은 파일을 넥사크로 스튜디오에서 지정한 Generate Path에 가져다 놓습니다. 넥사크로 스튜디오에서 Options > Generate Path를 지정해주면 애플리케이션 자바스크립트 코드가 지정된 폴더에 생성됩니다.



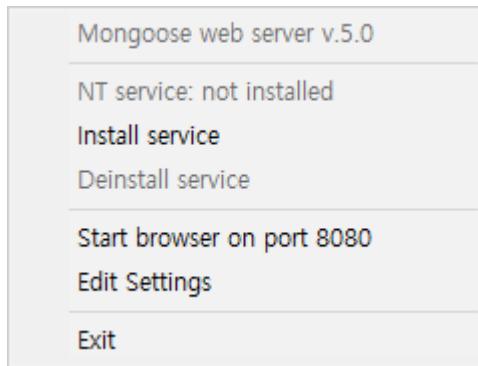
이번 장에서는 mongoose-5.0.exe 파일을 내려받아 사용했습니다. 해당 프로젝트 업데이트에 따라 일부 내용이 변경될 수 있습니다.

## 웹서버 실행

몽구스 실행 파일을 더블클릭해 실행해줍니다. 더블클릭하는 순간 몽구스 웹서버가 실행되고 아래와 같이 하단 윈도우 트레이 영역에 아이콘이 추가됩니다.

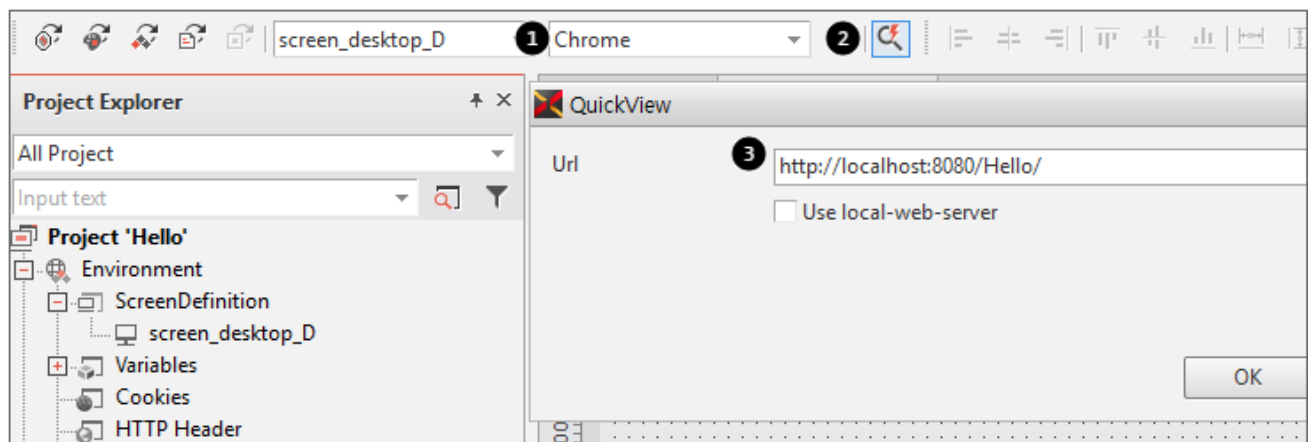


오른쪽 마우스를 클릭해 컨텍스트 메뉴를 열면 설정할 수 있는 메뉴가 보입니다. 이번 장에서는 웹서버 종료 시 사용하는 'Exit' 항목만 사용합니다. 기타 세부적인 설명은 몽구스 프로젝트 사이트의 설명을 참조하세요.



## 넥사크로플랫폼 애플리케이션 확인

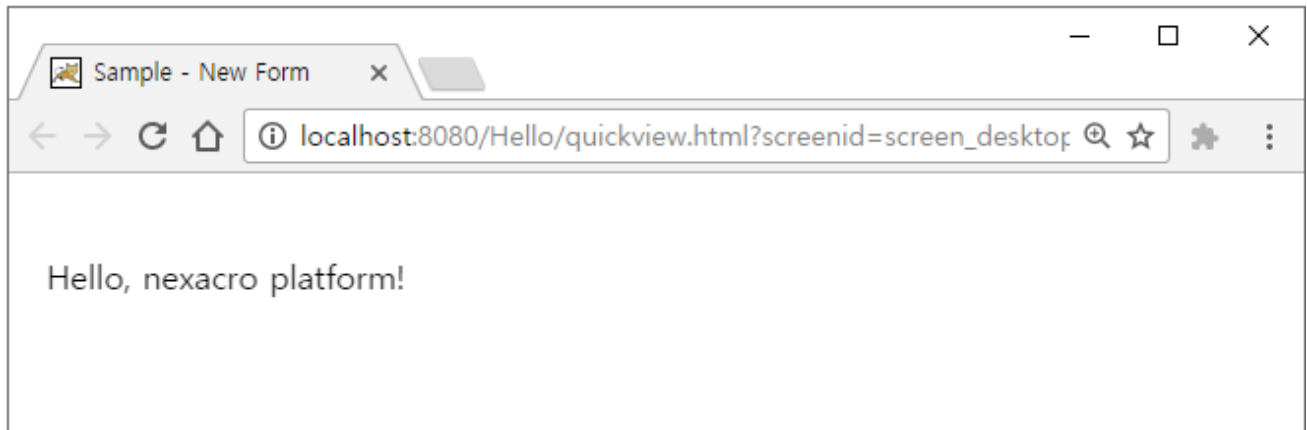
웹서버가 실행된 상태라면 넥사크로 스튜디오 [Build] 툴바에서 원하는 ❶ 웹 브라우저를 선택하고 ❷ [QuickView] 아이콘을 클릭합니다. URL을 확인하는 창이 나타나는데 최초 실행 시에는 URL 값이 입력되어 있지 않습니다.



로컬 웹서버를 사용하는 경우에는 메뉴[Tools > Options > Project > Local Web Server] 항목에서 설정한 URL 값이 자동으로 입력되고 수정할 수 없습니다.

❸ URL 창에 들어갈 값은 아래와 같습니다. 값을 입력하고 [OK] 버튼을 클릭하면 선택한 웹 브라우저에 애플리케이션이 실행됩니다.

| http://localhost:8080/Hello/



## 3.3 웹 애플리케이션 서버 사용하기

### 3.3.1 JDK(Java SE Development Kit)

웹 애플리케이션 서버는 톰캣(Tomcat)이나 제티(Jetty), 윈스턴(Winstone)과 같이 무료로 공개된 것을 사용하거나 상용 제품을 사용할 수 있습니다. 여기에서는 간단하게 사용할 수 있는 톰캣을 설치하는 기준으로 설명하겠습니다.

톰캣은 자바 기반으로 만들어졌기 때문에 실행하기 위해서는 JDK(Java SE Development Kit) 환경이 설정되어 있어야 합니다. JDK는 무료로 제공되고 있으며 간단하게 설치할 수 있습니다.

#### 내려받기

아래 사이트에서 시스템 환경에 맞는 JDK를 내려받을 수 있습니다.

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>



이번 장에서는 jdk-9.0.1\_windows-x64\_bin.exe 파일을 내려받아 사용했습니다. JDK 버전이 다른 경우 설치 및 설정 내용이 다를 수 있습니다.



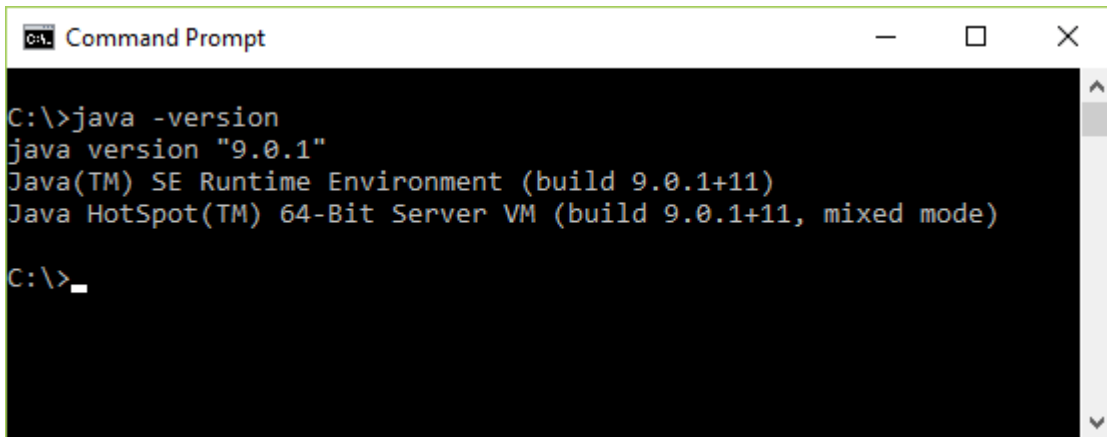
JDK나 톰캣 설치 파일은 윈도우 운영체제를 사용하는 경우 32비트와 64비트 설치 파일을 제공합니다. 운영체제 버전 확인은 아래 링크를 참고하세요.

<http://support.microsoft.com/kb/958406/ko>

## 설치 확인

내려받은 설치파일을 실행하면 자동으로 필요한 환경설정과 설치과정을 진행합니다. 설치가 완료되면 커멘드창에서 정상적으로 설치되었는지 확인합니다.

```
java -version
```



```

C:\>java -version
java version "9.0.1"
Java(TM) SE Runtime Environment (build 9.0.1+11)
Java HotSpot(TM) 64-Bit Server VM (build 9.0.1+11, mixed mode)
C:\>_

```



JDK 1.5 이상 버전을 설치하는 경우에는 실행에 필요한 환경 변수(JAVA\_HOME)를 따로 설정해주지 않아도 시스템 경로에 자동으로 실행 파일을 복사해 접근할 수 있습니다.

### 3.3.2 아파치 톰캣(Apache Tomcat)

넥사크로플랫폼 애플리케이션을 웹브라우저에서 동작하게 하려면 웹서버 환경을 만들어주어야 합니다. 아파치 톰캣은 간단한 설치만으로 이런 환경을 만들 수 있습니다.

## 내려받기

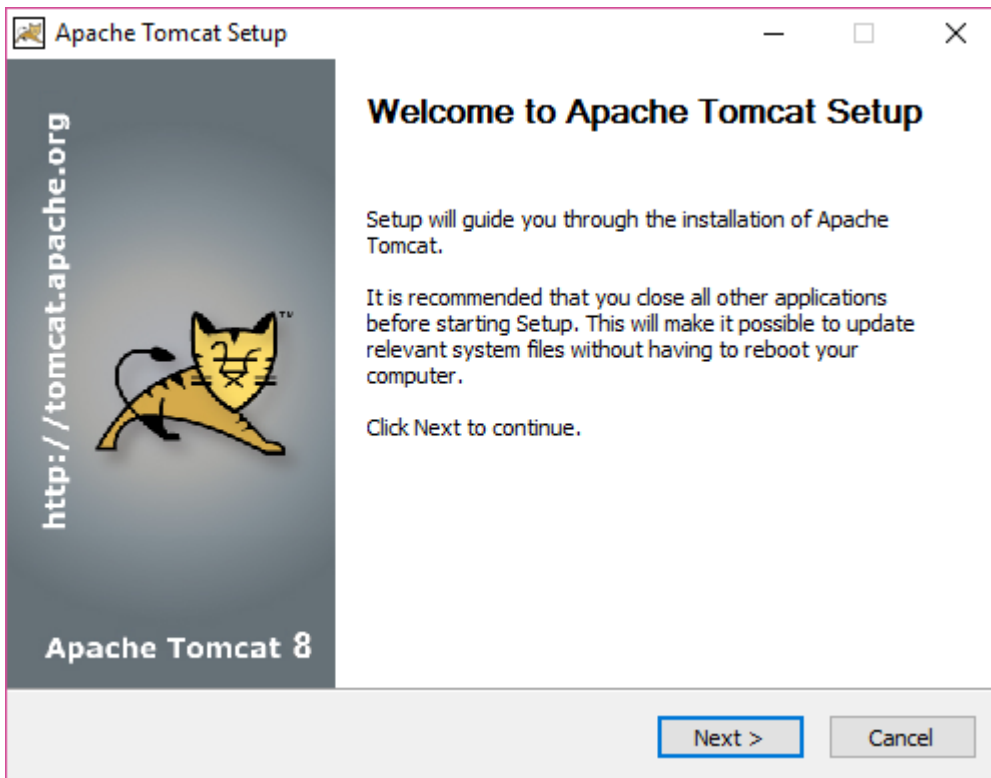
아래 사이트에서 실행 환경에 맞는 톰캣을 내려받을 수 있습니다.

<http://tomcat.apache.org>



톰캣은 압축된 바이너리 파일을 내려받아 직접 실행하거나 별도로 제공되는 인스톨 파일을 사용할 수 있습니다. 여기에서는 인스톨 파일로 설치하도록 하겠습니다.

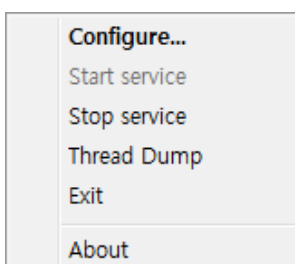
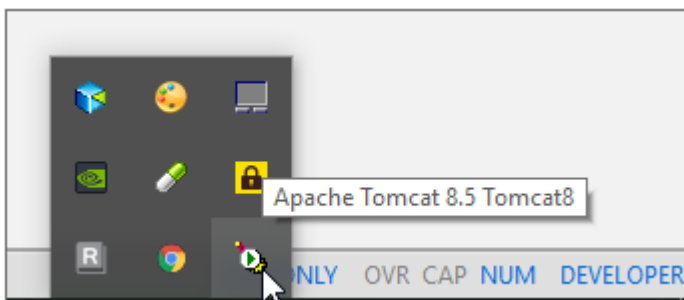
내려받은 파일을 실행하게 되면 설치 마법사가 필요한 설정 과정을 진행하며 JRE 경로도 자동으로 인식해 처리해줍니다.



이번 장에서는 apache-tomcat-8.5.23.exe 파일을 내려받아 사용했습니다. 해당 프로젝트 업데이트에 따라 일부 내용이 변경될 수 있습니다.

## 설치 확인

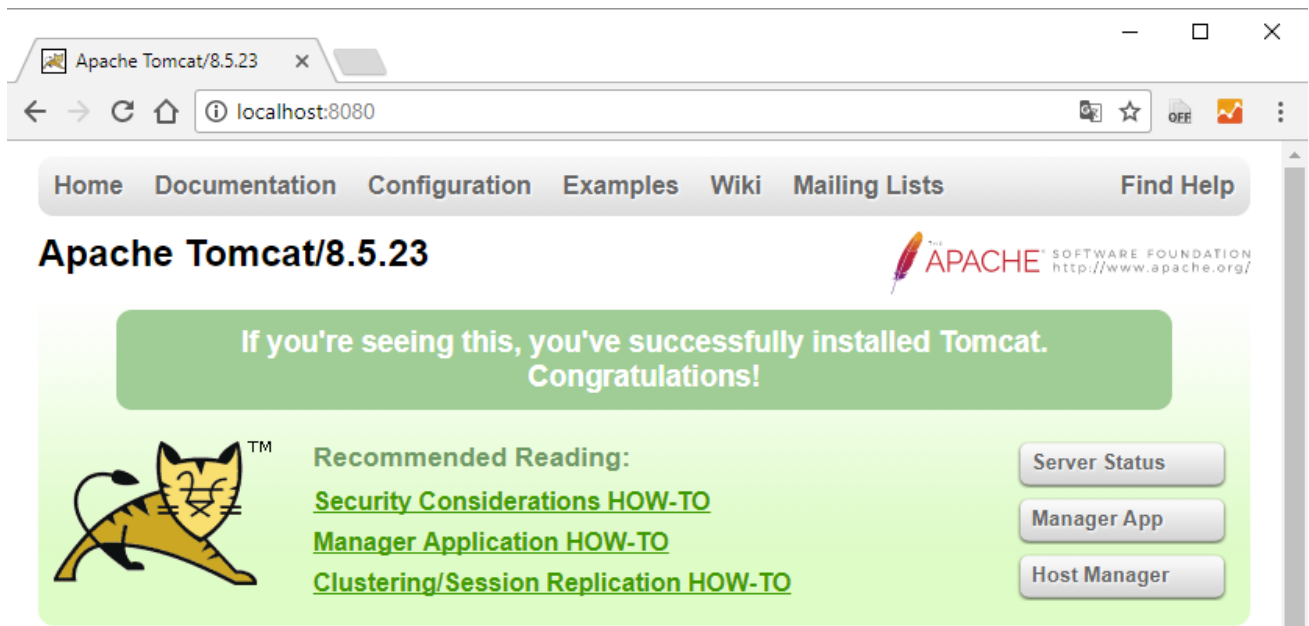
설치가 완료되면 윈도우 서비스로 등록되면서 자동으로 서버를 실행시켜줍니다. 윈도우 하단 트레이에서 오른쪽 마우스를 클릭해 컨텍스트 메뉴를 확인하면 서비스 상태 정보를 확인하거나 다른 설정을 할 수 있습니다.





웹브라우저에서 로컬 주소에 설정된 URL값을 입력해 정상 설치 여부를 확인합니다.

http://localhost:8080



## 컨텍스트 설정

톰캣에서 애플리케이션을 실행하기 위해서는 컨텍스트를 추가해주어야 합니다. 컨텍스트를 추가하는 방법은 몇 가지가 있지만, 이번 장에서는 컨텍스트 파일을 생성하고 임의의 폴더를 설정하는 방법을 살펴보겠습니다.



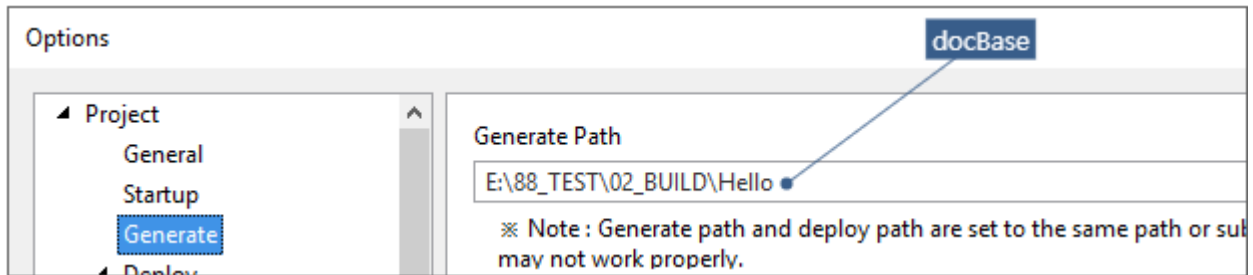
새로운 파일을 추가하거나 수정하기 위해 운영체제 설정에 따라 관리자 권한이 필요합니다.

## 컨텍스트 파일 생성

컨텍스트 파일은 넥사크로 스튜디오에서 Generate Path를 지정하고 해당 경로를 톰캣과 연결해주는 역할을 합니다. 톰캣이 설치된 경로 아래에 지정된 폴더에 XML 파일로 만든 애플리케이션 컨텍스트 파일을 추가해줍니다.

C:\Program Files\Apache Software Foundation\Tomcat 8.5\conf\Catalina\localhost

XML 파일을 아래와 같이 생성합니다. path 항목에는 URL 주소에 포함될 문자열을 입력하고 docBase 항목에는 Generate Path로 지정된 경로를 입력합니다. 아래와 같이 path 항목을 지정한 경우에 연결되는 URL은 http://localhost:8080/Hello/... 형식이 됩니다. 생성하는 파일명은 path 항목에 입력한 이름과 같아야 합니다. 아래 예제에서는 Hello.xml 파일을 생성합니다.



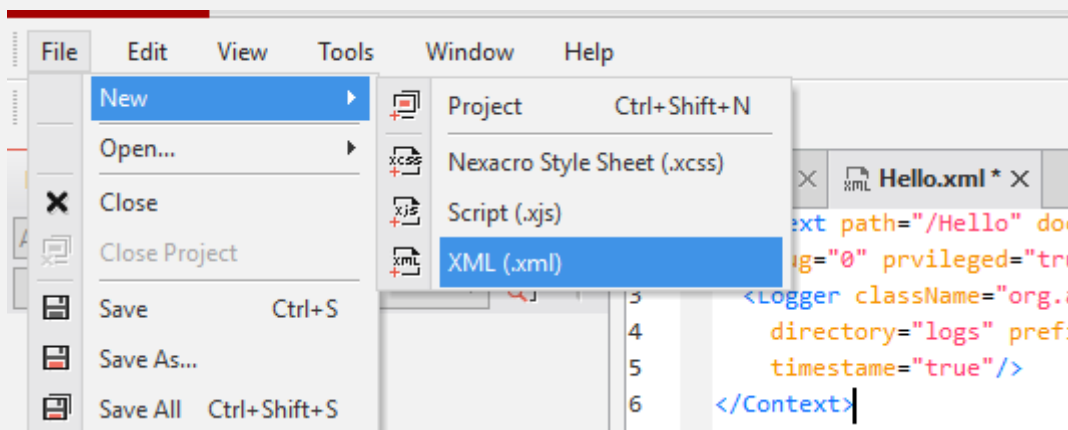
Hello.xml

```
<Context path="/Hello" docBase="E:\88_TEST\02_BUILD\Hello"
  debug="0" privileged="true" reloadable="true">
  <Logger className="org.apache.catalina.logger.FileLogger"
    directory="logs" prefix="localhost_log." suffix=".txt"
    timestamp="true"/>
</Context>
```

Foundation > Tomcat 8.5 > conf > Catalina > localhost		
Name	Date modified	Type
Hello.xml	10/26/2017 11:58 PM	XML Document



XML 파일은 윈도우 운영체제의 메모장이나 설치된 에디터 프로그램에서 새로 만들 수 있습니다. 넥사크로 스튜디오를 실행한 경우에는 넥사크로 스튜디오 메뉴에서 [File > New > XML(.xml)] 항목을 선택해 파일을 새로 만들 수 있습니다.

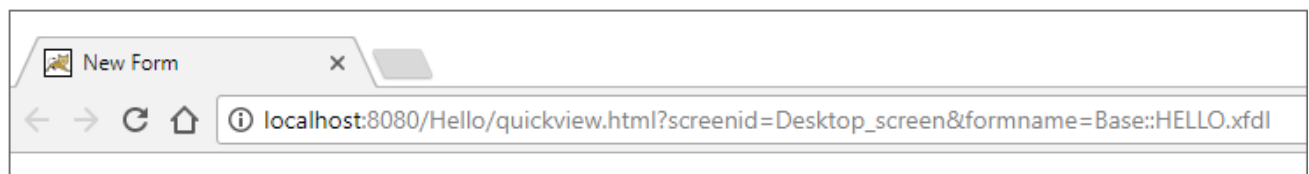
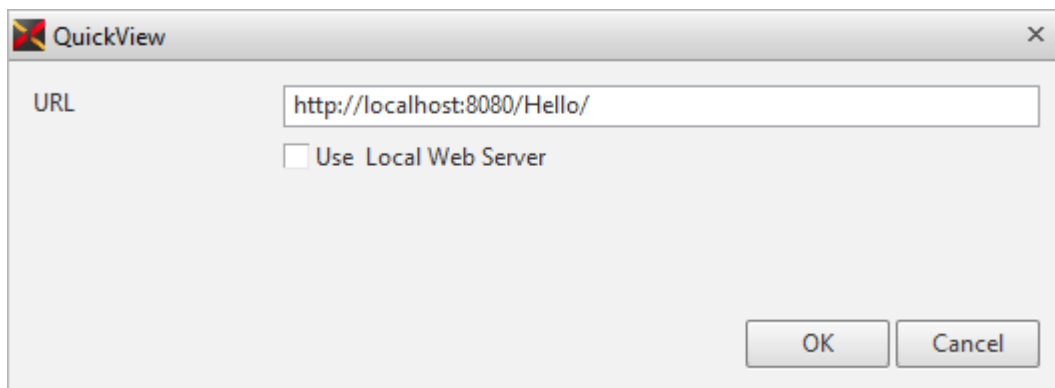




컨텍스트 파일은 프로젝트마다 따로 생성해주거나 하나의 컨텍스트 파일을 가지고 여러개의 프로젝트를 관리할 수도 있습니다. 이후 예제에서는 프로젝트마다 컨텍스트 파일을 생성하는 것으로 안내하고 있습니다.

## 넥사크로플랫폼 애플리케이션 확인

넥사크로 스튜디오에서 [QuickView] 메뉴 실행 시 URL 항목에 생성된 경로를 입력하면 넥사크로플랫폼 애플리케이션이 실행되는 것을 확인할 수 있습니다.



윈도우 트레이 영역에서 'Exit' 항목을 선택해주면 톡넷 서비스가 종료됩니다.

## 4.

# 화면 만들기 (고객사 목록 조회)

업무에 사용되는 애플리케이션 중 가장 기본적인 항목이 조회 화면입니다. 이번 장에서는 간단한 고객사 목록 조회 화면을 만들면서 그리드(Grid) 컴포넌트와 데이터셋(Dataset)을 다루는 방법을 살펴봅니다.

## 4.1 화면 구성

넥사크로플랫폼 애플리케이션을 만드는 일반적인 방법은 프로젝트와 폼을 만들고 폼 안에 필요한 컴포넌트를 배치하는 과정입니다.

### 4.1.1 프로젝트(Project)

아래와 같이 새로운 넥사크로플랫폼 프로젝트를 만듭니다.

[Menu] File > New > Project

항목	설정값	설명
Name	CustomerList	프로젝트 이름
Location	E:\W88_TEST\01_PROJECT\CustomerList	프로젝트 파일 저장 위치

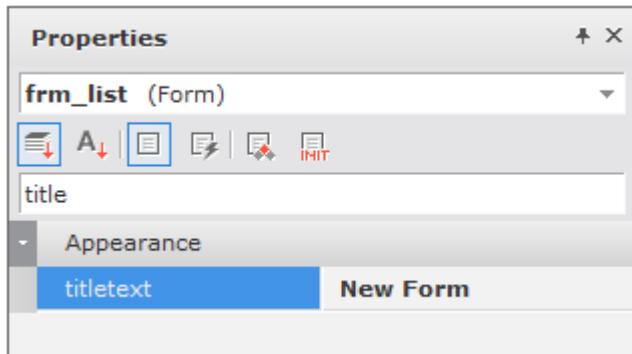
### 4.1.2 폼(Form)

만들어진 프로젝트에 고객사 목록을 표시할 폼을 추가합니다. 아래와 같이 새로운 폼을 만듭니다. 폼의 너비와 높이는 폼을 만든 후에 속성 창에서 수정할 수 있습니다.

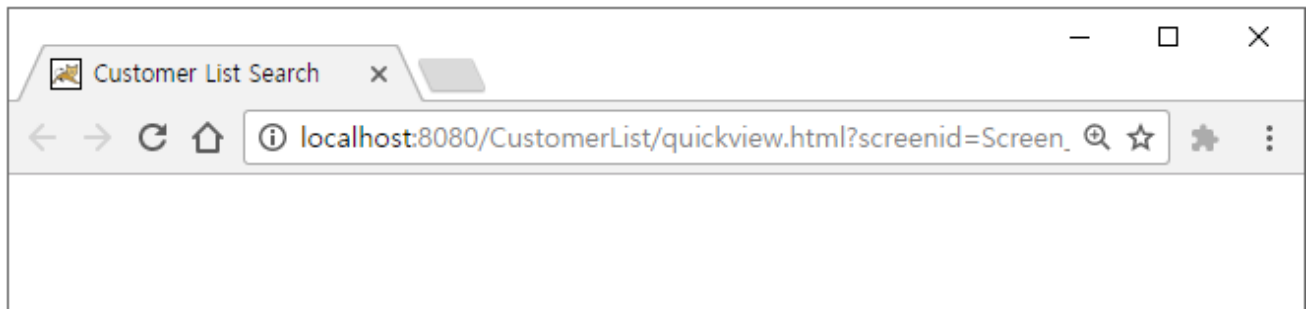
[Menu] File &gt; New &gt; Form

항목	설정값	설명
Name	frm_list	폼 이름
Location	Base	서비스 그룹 (기본값은 Base)
Width	800	폼 너비 (기본값은 1024)
Height	600	폼 높이 (기본값은 768)

속성창에서 폼의 titletext 속성값을 수정합니다. titletext 속성값은 웹브라우저에 표시되는 제목으로 적용됩니다.



속성	값	설명
titletext	Customer List Search	폼 제목



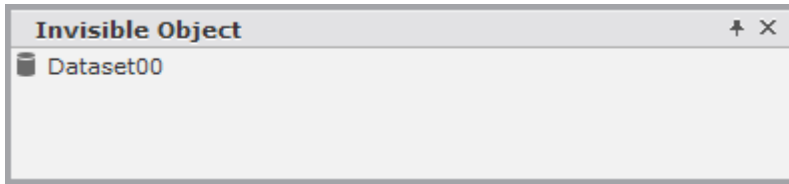
### 4.1.3 데이터셋(Dataset)

데이터셋 컴포넌트는 넥사크로플랫폼 애플리케이션 내에서 데이터를 관리하는 기능을 제공합니다. 사용자가 입력하거나 서버에서 가져온 데이터를 보관하고 데이터의 변경(추가/수정/삭제) 상태를 관리합니다.

데이터셋은 2차원 테이블 형태로 데이터를 관리합니다. 컬럼(Column) 구조로 로우(Row, Record) 단위로 데이터를 다룰 수 있습니다. 하나의 폼에는 여러 개의 데이터셋을 가질 수 있으며 GlobalVariables로 데이터셋을 만들어 여러 폼에서 공동으로 사용할 수도 있습니다.

넥사크로 스튜디오 툴바에서 데이터셋  을 선택한 상태에서 디자인 영역 아무 곳이나 클릭하면 데이터셋이 추가됩니다.

니다. 데이터셋은 화면에 보이는 컴포넌트가 아니라 애플리케이션 내부에서 데이터 관리를 위해 사용되는 컴포넌트이기 때문에 Invisible Objects 항목으로 관리됩니다.



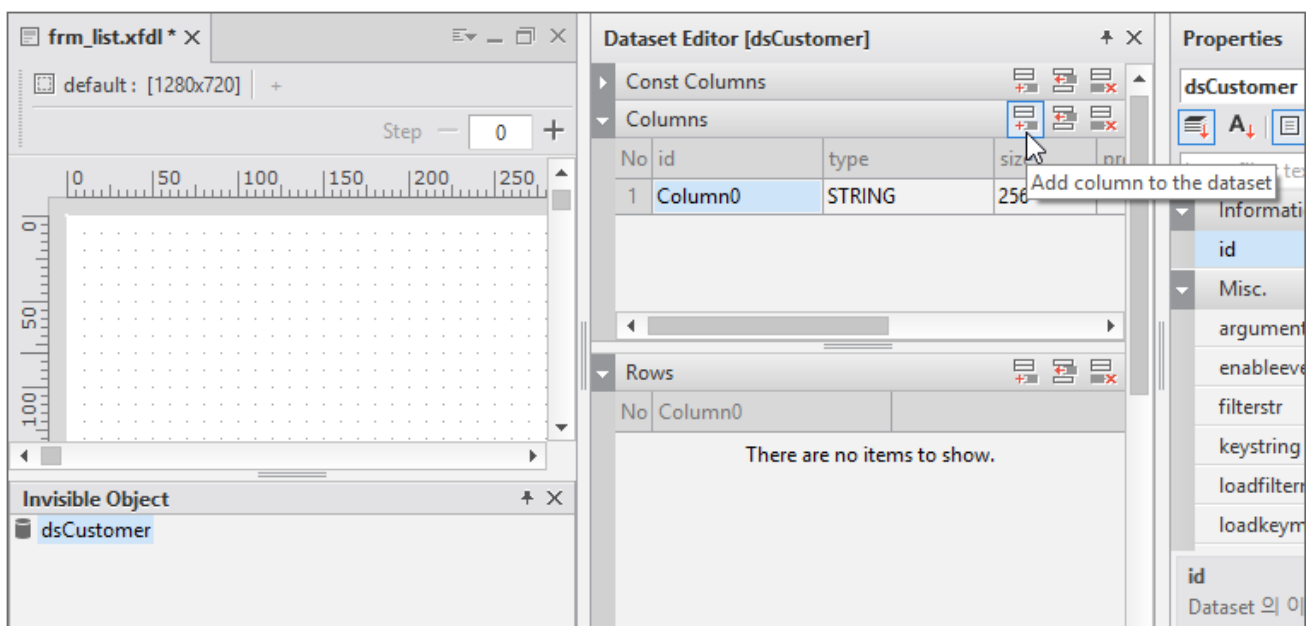
속성창에서 추가한 데이터셋 속성을 아래와 같이 수정합니다.

속성	값	설명
id	dsCustomers	데이터셋 id

## 4.1.4 데이터셋 콘텐츠 에디터(Dataset Contents Editor)

일반적인 컴포넌트는 속성창에서 관련된 속성을 수정하지만 데이터셋은 데이터를 다루기 위한 추가적인 도구로 데이터셋 콘텐츠 에디터가 제공됩니다.

데이터셋 콘텐츠 에디터는 2차원 테이블 형태로 데이터를 관리하기 위해 컬럼을 디자인하고 데이터를 직접 추가할 수 있는 기능을 제공합니다. 데이터셋을 더블클릭하면 데이터셋 콘텐츠 에디터가 나타납니다.



데이터셋에서 데이터를 관리하기 위해서는 사용할 컬럼을 추가해주어야 합니다. Columns 항목에서 [Add column to the dataset] 버튼을 클릭해 컬럼을 추가합니다.

예제에서 사용하는 컬럼은 아래와 같습니다.

No.	id	type	size	설명
1	id	STRING	4	일련번호
2	name	STRING	16	이름
3	email	STRING	32	이메일
4	phone	STRING	16	전화번호
5	comp_name	STRING	32	회사
6	department	STRING	32	부서
7	comp_phone	STRING	16	회사 전화번호
8	comp_addr	STRING	256	회사 주소

Rows 항목에서 테스트를 위한 데이터를 추가할 수 있습니다. 데이터를 추가하는 방법은 컬럼을 추가하는 방법과 같습니다.

추가된 항목은 데이터셋 콘텐츠 에디터에서 소스 코드로 확인할 수 있습니다. 소스 코드를 직접 수정해도 데이터셋 컬럼에 반영됩니다.



```

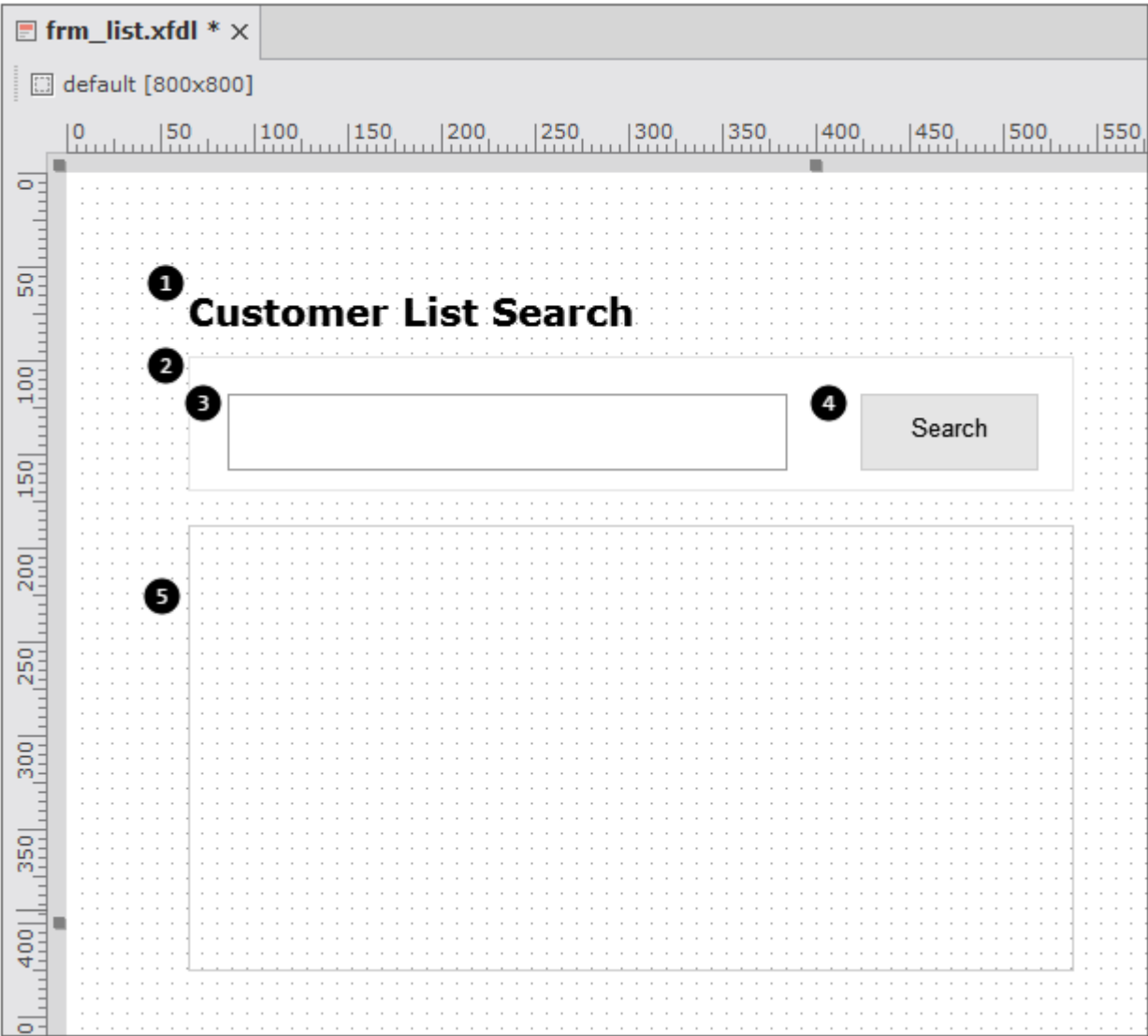
<ColumnInfo>
  <Column id="id" type="STRING" size="4"/>
  <Column id="name" type="STRING" size="16"/>
  <Column id="email" type="STRING" size="32"/>
  <Column id="phone" type="STRING" size="16"/>
  <Column id="comp_name" type="STRING" size="32"/>
  <Column id="department" type="STRING" size="32"/>

```

```
<Column id="comp_phone" type="STRING" size="16"/>
<Column id="comp_addr" type="STRING" size="256"/>
</ColumnInfo>
```

### 4.1.5 컴포넌트 배치





고객사 목록 조회 화면은 제목, 검색창, 검색버튼, 그리드로 구성됩니다. 전체적인 화면 구성은 아래와 같습니다.



화면 구성에 사용한 컴포넌트와 수정한 속성값은 아래와 같습니다.

컴포넌트	속성	값	설명
① Static	id	sttList	
	text	Customer List Search	표시할 문자열
	font	bold 20 Verdana	글꼴 속성



컴포넌트	속성	값	설명
② Div 	id	divCommand	
③ Edit 	id	edtSearch	
④ Button 	id	btnSearch	
	text	Search	버튼에 표시할 문자열
⑤ Grid 	id	grdCustomers	



Div 컴포넌트는 여러 개의 컴포넌트를 감싸주는 역할을 합니다. Div 컴포넌트를 먼저 배치하고 안에 들어갈 컴포넌트를 배치해야 합니다.

예제에서 Div 컴포넌트 안에 들어가는 Edit, Button 컴포넌트는 Div 컴포넌트를 먼저 화면에 배치하고 그 위에 컴포넌트를 추가해주어야 합니다.

## 4.2 그리드

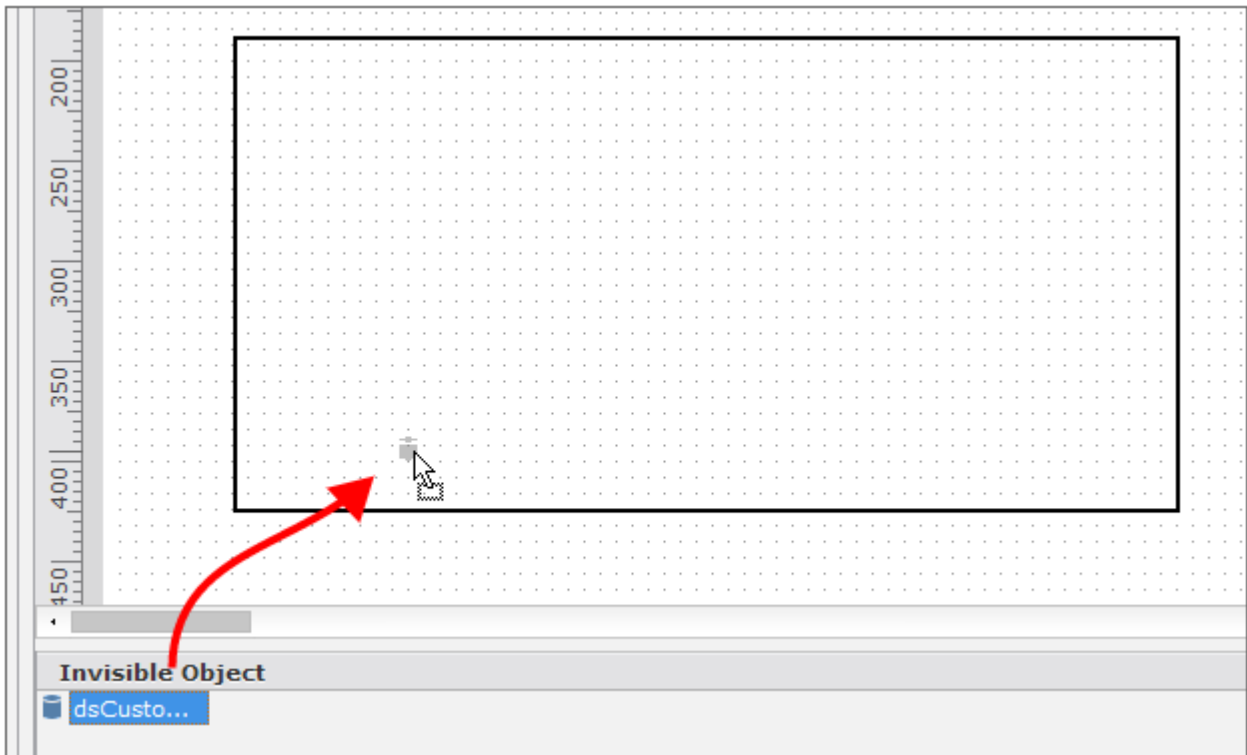
데이터를 다루는 화면을 구성한다면 가장 많이 사용하는 컴포넌트가 그리드입니다. 그리드는 컴포넌트 혼자서 사용하기보다는 데이터를 관리하는 데이터셋과 연결해 사용합니다.

### 4.2.1 데이터 바인딩

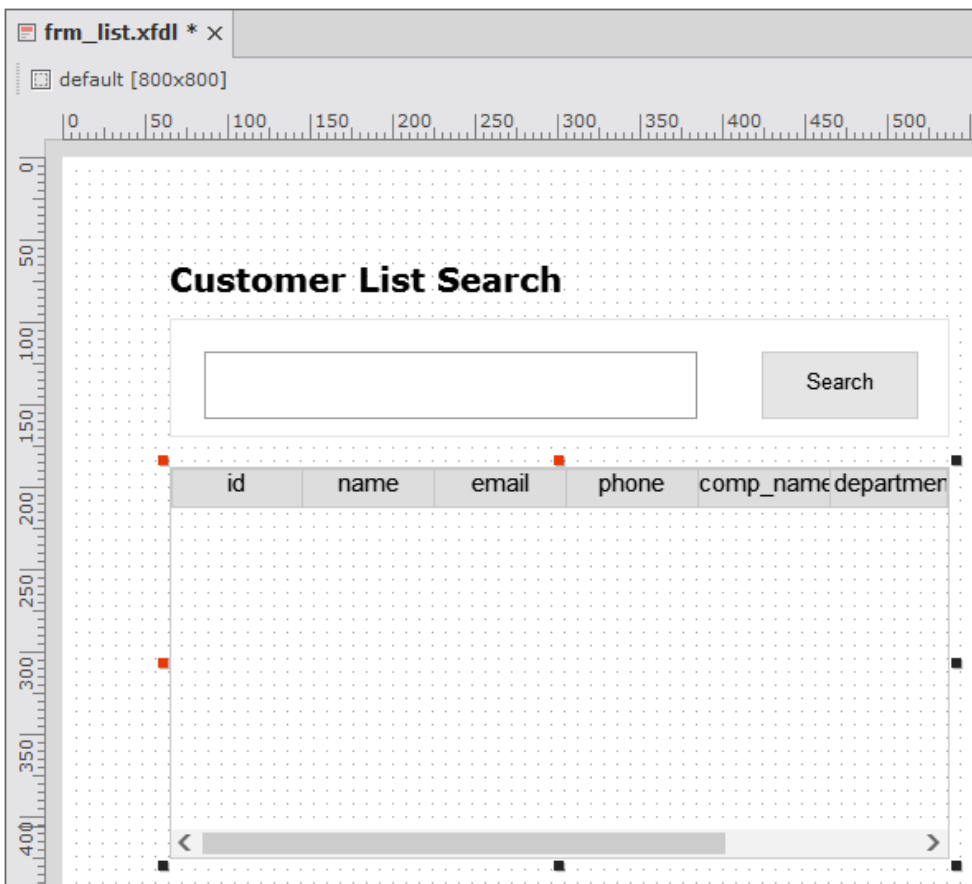
화면에 배치된 그리드에 데이터를 보여주기 위해서는 데이터셋을 연결해주어야 합니다. 이런 연결 작업을 데이터 바인딩(Binding)이라고 합니다. 그리드뿐 아니라 데이터를 화면에 보여줄 수 있는 모든 컴포넌트는 데이터셋과 연결할 수 있습니다.

그리드에 데이터셋을 연결하면 데이터를 보여주는 것뿐 아니라 그리드 내에서 직접 데이터를 수정하거나 입력할 수 있으며 변경된 데이터는 데이터셋에 바로 반영됩니다.

그리드에 데이터셋을 연결하는 방법은 binddataset 속성값에 데이터셋 id값을 지정해주거나 넥사크로 스튜디오 화면 위에서 데이터셋 컴포넌트를 마우스로 클릭하고 그리드 컴포넌트 위로 드래그해서 옮겨줍니다.

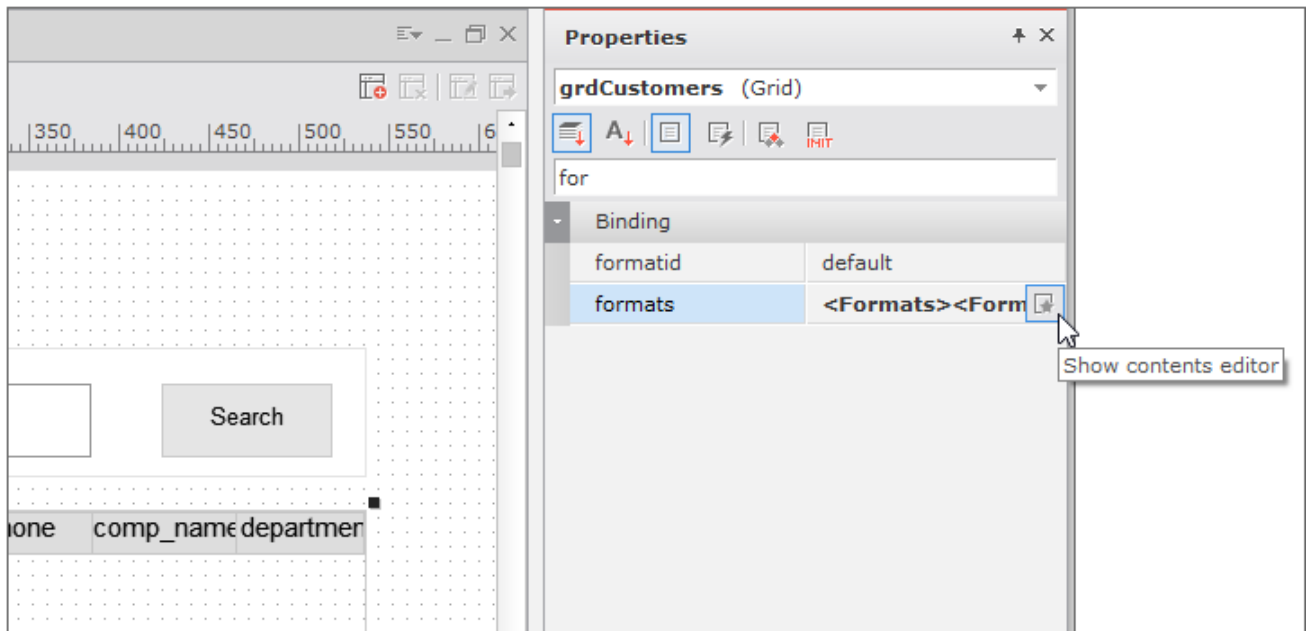


데이터셋을 드래그해서 그리드 컴포넌트 위에 가져다 놓으면 `binddataset` 속성값이 해당 데이터셋 id로 지정되면서 `formats` 속성에 데이터셋 컬럼을 기준으로 값이 만들어집니다. 그리고 화면 위에서도 데이터셋 컬럼을 기준으로 그리드 모양이 바뀐 것을 확인할 수 있습니다.



## 4.2.2 그리드 콘텐츠 에디터(Grid Contents Editor)

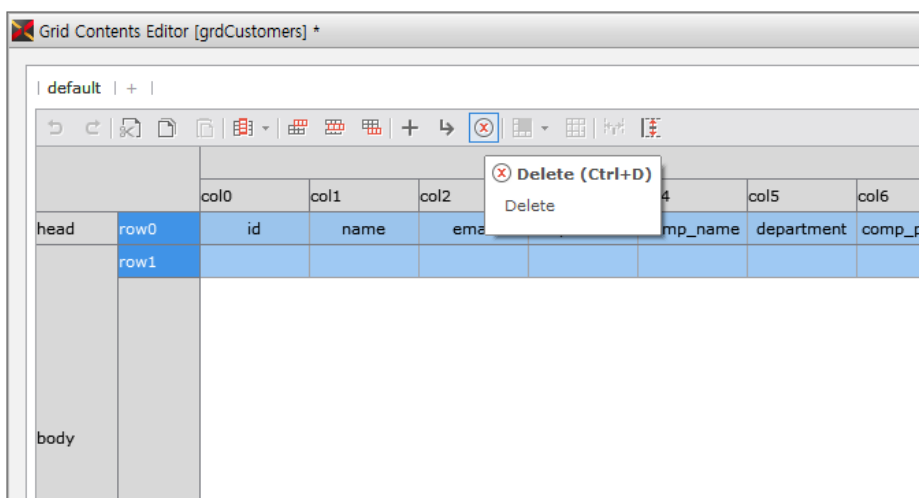
그리드와 데이터셋을 연결하면서 자동으로 formats 속성값이 만들어졌지만 다른 형식으로 데이터를 표기하기를 원한다면 그리드 콘텐츠 에디터에서 원하는 형식을 만들 수 있습니다. 그리드를 더블클릭하거나 속성창에서 formats 속성과 연결된 버튼을 클릭하면 그리드 콘텐츠 에디터가 실행됩니다.



그리드 콘텐츠 에디터는 그리드에 데이터를 어떻게 보여줄지를 편집할 수 있는 공간입니다. 데이터를 보여주는 형식만 다루며 실제 데이터셋에 영향을 주지는 않습니다.

### ① Dataset 바인딩 시 자동 생성된 포맷 삭제하기

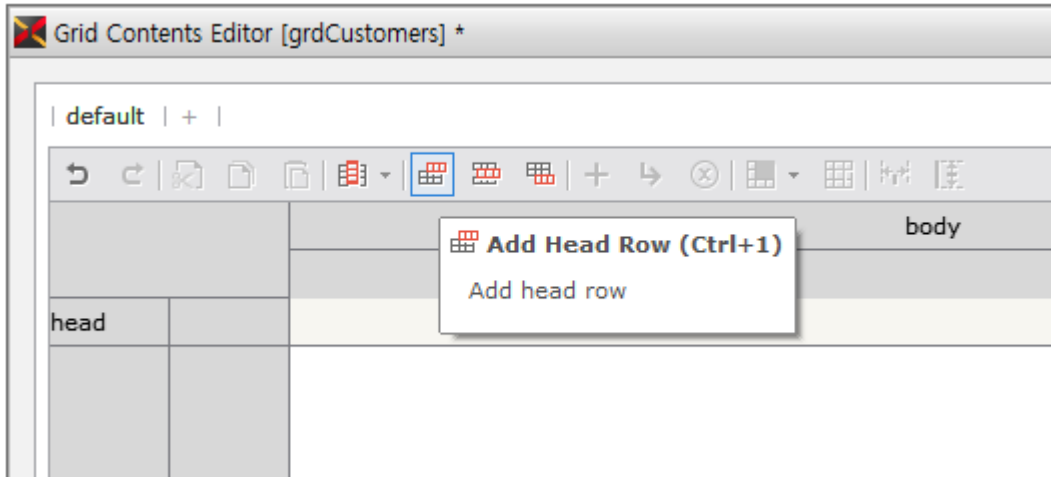
새로운 형식을 편집하기 위해 기존 설정된 내용을 삭제합니다. 그리드 콘텐츠 에디터에서 row0 항목을 선택하고 키보드 Shift 키 또는 Ctrl 키를 누른 상태에서 row1 항목을 선택하고 에디터 상단 툴바에서 Delete를 선택해 자동 생성된 포맷을 삭제합니다.



8개 컬럼을 2개의 행에서 보여주도록 수정합니다. 수정된 포맷은 4개의 컬럼을 가지고 컬럼에 따라 2개의 머리(Head) 행과 본문(Body) 행을 가집니다.

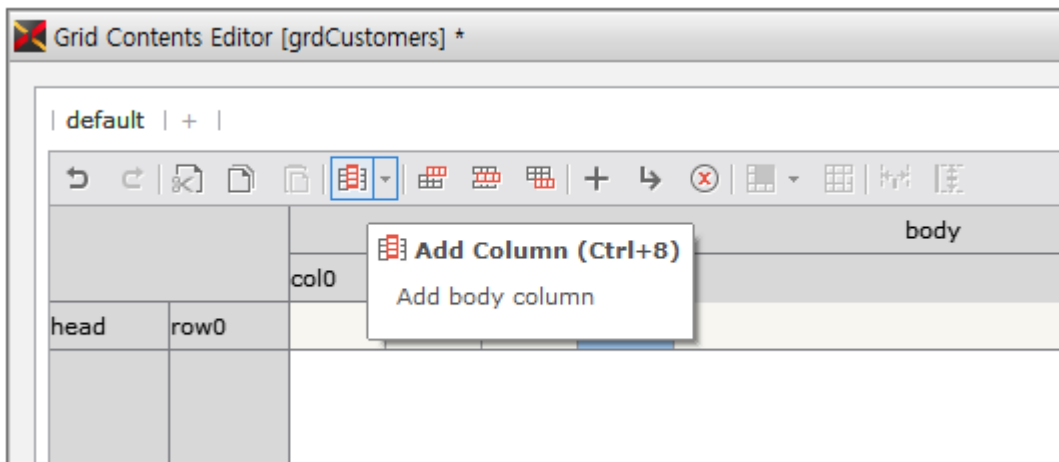
## 2 Head Row 추가

먼저 머리 행을 추가합니다. 그리드 콘텐츠 에디터 상단 툴바에서 [Add Head Row] 항목을 선택합니다.



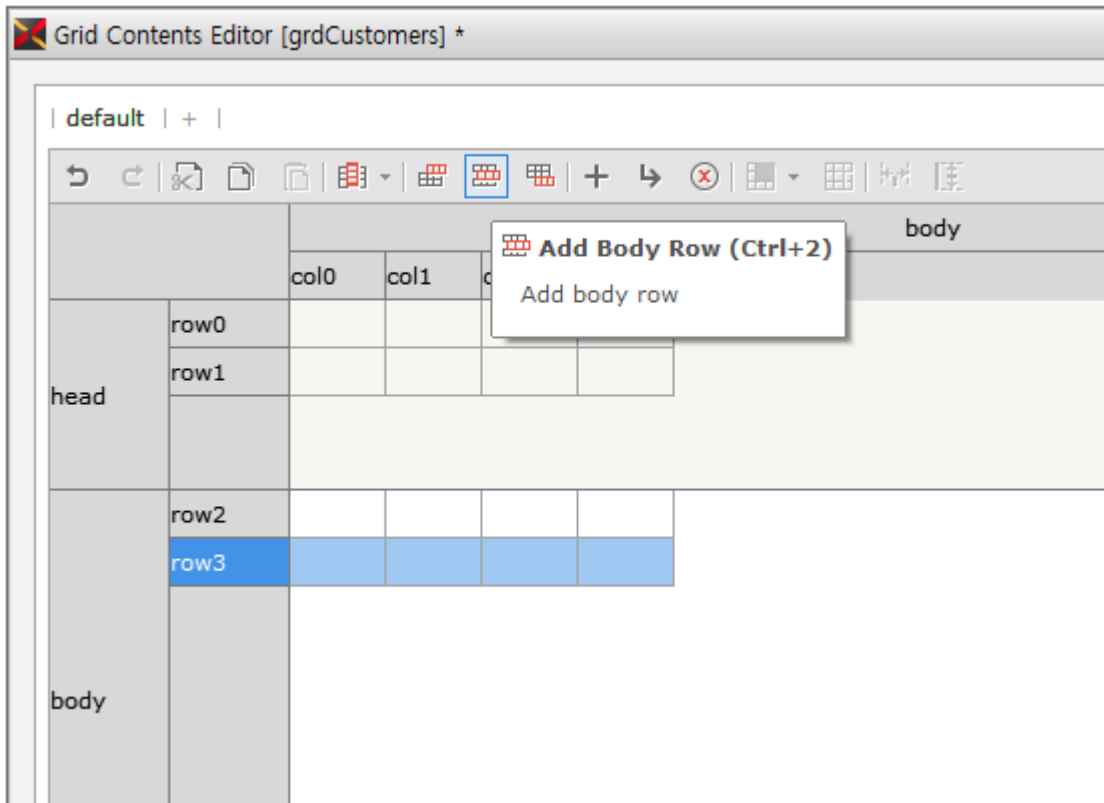
## 3 Column 4개 추가

그리고 4개의 컬럼을 추가합니다. 머리 행을 추가하고 나면 기본적으로 1개의 컬럼이 만들어지면서 Add Column 항목이 활성화됩니다. Add Column 항목을 선택해 3개의 컬럼을 추가합니다.



## 4 Head Row, Body Row 추가

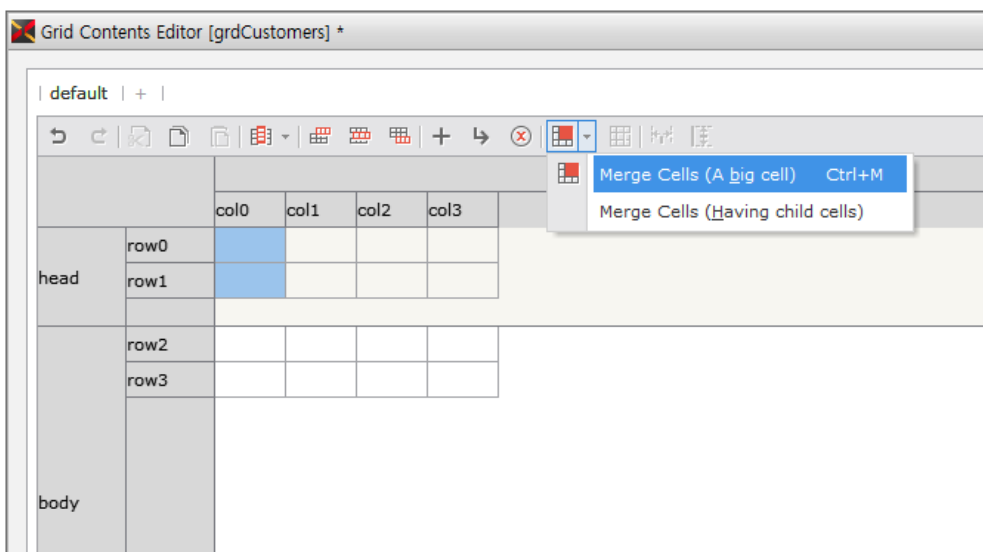
col2와 col3 컬럼에 2가지 데이터를 표기하기 위해 머리 행과 본문 행을 2개씩 추가합니다. 본문 행은 Add Body Row 항목을 선택해 추가할 수 있습니다.



#### 5 Merge Cells

col0과 col1 컬럼은 하나의 데이터만 연결해주기 때문에 셀을 합쳐주어야 합니다. col0의 머리 행에 있는 셀을 선택하고 에디터 툴바에서 [Merge Cells] 항목을 선택합니다. 여러 개의 셀을 선택할 때는 하나의 셀을 선택한 후에 마우스를 드래그하거나 Shift키를 누른 채 다른 셀을 선택합니다.

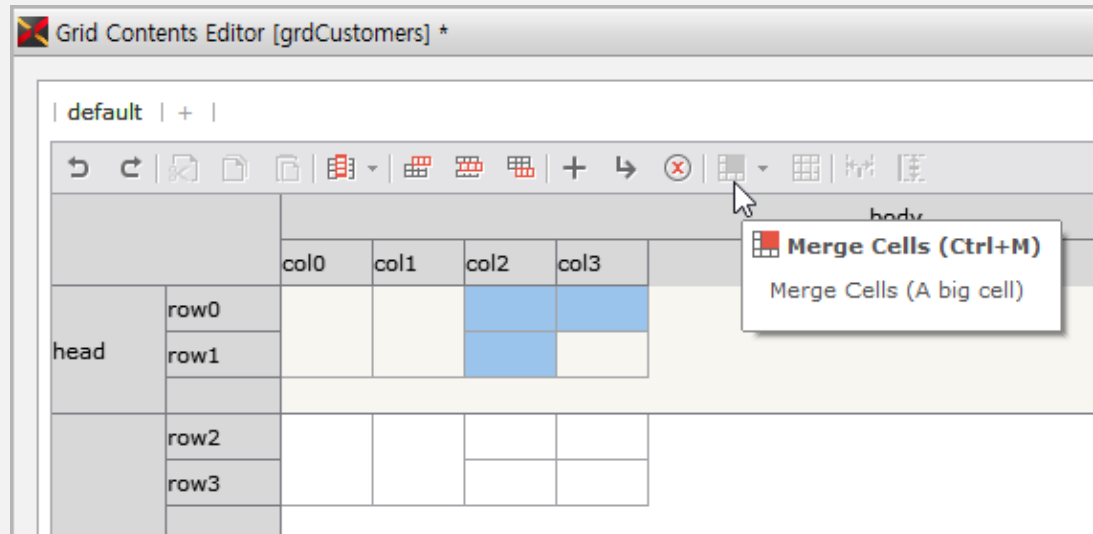
[Merge Cells] 항목은 2가지 옵션이 있습니다. [Merge Cells(A big cell)]은 각 셀 속성을 모두 삭제하고 하나의 셀로 합치는 것이고 [Merge Cells(Having child cells)]는 개별 셀 속성을 그대로 유지한 상태에서 하나의 셀처럼 보이도록 합치는 것입니다. 이번 예제에서는 각 셀 속성을 유지할 필요가 없으니 [Merge Cells(A big cell)] 항목을 선택합니다.



col0, col1의 본문 행의 셀도 하나로 합쳐줍니다.



Ctrl키를 누른 상태에서 합쳐질 셀을 선택할 수 있습니다. 하지만 컬럼과 행이 사각형 형태로 선택되지 않은 경우에는 [Merge Cells] 메뉴가 활성화되지 않습니다. 예를 들어 아래의 경우는 합치기 기능을 처리할 수 없습니다.

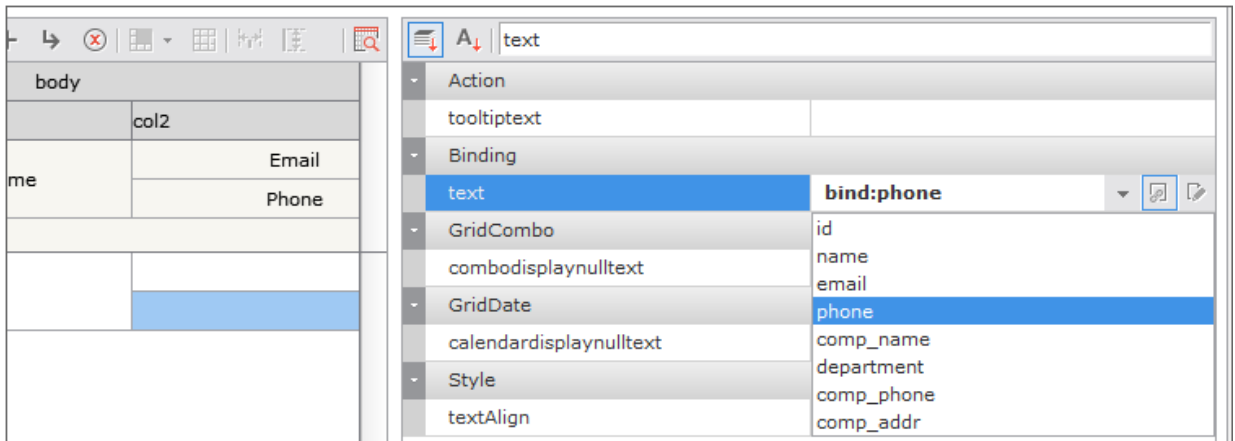


#### 6 컬럼에 바인딩하는 항목 지정

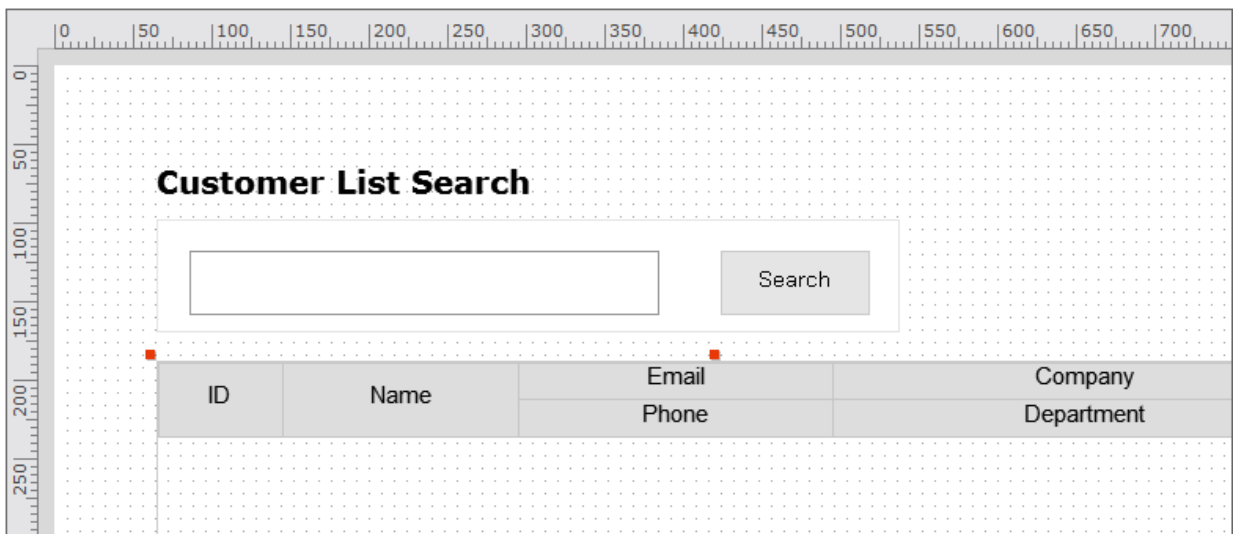
아래와 같이 각 컬럼에 해당하는 값을 지정합니다. 해당하는 셀을 선택하고 오른쪽에 보이는 속성 창에서 값을 입력할 수 있습니다.

컬럼	Head Cell: text	Body Cell: text	Column: size
col0	ID	bind:id	80
col1	Name	bind:name	150
col2	Email	bind:email	200
	Phone	bind:phone	
col3	Company	bind:comp_name	320
	Department	bind:department	

Grid 컴포넌트와 바인딩된 Dataset이 있는 경우 Dataset에 설정된 컬럼 목록을 에디터에서 확인할 수 있습니다. text 속성에 직접 "bind:id"라는 값을 입력하거나 목록에서 "id" 항목을 선택할 수 있습니다.



값을 입력하고 OK 버튼을 클릭하면 창이 닫히면서 그리드에 수정된 컬럼 형식이 반영됩니다.



## 4.3 데이터 테스트

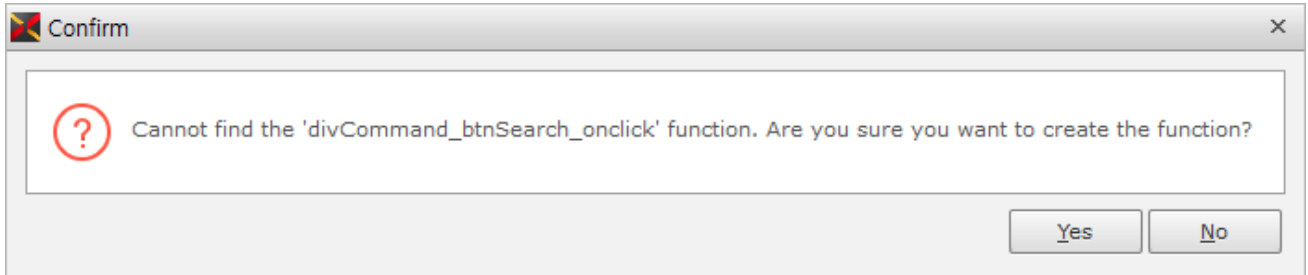
서비스 운영 단계에서는 데이터베이스에 있는 데이터를 가져와 화면에 보여주지만, 테스트 단계에서는 기능 구현 여부만 확인할 때도 있습니다. 간단한 스크립트로 데이터를 데이터셋에 추가해주고 연결된 그리드에 데이터를 표시합니다.

### 4.3.1 버튼 클릭 이벤트

btnSearch 버튼에 클릭 이벤트를 추가하고 데이터를 처리하는 스크립트를 작성합니다. btnSearch 버튼 컴포넌트를 선택하고 속성창에서 onclick 이벤트를 추가합니다. 이벤트 속성값을 추가하고 속성창을 더블클릭하거나 엔터키를

입력하면 해당 이벤트 함수를 생성합니다.

속성	값	설명
onclick	divCommand_btnSearch_onclick	버튼 클릭 이벤트



이벤트 스크립트는 아래와 같습니다. 데이터셋에 1건의 데이터 행을 추가하고 값을 지정해줍니다.

```
this.divCommand_btnSearch_onclick = function(obj:nexacro.Button,e:nexacro.ClickEventInfo)
{
    var row = this.dsCustomers.addRow();
    this.dsCustomers.setColumn(row, "id", "TC-001");
    this.dsCustomers.setColumn(row, "name", "Tzuyu");
    this.dsCustomers.setColumn(row, "email", "ceo@twice.com");
    this.dsCustomers.setColumn(row, "phone", "6987-6543");
    this.dsCustomers.setColumn(row, "comp_name", "TWICE");
    this.dsCustomers.setColumn(row, "department", "0");
    this.dsCustomers.setColumn(row, "comp_phone", "6506-7000");
    this.dsCustomers.setColumn(row, "comp_addr", "Seoul");
};
```



onclick 이벤트 처리 시 `TypeError`가 발생한다면 컴포넌트나 오브젝트의 id를 잘못 입력했을 수 있습니다. 예를 들어 예제에서 `this.dsCustomer.addRow()` 라고 입력한 경우 아래와 같은 오류가 발생합니다.

`TypeError: Cannot read property 'addRow' of undefined`

## 4.3.2 Generate

이제 만들어진 애플리케이션을 웹브라우저에서 실행합니다. 먼저 작성된 코드를 자바스크립트 파일로 변환해줍니다. Output 창에서 정상적으로 자바스크립트 파일이 생성되고 있는지 확인할 수 있습니다.




[Menu] Generate > Application

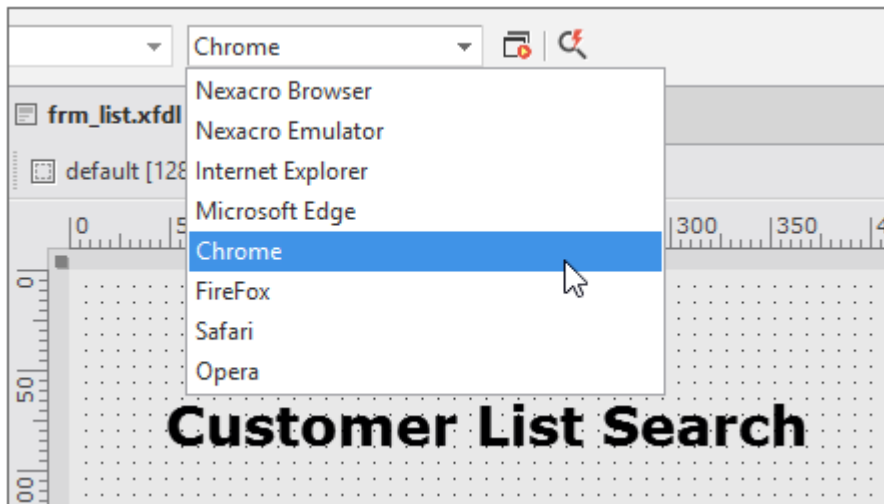
```

Output
nexacrogenerator 1> ----- Start to generator -----
nexacrogenerator 1> This file is already generated : "E:\88_TEST\02_BUILD\CustomerList\environment.xml.js"
nexacrogenerator 1> This theme is already generated : "E:\88_TEST\02_BUILD\CustomerList\_theme\_default"
nexacrogenerator 1> This file is already generated : "E:\88_TEST\02_BUILD\CustomerList\CustomerList.xadl.js"
nexacrogenerator 1> This file is already generated : "E:\88_TEST\02_BUILD\CustomerList\Base\frm_list.xfdl.js"
nexacrogenerator 1> ===== End to generator ( 1.22 sec ) : Success 0, Fail 0, Copy 0, Skip 4 =====

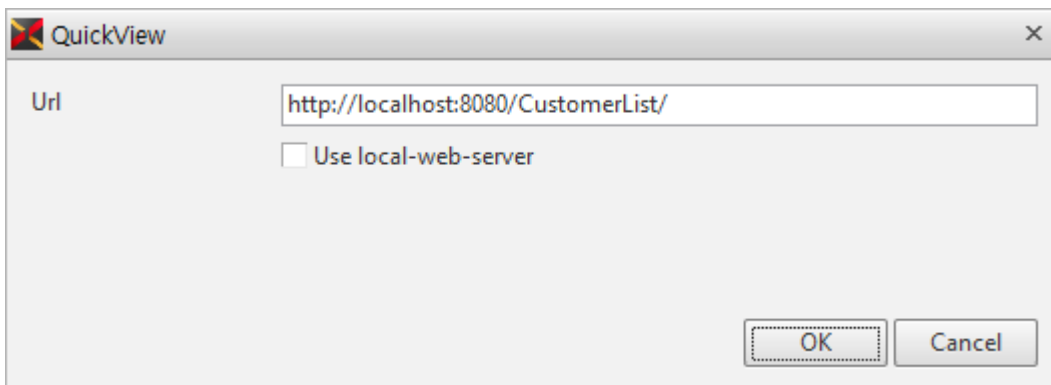
```

### 4.3.3 Quick View

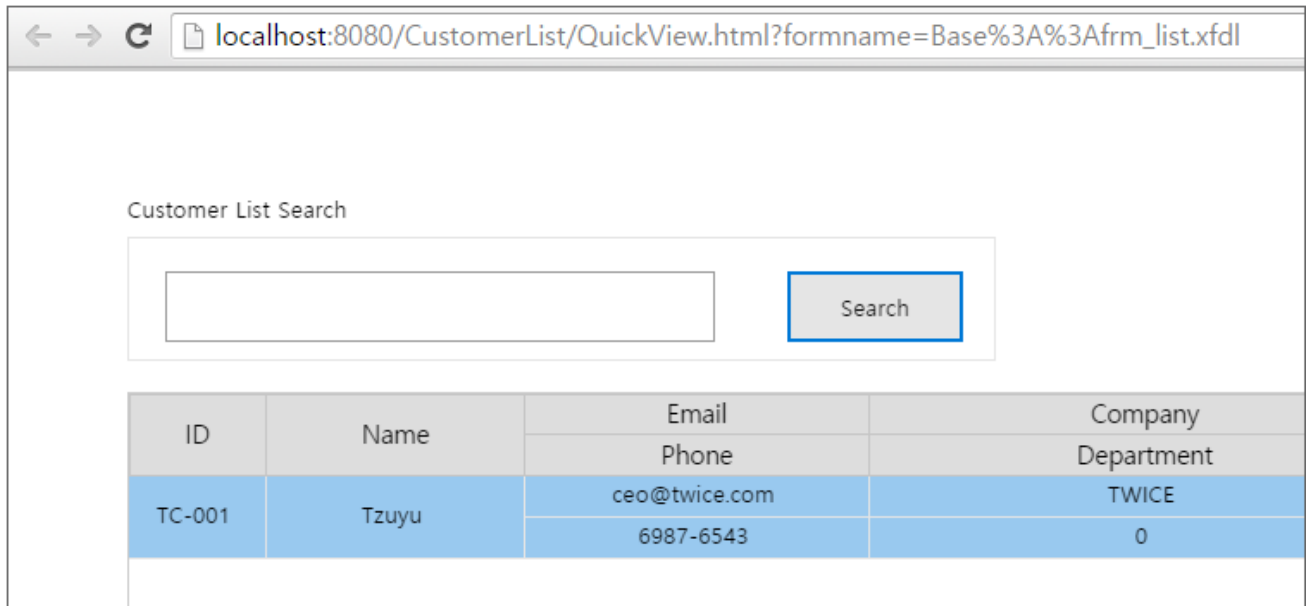
웹서버를 실행하고 넥사크로 스튜디오에서 Quick View 를 실행합니다. 웹브라우저에서 Quick View를 실행하려면 빌드 툴바를 활성화하고 브라우저 옵션을 변경해주어야 합니다. 웹브라우저 옵션에는 PC에 설치된 웹브라우저를 선택할 수 있습니다.



처음 웹브라우저로 Quick View를 실행하게 되면 Generate 시 생성된 생성된 파일이 있는 경로를 입력해야 합니다.



지정된 웹브라우저가 실행되면서 실행된 넥사크로플랫폼 애플리케이션을 확인할 수 있습니다. Search 버튼을 클릭하면 스크립트에서 추가한 데이터가 그리드에 표시됩니다.



Customer List Search

ID	Name	Email	Company
		Phone	Department
TC-001	Tzuyu	ceo@twice.com	TWICE
		6987-6543	0

## 5.

# 화면 만들기 (트랜잭션)

업무에 사용되는 애플리케이션은 특별한 경우를 제외하고는 데이터를 개별 PC에서 관리하지 않습니다. 데이터를 관리하는 서버에서 조회된 데이터를 애플리케이션에서 편집하고 그 결과를 다시 데이터 서버에 관리하도록 합니다.

이번 장에서는 넥사크로플랫폼 애플리케이션에서 코드 데이터를 어떻게 다루고 서버에 있는 데이터를 어떤 식으로 가져오는지 살펴보고 웹브라우저에서 로그를 확인하는 방법까지 알아봅니다.

## 5.1 그리드 콤보

데이터베이스에 관리되는 항목 중 국가, 부서, 직급과 같은 데이터는 여러 번 반복해서 사용되는 항목입니다. 이런 데이터를 문자열 그대로 입력하면 입력하는 사용자마다 다르게 입력할 수도 있고 데이터베이스에서 불필요하게 공간을 차지할 수 있습니다. 그래서 이런 데이터는 별도의 코드 테이블을 만들어 관리합니다.

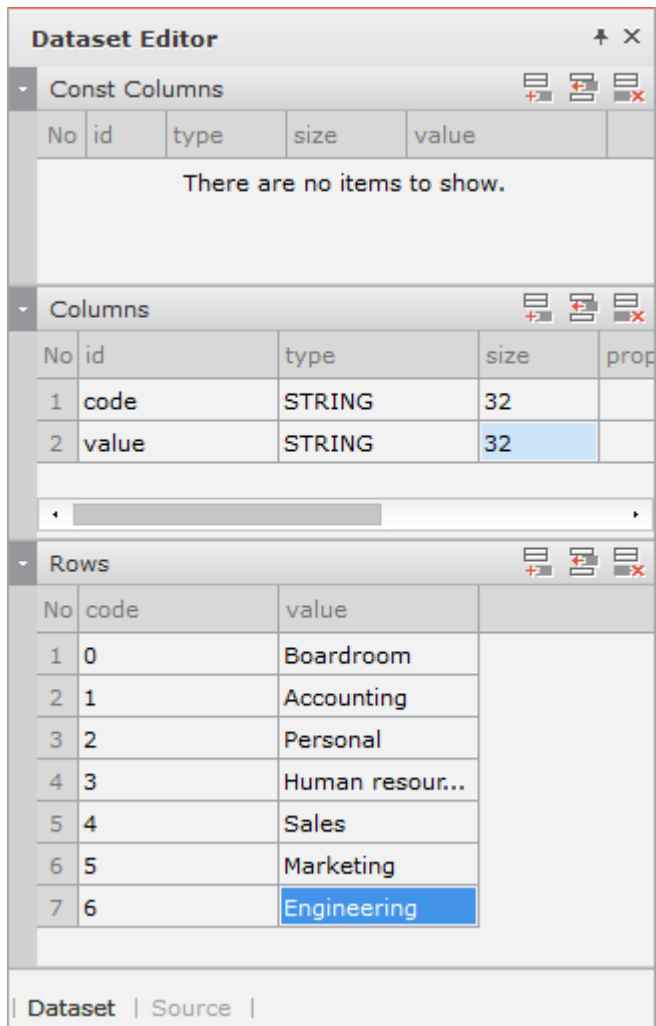
예제에서도 Department 항목은 부서 이름이 아닌 코드만 데이터베이스에서 관리하고 있습니다. 하지만 사용자에게 보여줄 때는 부서 이름을 보여주어야 합니다. 그리드 컴포넌트에는 데이터 형식에 따라 displaytype 속성을 선택할 수 있습니다.

### 5.1.1 데이터셋(Dataset)

먼저 데이터셋을 새로 만들고 속성과 컬럼값을 아래와 같이 입력합니다.

속성	값	설명
id	dsDepartment	데이터셋 id

No.	id	type	size	설명
1	code	STRING	32	부서 코드
2	value	STRING	32	부서 이름



아래 코드를 [Dataset Editor > Source] 탭에 입력할 수도 있습니다.

```
<ColumnInfo>
  <Column id="code" type="STRING" size="32"/>
  <Column id="value" type="STRING" size="32"/>
</ColumnInfo>
<Rows>
  <Row>
    <Col id="code">0</Col>
    <Col id="value">Boardroom</Col>
  </Row>
  <Row>
    <Col id="code">1</Col>
    <Col id="value">Accounting</Col>
  </Row>
  <Row>
    <Col id="code">2</Col>
```

```

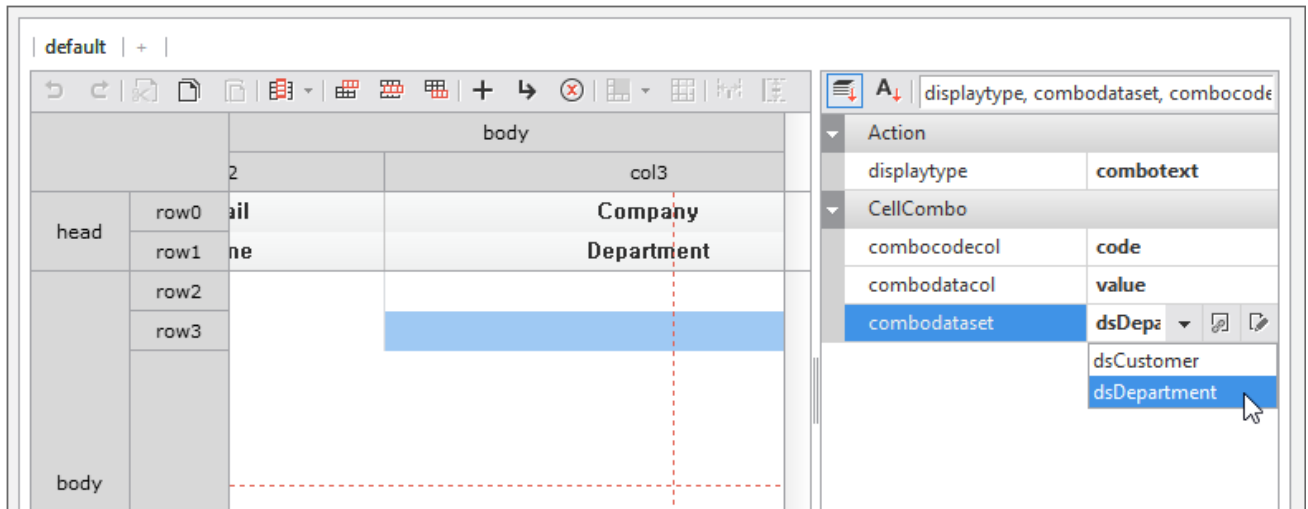
    <Col id="value">Personal</Col>
</Row>
<Row>
    <Col id="code">3</Col>
    <Col id="value">Human resources</Col>
</Row>
<Row>
    <Col id="code">4</Col>
    <Col id="value">Sales</Col>
</Row>
<Row>
    <Col id="code">5</Col>
    <Col id="value">Marketing</Col>
</Row>
<Row>
    <Col id="code">6</Col>
    <Col id="value">Engineering</Col>
</Row>
</Rows>

```

## 5.1.2 그리드(Grid)

그리드를 더블클릭해 그리드 콘텐츠 에디터를 실행합니다. Department 항목의 데이터 셀을 선택하고 속성창에서 다음 항목을 수정합니다. combodataset 항목은 직접 입력하거나 목록에서 선택할 수 있으며 combodataset 항목이 입력되면 combocodecol, combodatacol 항목도 직접 입력하거나 목록에서 선택할 수 있습니다.

속성	값	설명
displaytype	combotext	해당 셀의 데이터 표시 형식
combodataset	dsDepartment	displaytype이 combotext, combocontr 이 일 때 콤보 컴포넌트와 연결될 데이터셋
combocodecol	code	코드 컬럼
combodatacol	value	데이터 컬럼



### 5.1.3 Generate

ADL이 아닌 품을 수정한 경우에는 수정된 품에 해당하는 파일만 다시 만들면 됩니다. 메뉴에서 Generate Application이 아닌 Generate File을 선택합니다.

[Menu] Generate > File



특정 파일을 수정하고 저장하는 경우 해당 파일을 자동으로 다시 생성합니다. 위의 설명처럼 직접 메뉴를 선택해서 다시 생성하지 않아도 수정된 내용을 확인할 수 있습니다.

Quick View로 다시 실행하면 Department 항목이 코드 대신 부서 이름으로 표기되는 것을 확인할 수 있습니다.

ID	Name	Email	Company
		Phone	Department
TC-001	Tzuyu	ceo@twice.com	TWICE
		6987-6543	0

ID	Name	Email	Company
		Phone	Department
TC-001	Tzuyu	ceo@twice.com	TWICE
		6987-6543	Boardroom

## 5.2 트랜잭션

이번에는 transaction 메소드를 사용해 실제 서버에서 데이터를 가져옵니다. 넥사크로플랫폼 애플리케이션에서 사용하는 데이터 형식에 따라 만들어진 XML 파일을 가져오는 방식으로 테스트를 진행합니다.

단순 데이터 조회를 위한 테스트 데이터를 작성한 것이며 실제 애플리케이션 운영 환경에서는 데이터 저장소에 저장된 데이터를 가져오는 형식으로 개발해야 합니다.

### 5.2.1 sample.xml

예제에 사용하는 XML 파일은 아래와 같습니다. 4개의 레코드를 가지고 있으며 조회가 성공된 것을 전제로 작성했습니다.

sample.xml

```
<?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"/>
      <Column id="name" type="STRING" size="16"/>
      <Column id="email" type="STRING" size="32"/>
      <Column id="phone" type="STRING" size="16"/>
      <Column id="comp_name" type="STRING" size="32"/>
      <Column id="department" type="STRING" size="32"/>
      <Column id="comp_phone" type="STRING" size="16"/>
      <Column id="comp_addr" type="STRING" size="256"/>
    </ColumnInfo>
    <Rows>
      <Row>
        <Col id="id">TC-001</Col>
        <Col id="name">Tzuyu</Col>
        <Col id="email">ceo@twice.com</Col>
        <Col id="phone">6987-6543</Col>
        <Col id="comp_name">TWICE</Col>
```

```

        <Col id="department">0</Col>
        <Col id="comp_phone">6506-7000</Col>
        <Col id="comp_addr">Seoul</Col>
    </Row>
    <Row>
        <Col id="id">TC-002</Col>
        <Col id="name">Kei</Col>
        <Col id="email">kei@lovelyz.ca</Col>
        <Col id="phone">7357-3715</Col>
        <Col id="comp_name">Lovelyz</Col>
        <Col id="department">4</Col>
        <Col id="comp_phone">7357-7000</Col>
        <Col id="comp_addr">Lansing</Col>
    </Row>
    <Row>
        <Col id="id">TC-003</Col>
        <Col id="name">Choa</Col>
        <Col id="email">choa@aoa.com</Col>
        <Col id="phone">9025-0645</Col>
        <Col id="comp_name">AOA</Col>
        <Col id="department">2</Col>
        <Col id="comp_phone">9025-7000</Col>
        <Col id="comp_addr">Coral Springs</Col>
    </Row>
    <Row>
        <Col id="id">TC-004</Col>
        <Col id="name">Zhou Jieqiong</Col>
        <Col id="email">zhou.jieqiong@ioi.ca</Col>
        <Col id="phone">7026-3815</Col>
        <Col id="comp_name">I.O.I</Col>
        <Col id="department">3</Col>
        <Col id="comp_phone">7026-7000</Col>
        <Col id="comp_addr">Elmira</Col>
    </Row>
</Rows>
</Dataset>
</Root>

```



## 5.2.2 버튼 클릭 이벤트

앞에서 작성한 버튼 클릭 이벤트(divCommand\_btnSearch\_onclick)에 수작업으로 데이터를 추가하는 코드 대신에 sample.xml 파일에 접근해서 데이터를 가져오는 코드를 추가합니다.

수정된 코드는 아래와 같습니다. transaction 메소드에 필요한 파라미터를 지정하고 메소드를 호출합니다. transaction 메소드는 특정 데이터셋의 값을 갱신하기 위한 서비스(또는 파일)에 접근해 데이터를 내려받고 작업이 완료되면 콜백(CallBack) 함수를 호출하는 메소드입니다.

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

    var id = "search";
    var url = "http://localhost:8080/CustomerList/sample.xml";
    var reqDs = "";
    var respDs = "dsCustomers=customers";
    var args = "";
    var callback = "received";

    this.transaction(id, url, reqDs, respDs, args, callback);
}
```

예제에서 transaction 메소드에 사용한 파라미터는 아래와 같습니다.

파라미터	타입	설명
id	String	트랜잭션 구분하는 ID
url	String	트랜잭션을 요청하는 서비스 또는 파일의 URL
reqDs	String	애플리케이션에 사용된 데이터셋 데이터가 변경되었을 때 변경된 데이터셋을 지정하는 값입니다. (서비스 또는 파일에 지정된 데이터셋) = (애플리케이션에서 수정된 데이터셋) 형식으로 전달하며 1개 이상의 값을 빈칸으로 구분해 지정할 수 있습니다.
respDs	String	트랜잭션 처리 결과를 받을 때 지정하는 값입니다. (애플리케이션에서 수정된 데이터셋) = (서비스 또는 파일에 지정된 데이터셋) 형식으로 전달하며 1개 이상의 값을 빈칸으로 구분해 지정할 수 있습니다.
args	String	트랜잭션 요청 시 전송되는 파라미터를 지정하는 값입니다. parameter_name = value 형식으로 전달하며 1개 이상의 값을 빈칸으로 구분해 지정할 수 있습니다.
callback	String	트랜잭션의 결과를 처리할 콜백 함수를 지정하는 값입니다.



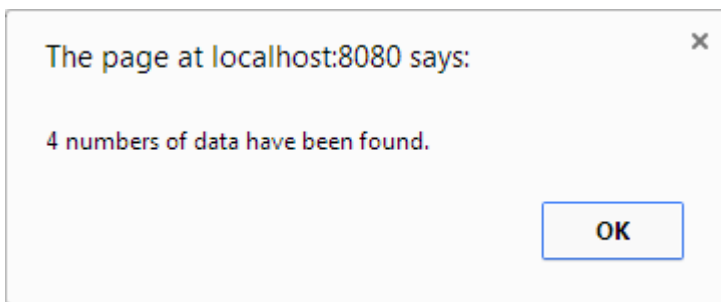
transaction 메소드는 비동기통신, 바이너리통신, 압축통신을 지정하는 3개의 파라미터가 더 있으며 해당 파라미터는 생략하면 기본값이 적용됩니다.

### 5.2.3 콜백 함수

transaction 메소드로 서비스 또는 파일에 접근한 이후에 지정된 콜백 함수를 실행합니다. 콜백함수는 아래와 같이 작성합니다.

```
this.received = function(id, code, message)
{
    if (code == 0) {
        this.alert(this.dsCustomers.rowcount + " numbers of data have been found.");
        trace(this.dsCustomers.rowcount + " numbers of data have been found.");
    } else {
        this.alert("Error["+code+"]: "+message);
        trace("Error["+code+"]: "+message);
    }
}
```

예제에서 사용된 XML 파일은 항상 성공 메시지만 보내기 때문에 아래와 같은 결과를 보여주며 그리드에 데이터를 표시합니다.



트랜잭션 url이 넥사크로플랫폼 애플리케이션과 같은 서버에 있지 않다면 Access-Control-Allow-Origin 오류가 발생할 수 있습니다.



오류가 발생하는 경우 code 값은 -1을 반환하고 message 값은 오류의 상태를 반환합니다. 상황에 따라 code값만 반환하고 message 값이 반환되지 않는 경우도 있는데, 서버의 예외적인 예외라 어떤 상황이라고 특정할 수 없습니다.

## 5.3 로그

넥사크로플랫폼 애플리케이션에서 발생하는 오류 메시지나 trace 메소드에서 처리되는 메시지는 런타임과 웹브라우저에 따라 조금씩 다릅니다. 폼 디자인이나 Generate 과정에서 발생하는 오류 메시지는 넥사크로 스튜디오 Output 창에서 확인할 수 있습니다.

하지만 애플리케이션 실행 중에 발생하는 오류나 메시지는 런타임일 경우에는 넥사크로 스튜디오 Output 창에서 확인할 수 있으며 웹브라우저에서 실행할 때는 각 웹브라우저 설정에 따라 확인 방법이 달라집니다.



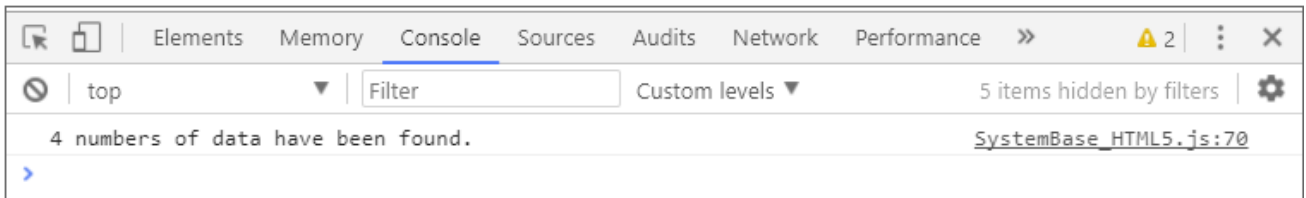
아래 내용은 특정 시점의 웹브라우저 버전에서 확인한 내용입니다. 사용하는 버전에 따라 일부 내용이 변경될 수 있습니다.

- 구글 크롬 61.x
- 파이어폭스 55.x
- 인터넷 익스플로러 11.x

### 5.3.1 구글 크롬

아래 메뉴에서 자바스크립트 콘솔을 보이게 할 수 있습니다.

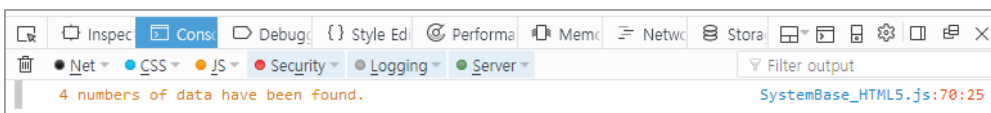
[Menu] More tools > Developer tools



### 5.3.2 파이어폭스

아래 메뉴에서 자바스크립트 콘솔을 보이게 할 수 있습니다. 구글 크롬보다 좀 더 많은 기능과 정보를 제공하고 있습니다.

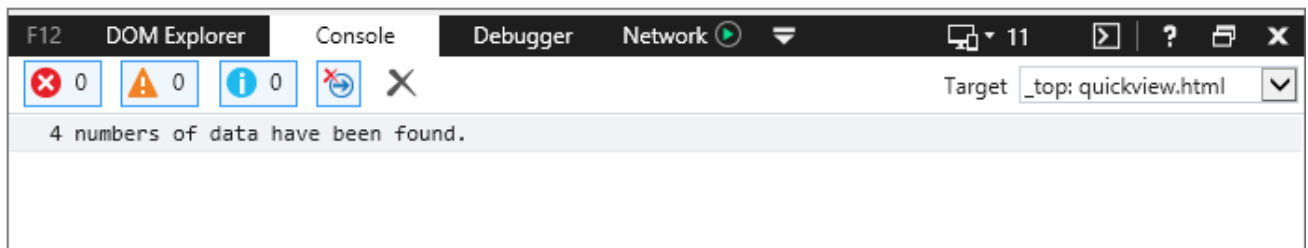
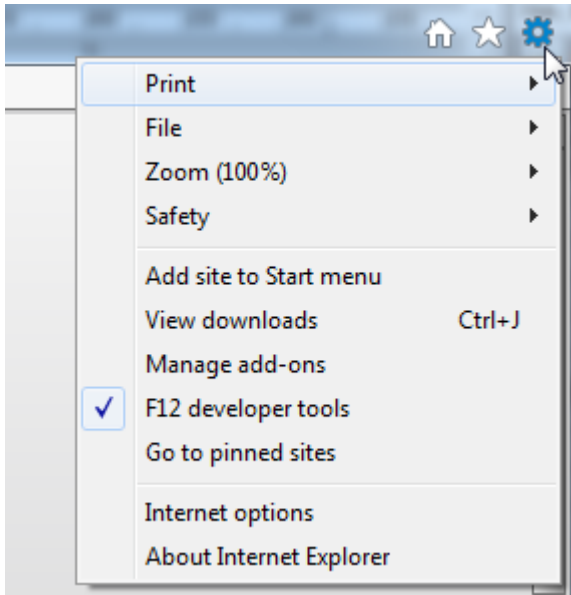
[Menu] Web Developer > Web Console



### 5.3.3 인터넷 익스플로러

아래 메뉴에서 자바스크립트 콘솔을 보이게 할 수 있습니다. 인터넷 익스플로러는 메뉴를 사용하지 않고 설정 아이콘을 클릭해도 바로 developer tools 항목으로 접근할 수 있습니다.

[Menu] Tools > developer tools > Console



마이크로소프트 Edge 브라우저 자바스크립트 콘솔 설정은 인터넷 익스플로러와 같습니다.

## 6.

# 화면 만들기 (X-API)



이번 장은 로컬 웹 서버나 경량 웹 서버로 실행할 수 없습니다. 톰캣이나 다른 WAS 서비스를 설정해두고 해당 서버를 사용해야 합니다.

톰캣 서버 설정에 관해서는 [아파치 톰캣\(Apache Tomcat\)](#) 항목을 참고해주세요.

**화면 만들기 (트랜잭션)**에서는 미리 작성해 저장한 XML 파일 데이터를 가져와 화면에 보여주는 기능을 설명했습니다. 하지만 실제 비즈니스 환경에서는 다양한 데이터베이스에서 데이터를 조회하고 입력한 데이터를 데이터베이스에 저장하는 등의 복잡한 업무 처리가 필요합니다.

넥사크로플랫폼 애플리케이션은 화면(Form)에서 입력받은 데이터를 서버로 전송하고 데이터를 받아오기 위해 `transaction()` 메소드를 사용합니다. `transaction()` 메소드는 입력 받은 변수와 데이터셋을 XML 형식으로 변형해 서버에 전달하고 콜백 함수를 통해 받아온 데이터를 처리합니다.

이런 과정에서 클라이언트가 전달한 변수와 데이터셋을 처리하고 데이터베이스에 있는 데이터를 처리하기 위해 서버 단에서 동작하는 서비스가 필요합니다. 서버에 구현한 서비스에서는 요청받은 데이터를 다양한 데이터베이스에서 가져와 이를 적절한 형태로 가공하고 클라이언트에 전달해줍니다. 요청에 문제가 생겼을 때 어떤 문제인지 확인할 수 있는 에러 코드와 메시지를 반환해줍니다.

서비스는 JSP, Servlet, ASP, PHP 등 서버 환경에 따라 다양한 프로그래밍 언어로 작성할 수 있습니다. 이번 장에서는 간단한 JSP 서비스를 만들고 어떻게 동작하는지 살펴봅니다.

## 6.1 서비스

화면에서 입력받은 데이터를 `transaction()` 메소드를 통해 서버에 전달해 처리하고 서버에 저장된 데이터를 다시 조회하는 서비스를 구현합니다.



간단한 테스트를 위해 데이터베이스에 연결하는 대신 파일 형태로 서버에 저장합니다.

아래 3가지 서비스를 설명합니다.

- initdata.jsp: 서버에 기본 자료를 생성하고 파일로 저장합니다.
- search.jsp: 저장된 파일에서 데이터를 읽어 Dataset을 만들고 클라이언트로 전송합니다.
- save\_list.jsp: 서버에 전송한 데이터로 기존 파일을 수정합니다.



확장자 JSP는 JavaServer Pages의 약자로 HTML 코드 안에 Java 코드를 추가해 동적으로 화면을 생성할 수 있는 언어를 의미합니다. 기본적인 화면 구조는 HTML 코드로 구성하고 데이터베이스에서 값을 받아 처리하거나 연산이 필요한 부분만 Java 코드를 사용합니다.

## 6.2 X-API

### 6.2.1 배포 파일

넥사크로플랫폼 X-API 라이브러리는 데이터 처리를 위한 서비스 구현 시 필요한 기능을 라이브러리 형태로 구현해 제공하고 있습니다. 제공하는 파일은 아래와 같습니다.

- nexacro17-xapi-1.0.jar (X-API 라이브러리 파일)
- commons-logging-1.1.1.jar (<http://commons.apache.org/proper/commons-logging/>)
- nexacro17\_server\_license.xml (라이선스 파일)
- docs > api (X-API 매뉴얼)



예제에서는 데이터 처리를 위해 넥사크로플랫폼 X-API를 사용하고 있습니다. X-API는 데이터 처리를 위해 필요한 기능을 구현한 라이브러리일뿐 X-API를 꼭 사용해야 하는 것은 아닙니다.



X-API 배포 파일은 기술지원사이트에서 내려받을 수 있습니다.

[<http://support.tobesoft.co.kr> > Downloads > nexacro platform 17 > Server > X-API]

개발용 라이선스 파일은 Downloads 메뉴 아래 [개발용라이선스] 링크에서 신청할 수 있습니다.

**TOBESOFT Technical Support**

Products QnA **Downloads** Education Documentation Community

**Downloads**

- nexacro platform17** >
- nexacro platform14
- XPLATFORM
- MiPlatform
- Nex-UP
- License정보
- 개발용라이선스

**nexacro platform17**

Developer **Server** X-Series

유지보수 미계약 고객(사)의 경우 패치 이후 발생하는 문제와 관련한 일체의 책임  
따라서 유지보수 미계약 고객(사)의 경우 관련 영업대표를 통해 유지보수를 체결해  
☞ nexacro platform17 시스템 요구사항

2017-09-21

**nexacro platform17 X-API**

Products	Files	
API (java)	nexacro17-xapi_20170921_2.zip	X-API(korean)

## 6.2.2 설치

톰캣을 설치하고 X-API 예제에서 사용할 컨텍스트 파일을 생성합니다. 톰캣 설치와 컨텍스트 파일을 생성하는 자세한 방법은 [아파치 톰캣\(Apache Tomcat\)](#) 항목을 참고하세요.

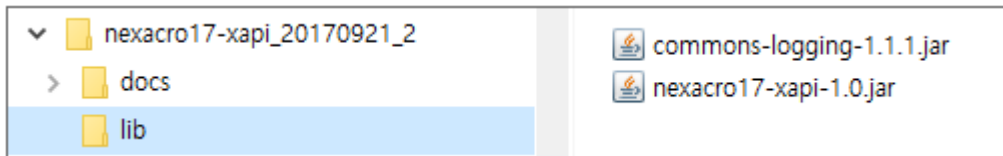
- 1 컨텍스트 파일을 생성합니다. 아래 경로에 CustomerList.xml 파일을 생성합니다.

C:\Program Files\Apache Software Foundation\Tomcat 8.5\conf\Catalina\localhost

CustomerList.xml

```
<Context path="/CustomerList" docBase="E:\88_TEST\02_BUILD\CustomerList"
  debug="0" privileged="true" reloadable="true">
  <Logger className="org.apache.catalina.logger.FileLogger"
    directory="logs" prefix="localhost_log." suffix=".txt"
    timestamp="true"/>
</Context>
```

- 2 내려받은 X-API 파일의 압축을 해제합니다. 압축을 해제하면 docs, lib 폴더 2개가 보이는데 그중에서 lib 폴더 안에 있는 2개의 jar 파일을 사용할 겁니다.



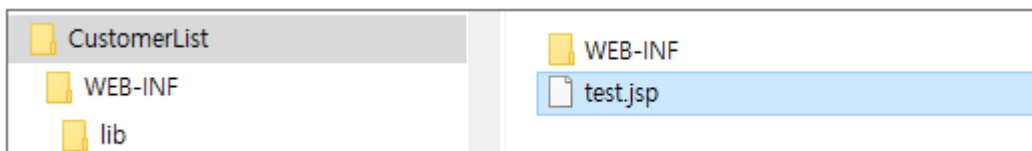
- 3 docBase로 지정한 폴더를 생성하고 해당 폴더 안에 WEB-INF 폴더를 생성합니다. 그리고 WEB-INF 폴더 안에 lib 폴더를 생성하고 jar 파일과 라이선스 파일을 복사합니다.

E:\88\_TEST\02\_BUILD\CustomerList\WEB-INF\lib



- 4 docBase로 지정한 폴더 아래에 테스트를 위한 jsp 파일을 생성합니다.

E:\88\_TEST\02\_BUILD\CustomerList



test.jsp

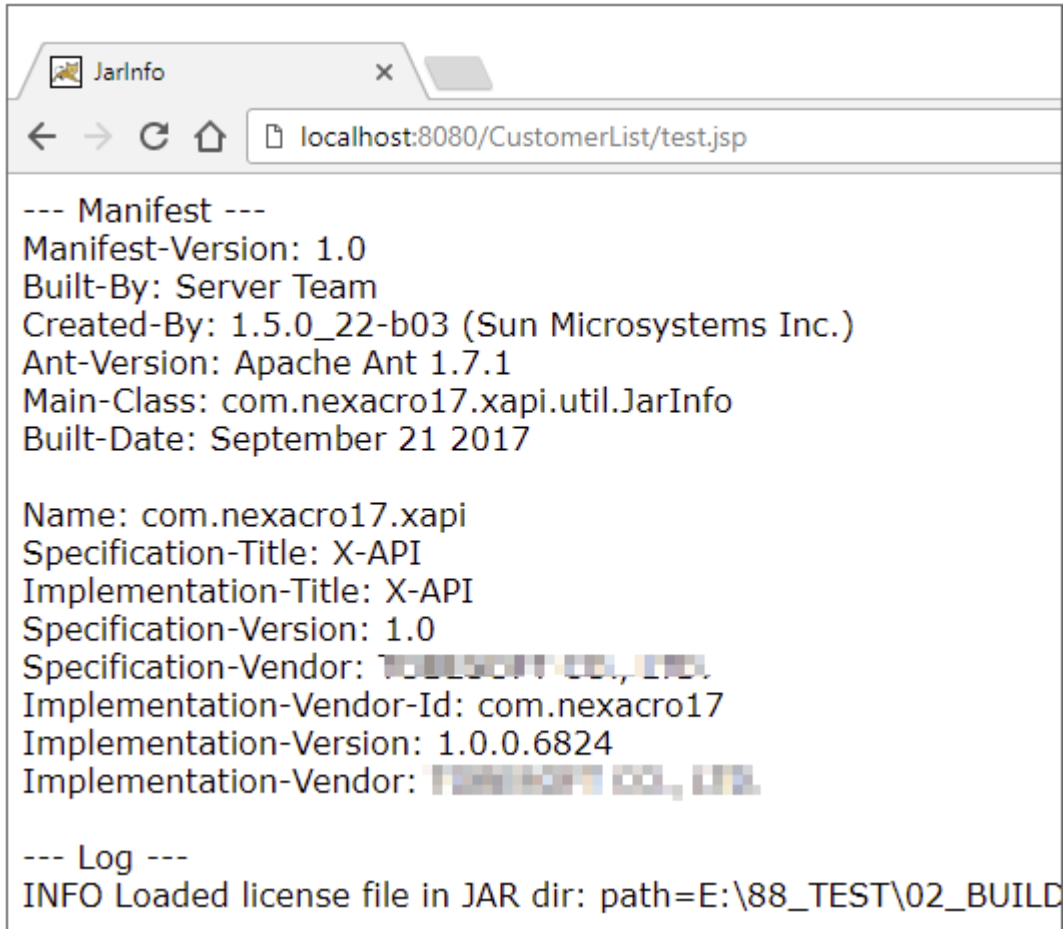
```
<%@ page contentType="text/html; charset=UTF-8" %>

<html>
  <head>
    <title>JarInfo</title>
    <style>
      * { font-family: Verdana }
    </style>
  </head>
  <body>
    <pre>
<%
  new com.nexacro17.xapi.util.JarInfo().info(out);
%>
    </pre>
  </body>
</html>
```



- 5 웹브라우저에서 아래 주소로 접근했을 때 설치 정보가 정상적으로 표시되는지 확인합니다. 설치정보가 표시된다면 정상적으로 X-API가 설치된 겁니다.

`http://localhost:8080/CustomerList/test.jsp`



WEB-INF는 Web Information의 약자입니다. 컨텍스트 루트 디렉토리 내에 있지만 사용자가 접근할 수 없으며 내부적으로 참조하기 위한 정보를 담고 있습니다.

lib 디렉토리는 jar 파일을 넣어 둘 수 있는 공간입니다. jar는 Java Archive의 약자로 여러개의 자바 클래스 파일과 리소스 등을 하나의 파일로 묶어서 라이브러리처럼 관리하는 파일입니다.



라이브러리를 복사할 위치는 사용하고 있는 WAS 설정에 따라 달라질 수 있습니다.



라이브러리 파일을 복사한 후에는 WAS 서비스를 재시작해주어야 합니다.  
WAS 설정에 따라 재시작이 필요없는 경우도 있습니다.



라이브러리 파일과 라이선스 파일은 같은 위치에 복사해야 인식할 수 있습니다.

### 6.2.3 전용객체

연결된 라이브러리에서 사용되는 전용객체는 다음과 같습니다.

- PlatformData : 데이터를 보관하는 기본객체
- PlatformRequest : JSP 요청 시에 XML Format Data를 읽고 객체화하는 Input 객체
- PlatformResponse : JSP 요청 시에 XML Format Data를 출력하는 output 객체
- DatasetList & Dataset : 데이터를 2차원 Table 형태 또는 Table Array 타입으로 보관
- VariableList & Variable : I/O 인자값으로 사용되는 단일 값을 보관

자세한 설명은 라이브러리에 포함된 X-API 매뉴얼을 참고해주세요.

## 6.3 화면에 Button 컴포넌트 추가

기존 화면에 데이터 초기화, 저장을 위한 Button 컴포넌트를 추가합니다.

Customer List Search

Search

1 Save

2 Initdata

컴포넌트	속성	값	설명
① Button	id	btnSaveList	
	text	Save	버튼에 표시할 문자열
② Button	id	btnInitdata	
	text	Initdata	버튼에 표시할 문자열

## 6.4 initdata.jsp

기본 자료를 생성하고 서버에 파일로 저장합니다.

### 6.4.1 pseudo code

```
// 1. 자바 라이브러리 지정 (nexacro platform X-API 포함)
// 2. MIME Type 정의
// 3. nexacro platform 기본객체(PlatformData) 생성
try {
    // 4. Data 처리
    // 5. ErrorCode, ErrorMsg 처리하기 (성공 메시지)
} catch (Error) {
    // 5. ErrorCode, ErrorMsg 처리하기 (실패 메시지)
}
// 6. 결과 Data Client에게 보내기
```

### 6.4.2 코드 구현

#### 자바 라이브러리 지정

JSP 서비스를 작성하기 위해 기본 자바 라이브러리를 지정합니다. 코드는 아래와 같습니다.

```
<!-- 1. Designating a Java library -->
<%@ page import="java.io.*" %>
<%@ page import="com.nexacro17.xapi.data.*" %>
<%@ page import="com.nexacro17.xapi.tx.*" %>
```

#### MIME 타입 정의

XML 생성을 위한 MIME(Multipurpose Internet Mail Extensions) 타입을 정의합니다.

```
<!-- 2. Defining a MIME type -->
<%@ page contentType="text/xml; charset=UTF-8" %>
```

## 기본객체(PlatformData) 생성하기

데이터를 처리하기 위한 기본객체를 선언합니다. 기본 객체인 PlatformData는 넥사크로플랫폼 애플리케이션에서 사용하는 모든 데이터(Dataset, 변수)를 한꺼번에 담을 수 있는 객체입니다.

PlatformData를 선언하는 코드는 다음과 같습니다.

```
/** 3. Creating a basic object of Nexacro Platform */
PlatformData pdata = new PlatformData();
```

## Dataset을 생성하여 File로 저장하기

Dataset을 생성하여 column 정보를 입력하고 2개의 row를 생성한 후 row마다 column을 입력합니다. 생성한 DataSet은 쉽게 다룰 수 있도록 PlatformData에 등록합니다.

등록된 PlatformData 객체를 사용하여 "./saveFile.bin"이라는 이름의 파일로 저장합니다. Dataset을 생성하고 파일로 저장하는 코드는 다음과 같습니다.

```
/** 4. Processing data: saving data as a file */
/** 4.1 Creating a dataset and inputting basic data to the dataset */
DataSet ds = new DataSet("customers");
ds.addColumn("id",DataTypes.STRING, 4);
ds.addColumn("name",DataTypes.STRING, 16);
...

int row = 0;
int i = 0;
String[] customers = new String[8];

customers[0] = "TC-001";
customers[1] = "Tzuyu";
...

for (i = 0; i < 2; i++)
{
    row = ds.newRow();
    ds.set(row,"id",customers[0]);
    ds.set(row,"name",customers[1]);
    ...
}

pdata.addDataSet(ds);
```

```

/** 4.2 Saving a dataset to a file */
String targetFilename = "./saveFile.bin";
OutputStream target = new FileOutputStream(targetFilename);
PlatformResponse res = new PlatformResponse(target,
    PlatformType.CONTENT_TYPE_BINARY);
res.setData(pdata);
res.sendData();
target.close();

```

## ErrorCode, ErrorMsg 처리하기

예외 상황이 발생했을 때 이를 처리하기 위한 부분입니다.

```

/** 5.1 Processing ErrorCode and ErrorMsg */
int nErrorCode = 0;
String strErrorMsg = "START";
try {
    /** 5.2 Setting ErrorCode and ErrorMsg for success */
    nErrorCode = 0;
    strErrorMsg = "SUCC";
} catch (Throwable th) {
    /** 5.3 Setting ErrorCode and ErrorMsg for failure */
    nErrorCode = -1;
    strErrorMsg = th.getMessage();
}

/** 5.4 Saving ErrorCode and ErrorMsg to send them to the client */
PlatformData senddata = new PlatformData();
VariableList varList = senddata.getVariableList();
varList.add("ErrorCode", nErrorCode);
varList.add("ErrorMsg", strErrorMsg);

```

## 결과값 Client에게 보내기

초기값이 정상적으로 파일에 저장되었는지를 사용자에게 전달하기 위하여 PlatformData 객체를 사용합니다. 이때, 앞에서 저장한 ErrorCode와 ErrorMsg가 전달됩니다.

VariableList는 PlatformData의 멤버이므로 수행 결과값은 PlatformData 객체에 들어있습니다. 이제 PlatformData의 데이터를 넥사크로플랫폼에서 처리할 수 있는 XML Format으로 추출하여 전송하는 부분을 구현해 보겠습니다.

데이터를 전송하는 기능을 쉽게 구현하기 위해 PlatformResponse 객체를 만들고 PlatformData 객체에서 데이터를 출력시키는 코드는 다음과 같습니다.

```
/** 6. Sending result data to the client */
HttpPlatformResponse res = new HttpPlatformResponse(response,
    PlatformType.CONTENT_TYPE_XML, "UTF-8");
res.setData(senddata);
res.sendData();
```

정상적으로 데이터가 처리되었다면 클라이언트에게 전달되는 XML 값은 아래와 같습니다.

```
<Root xmlns="http://www.nexacro.com/platform/dataset" ver="5000">
  <Parameters>
    <Parameter id="ErrorCode" type="int">0</Parameter>
    <Parameter id="ErrorMsg" type="string">SUCC</Parameter>
  </Parameters>
</Root>
```

### 6.4.3 전체 코드

initdata.jsp

```
<!-- 1.Designating a Java library -->
<%@ page import="java.io.*" %>
<%@ page import="com.nexacro17.xapi.data.*" %>
<%@ page import="com.nexacro17.xapi.tx.*" %>

<!-- 2. Defining a MIME type -->
<%@ page contentType="text/xml; charset=UTF-8" %>

<%
/** 3. Creating a basic object of Nexacro Platform */
PlatformData pdata = new PlatformData();

/** 5-1. Processing ErrorCode and ErrorMsg */
int nErrorCode = 0;
String strErrorMsg = "START";

try {
```

```

/** 4. Processing data: saving data as a file */
/** 4.1 Creating a dataset and inputting basic data to the dataset */
DataSet ds = new DataSet("customers");
ds.addColumn("id",DataTypes.STRING, 4);
ds.addColumn("name",DataTypes.STRING, 16);
ds.addColumn("email", DataTypes.STRING, 32);
ds.addColumn("phone", DataTypes.STRING, 16);
ds.addColumn("comp_name", DataTypes.STRING, 32);
ds.addColumn("department", DataTypes.STRING, 32);
ds.addColumn("comp_phone", DataTypes.STRING, 16);
ds.addColumn("comp_addr", DataTypes.STRING, 256);

int row = 0;
int i = 0;
String[] customers = new String[8];

customers[0] = "TC-001";
customers[1] = "Tzuyu";
customers[2] = "ceo@twice.com";
customers[3] = "6987-6543";
customers[4] = "TWICE";
customers[5] = "0";
customers[6] = "6506-7000";
customers[7] = "Seoul";

for (i = 0; i < 1; i++)
{
    row = ds.newRow();
    ds.set(row,"id",customers[0]);
    ds.set(row,"name",customers[1]);
    ds.set(row,"email",customers[2]);
    ds.set(row,"phone",customers[3]);
    ds.set(row,"comp_name",customers[4]);
    ds.set(row,"department",customers[5]);
    ds.set(row,"comp_phone",customers[6]);
    ds.set(row,"comp_addr",customers[7]);
}

pdata.addDataSet(ds);

/** 4.2 Saving a dataset to a file */

```

```

String targetFilename = "./saveFile.bin";
OutputStream target = new FileOutputStream(targetFilename);
PlatformResponse res = new PlatformResponse(target,
    PlatformType.CONTENT_TYPE_BINARY);
res.setData(pdata);
res.sendData();
target.close();
System.out.println("after file write..");

/** 5.2 Setting ErrorCode and ErrorMsg for success**/
nErrorCode = 0;
strErrorMsg = "SUCC";

} catch (Throwable th) {
    /** 5.3 Setting ErrorCode and ErrorMsg for failure **/
    nErrorCode = -1;
    strErrorMsg = th.getMessage();
}

/** 5.4 Saving the ErrorCode and ErrorMsg to send them to the client **/
PlatformData senddata = new PlatformData();
VariableList varList = senddata.getVariableList();
varList.add("ErrorCode", nErrorCode);
varList.add("ErrorMsg", strErrorMsg);

/** 6. Sending result data to the client **/
HttpPlatformResponse res = new HttpPlatformResponse(response,
    PlatformType.CONTENT_TYPE_XML, "UTF-8");
res.setData(senddata);
res.sendData();
%>

```

## 6.4.4 데이터 초기화 이벤트

[화면 만들기 \(트랜잭션\)](#)에서 만든 화면에 버튼 컴포넌트를 추가하고 아래와 같이 onclick 이벤트를 추가합니다. 화면 내 버튼을 클릭하면 initdata.jsp 서비스가 호출되면서 서버에 데이터셋이 담긴 파일을 생성합니다.



```

this.divCommand_btnInitdata_onclick = function(obj:nexacro.Button, e:nexacro.ClickEventInfo)
{
    var id = "initdata";
    var url = "SvcList::initdata.jsp";
    var reqDs = "";
    var respDs = "";
    var args = "";
    var callback = "initdata_received";

    this.transaction(id, url, reqDs, respDs, args, callback);
}


this.initdata_received = function(id, code, message)
{
    if (code == 0) {
        this.alert(message);
        trace(message);
    } else {
        this.alert("Error["+code+"]: "+message);
        trace("Error["+code+"]: "+message);
    }
}

```

url 속성은 전체 URL을 지정할 수 있습니다. 매번 전체 URL을 지정하는 것이 번거롭고 자주 사용하는 도메인이라면 TypeDefinition에 서비스 URL을 추가하고 변경되는 파일명만 지정할 수 있습니다.

지정된 서비스는 아래와 같이 사용합니다.

```
var url = "[Service ID]::[file name]";
```


TypeDefinition - Services \*

Resource Service

No	PrefixID	Type	URL
1	theme	resource	./_resource/_theme/
2	initvalue	resource	./_resource/_initvalue/
3	imagerc	resource	./_resource/_images/
4	font	resource	./_resource/_font/

User Service

	Default			
	PrefixID	Type	URL	CacheLevel
—	Base	form	./Base/	session
—	SvcList	JSP	http://localhost:8080/CustomerList	session

## 6.4.5 저장된 파일

버튼을 클릭해 initdata.jsp 파일을 호출하면 "saveFile.bin"이라는 이름의 파일이 생성됩니다. 생성된 파일은 WAS (예제에서는 아파치 톰캣)을 설치한 폴더 아래에 생성됩니다.



특정 경로에 파일을 저장하고자 하는 경우에는 아래와 같이 initdata.jsp 파일을 수정해 적용할 수 있습니다.

```
//String targetFilename = "./saveFile.bin";
File targetFile = new File("c:/saveFile.bin");
OutputStream target = new FileOutputStream(targetFile);
```

## 6.5 search.jsp

저장된 파일에서 데이터를 읽어 Dataset을 만들고 클라이언트로 전송합니다.

### 6.5.1 pseudo code

```
// 1. 자바 라이브러리 지정 (nexacro platform X-API 포함)
// 2. MIME Type 정의
// 3. nexacro platform 기본객체(PlatformData) 생성
try {
    // 4. Data 처리
    // 5. ErrorCode, ErrorMsg 처리하기 (성공 메시지)
} catch (Error) {
    // 5. ErrorCode, ErrorMsg 처리하기 (실패 메시지)
}
// 6. 결과 Data Client에게 보내기
```

## 6.5.2 코드 구현

### 자바 라이브러리 지정

JSP 서비스를 작성하기 위해 기본 자바 라이브러리를 지정합니다. 코드는 아래와 같습니다.

```
<!-- 1. Designating a Java library -->
<%@ page import="java.io.*" %>
<%@ page import="com.nexacro17.xapi.data.*" %>
<%@ page import="com.nexacro17.xapi.tx.*" %>
```

### MIME 타입 정의

XML 생성을 위한 MIME(Multipurpose Internet Mail Extensions) 타입을 정의합니다.

```
<!-- 2. Defining a MIME type -->
<%@ page contentType="text/xml; charset=UTF-8" %>
```

### 기본객체(PlatformData) 생성하기

데이터를 처리하기 위한 기본객체를 선언합니다. 기본 객체인 PlatformData는 넥사크로플랫폼 애플리케이션에서 사용하는 모든 데이터(Dataset, 변수)를 한꺼번에 담을 수 있는 객체입니다.

PlatformData를 선언하는 코드는 다음과 같습니다.

```
/** 3. Creating a basic object of Nexacro Platform */
PlatformData pdata = new PlatformData();
```

### File의 내용을 읽어서 Dataset 생성하기

"./saveFile.bin" 파일의 내용을 읽어서 PlatformData 객체에 저장합니다. PlatformData 객체에 Dataset이 담겨 있습니다.

파일에서 내용을 읽어서 PlatformData에 담는 코드는 다음과 같습니다.

```
/** 4. Processing data: Loading data from the file: loading data from the file */
/** 4.1 Loading data from the file */
String sourceFilename = "./saveFile.bin";
InputStream source = new FileInputStream(sourceFilename);
```

```
PlatformRequest req = new PlatformRequest(source,
    PlatformType.CONTENT_TYPE_BINARY);
req.receiveData();
source.close();

/** 4.2 Copying the loaded data to the dataset */
pdata = req.getData();
```

## ErrorCode, ErrorMsg 처리하기

예외 상황이 발생했을 때 이를 처리하기 위한 부분입니다.

```
/** 5.1 Processing ErrorCode and ErrorMsg */
int nErrorCode = 0;
String strErrorMsg = "START";
try {
    /** 5.2 Setting ErrorCode and ErrorMsg for success*/
    nErrorCode = 0;
    strErrorMsg = "SUCC";
} catch (Throwable th) {
    /** 5.3 Setting ErrorCode and ErrorMsg for failure */
    nErrorCode = -1;
    strErrorMsg = th.getMessage();
}

/** 5.4 Saving ErrorCode and ErrorMsg to send them to the client */
PlatformData pdata = new PlatformData();
VariableList varList = pdata.getVariableList();
varList.add("ErrorCode", nErrorCode);
varList.add("ErrorMsg", strErrorMsg);
```

## 결과값 Client에게 보내기

초기값이 정상적으로 파일에 저장되었는지를 사용자에게 전달하기 위하여 PlatformData 객체를 사용합니다. 이때, 앞에서 저장한 ErrorCode와 ErrorMsg가 전달됩니다.

[initdata.jsp](#)에서 PlatformData 객체는 ErrorCode와 ErrorMsg만을 담고 있지만, [search.jsp](#)에서는 PlatformData에 Client가 사용하는 명함리스트 Dataset을 담고 있습니다.

VariableList는 PlatformData의 멤버이므로 수행 결과값은 PlatformData 객체에 들어있습니다. 이제 PlatformDat

a의 데이터를 넥사크로플랫폼에서 처리할 수 있는 XML Format으로 추출하여 전송하는 부분을 구현해 보겠습니다. 데이터를 전송하는 기능을 쉽게 구현하기 위해 PlatformResponse 객체를 만들고 PlatformData 객체에서 데이터를 출력시키는 코드는 다음과 같습니다.

```
/** 6. Sending result data to the client */
HttpPlatformResponse res = new HttpPlatformResponse(response,
    PlatformType.CONTENT_TYPE_XML, "UTF-8");
res.setData(pdata);
res.sendData();
```

정상적으로 데이터가 처리되었다면 클라이언트에게 전달되는 XML 값은 아래와 같습니다.

```
<Root xmlns="http://www.nexacro.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"/>
      <Column id="name" type="string" size="16"/>
      <Column id="email" type="string" size="32"/>
      <Column id="phone" type="string" size="16"/>
      <Column id="comp_name" type="string" size="32"/>
      <Column id="department" type="string" size="32"/>
      <Column id="comp_phone" type="string" size="16"/>
      <Column id="comp_addr" type="string" size="256"/>
    </ColumnInfo>
    <Rows>
      <Row>
        <Col id="id">TC-001</Col>
        <Col id="name">Tzuyu</Col>
        <Col id="email">ceo@twice.com</Col>
        <Col id="phone">6987-6543</Col>
        <Col id="comp_name">TWICE</Col>
        <Col id="department">0</Col>
        <Col id="comp_phone">6506-7000</Col>
        <Col id="comp_addr">Seoul</Col>
      </Row>
    </Rows>
  </Dataset>
```

&lt;/Root&gt;

## 6.5.3 전체 코드

search.jsp

```

<!-- 1.Designating a Java library -->
<%@ page import="java.io.*" %>
<%@ page import="com.nexacro17.xapi.data.*" %>
<%@ page import="com.nexacro17.xapi.tx.*" %>

<!-- 2. Defining a MIME type -->
<%@ page contentType="text/xml; charset=UTF-8" %>
<%

/** 3. Creating a basic object of Nexacro Platform */
PlatformData pdata = new PlatformData();

/** 5.1 Processing ErrorCode and ErrorMsg */
int nErrorCode = 0;
String strErrorMsg = "START";

try {
    /** 4. Processing data : Loading data from the file: loading data from the file */
    /** 4.1 Loading data from the file */
    String sourceFilename = "./saveFile.bin";
    InputStream source = new FileInputStream(sourceFilename);

    PlatformRequest req = new PlatformRequest(source,
        PlatformType.CONTENT_TYPE_BINARY);
    req.receiveData();
    source.close();

    /** 4.2 Copying the loaded data to the dataset */
    pdata = req.getData();

    /** 5.2 Setting ErrorCode and ErrorMsg for success */
    nErrorCode = 0;
    strErrorMsg = "SUCC";

} catch (Throwable th) {

```

```

    /** 5.3 Setting ErrorCode and ErrorMsg for failure */
    nErrorCode = -1;
    strErrorMsg = th.getMessage();
}

/** 5.4 Saving ErrorCode and ErrorMsg to send them to the client */
VariableList varList = pdata.getVariableList();
varList.add("ErrorCode", nErrorCode);
varList.add("ErrorMsg", strErrorMsg);

/** 6. Sending result data to the client */
HttpPlatformResponse res = new HttpPlatformResponse(response,
    PlatformType.CONTENT_TYPE_XML, "UTF-8");
res.setData(pdata);
res.sendData();
%>

```

## 6.5.4 데이터 조회 이벤트

[화면 만들기 \(트랜잭션\)](#)에서 만든 화면에서 조회 버튼에 지정된 onclick 이벤트를 수정합니다. 화면 내 버튼을 클릭하면 search.jsp 서비스가 호출되면서 서버에 저장된 파일을 읽어 데이터셋을 반환해줍니다.

서버에서 보낸 데이터셋 정보는 화면 내 작성된 dsCustomers 데이터셋에 담겨 해당 데이터를 그리드에 표시합니다.

```

this.divCommand_btnSearch_onclick = function(obj:nexacro.Button, e:nexacro.ClickEventInfo)
{
    var id = "search";
    var url = "SvcList::search.jsp";
    var reqDs = "";
    var respDs = "dsCustomers=customers";
    var args = "";
    var callback = "search_received";

    this.transaction(id, url, reqDs, respDs, args, callback);
}

this.search_received = function(id, code, message)
{
    if (code == 0) {

```

```

        var rowcount = this.dsCustomers.rowcount;
        this.alert(rowcount + " numbers of data have been found.");
        trace(rowcount + " numbers of data have been found.");
    } else {
        this.alert("Error["+code+"]: "+message);
        trace("Error["+code+"]: "+message);
    }
}

```

## 6.5.5 데이터베이스 연결

예제에서는 파일로 저장된 데이터셋을 가져와 그대로 클라이언트에게 전달합니다. 파일이 아닌 데이터베이스에 연결한 경우에는 아래와 같이 적용할 수 있습니다.



아래 예제에서 데이터베이스 설정은 예시를 위한 코드입니다. 실제 코드에서는 사용하는 환경에 맞게 수정해서 사용해야 합니다.

search.jsp (JDBC Connect)

```

<%@ page import = "java.util.*" %>
<%@ page import = "java.sql.*" %>
<%@ page import="com.nexacro17.xapi.data.*" %>
<%@ page import = "java.io.*" %>

<%@ page contentType="text/xml; charset=utf-8" %>

<%
/***** Service API initialization *****/
PlatformData pdata = new PlatformData();

int nErrorCode = 0;
String strErrorMsg = "START";

/***** JDBC Connection *****/
Connection conn = null;
Statement stmt = null;
ResultSet rs = null;
Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
conn = DriverManager.getConnection("jdbc:sqlserver://localhost:1433;;DatabaseName=Sample",

```



```

    "guest","guest");
stmt = conn.createStatement();

try {
    /***** SQL query *****/
    String SQL = "select * from sample_customers_list";
    rs = stmt.executeQuery(SQL);

    /***** Dataset Create *****/
    DataSet ds = new DataSet("customers");
    ds.addColumn("id",DataTypes.STRING, 4);
    ds.addColumn("name",DataTypes.STRING, 16);
    ds.addColumn("email", DataTypes.STRING, 32);
    ds.addColumn("phone", DataTypes.STRING, 16);
    ds.addColumn("comp_name", DataTypes.STRING, 32);
    ds.addColumn("department", DataTypes.STRING, 32);
    ds.addColumn("comp_phone", DataTypes.STRING, 16);
    ds.addColumn("comp_addr", DataTypes.STRING, 256);
    int row = 0;
    while(rs.next())
    {
        row = ds.newRow();
        ds.set(row, "id", rs.getString("id"));
        ds.set(row, "name", rs.getString("name"));
        ds.set(row, "email", rs.getString("email"));
        ds.set(row, "phone", rs.getString("phone"));
        ds.set(row, "comp_name", rs.getString("comp_name"));
        ds.set(row, "department", rs.getString("department"));
        ds.set(row, "comp_phone", rs.getString("comp_phone"));
        ds.set(row, "comp_addr", rs.getString("comp_addr"));
    }

    /***** Adding Dataset to PlatformData *****/
    pdata.addDataSet(ds);

    nErrorCode = 0;
    strErrorMsg = "SUCC";
}
catch(SQLException e) {
    nErrorCode = -1;
    strErrorMsg = e.getMessage();
}

```

```

}

/***** JDBC Close *****/
if ( stmt != null ) try { stmt.close(); } catch (Exception e) {}
if ( conn != null ) try { conn.close(); } catch (Exception e) {}

PlatformData senddata = new PlatformData();
VariableList varList = senddata.getVariableList();
varList.add("ErrorCode", nErrorCode);
varList.add("ErrorMsg", strErrorMsg);

/***** XML data Create *****/
HttpPlatformResponse res = new HttpPlatformResponse(response,
    PlatformType.CONTENT_TYPE_XML, "UTF-8");
res.setData(pdata);
res.sendData();
%>

```

## 6.6 save\_list.jsp

서버로 전송한 데이터를 파일로 저장합니다.

### 6.6.1 pseudo code

[initdata.jsp](#), [search.jsp](#)와 다르게 '클라이언트 요청받기'라는 단계가 추가되었습니다.

```

// 1. Designating a Java library (including nexacro platform X-API)
// 2. Defining a MIME type
// 3. Creating a basic object of Nexacro Platform (PlatformData)
try {
    // 4. Receiving a request from the client
    // 5. Processing data
    // 6. Processing ErrorCode and ErrorMsg (success message)
} catch (Error) {
    // 6. Processing ErrorCode and ErrorMsg (failure message)

```

```
}
// 7. Sending result data to the client
```

## 6.6.2 코드 구현

### 자바 라이브러리 지정

JSP 서비스를 작성하기 위해 기본 자바 라이브러리를 지정합니다. 코드는 아래와 같습니다.

```
<!-- 1. Designating a Java library -->
<%@ page import="java.io.*" %>
<%@ page import="com.nexacro17.xapi.data.*" %>
<%@ page import="com.nexacro17.xapi.tx.*" %>
```

### MIME 타입 정의

XML 생성을 위한 MIME(Multipurpose Internet Mail Extensions) 타입을 정의합니다.

```
<!-- 2. Defining a MIME type -->
<%@ page contentType="text/xml; charset=UTF-8" %>
```

### 기본객체(PlatformData) 생성하기

데이터를 처리하기 위한 기본객체를 선언합니다. 기본 객체인 PlatformData는 넥사크로플랫폼 애플리케이션에서 사용하는 모든 데이터(Dataset, 변수)를 한꺼번에 담을 수 있는 객체입니다.

PlatformData를 선언하는 코드는 다음과 같습니다.

```
/** 3. Creating a basic object of Nexacro Platform */
PlatformData pdata = new PlatformData();
```

### 클라이언트 요청받기

클라이언트가 매개변수로 보낸 Dataset을 받아서 처리합니다.

```
/** 4. Receiving a request from the client */
// create HttpPlatformRequest for receive data from client
HttpPlatformRequest req = new HttpPlatformRequest(request);
```

```
req.receiveData();
```

## 클라이언트가 보낸 데이터 추출 후 파일로 저장하기

클라이언트가 보낸 정보를 처리할 수 있도록 PlatformData로 변환한 후 Dataset을 추출합니다. 생성된 PlatformData는 "./saveFile.bin" 파일로 저장합니다.

```
/** 5. Processing data: Loading data from the file */
/** 5.1 Loading data from the http object */
pdata = req.getData();

/** Obtaining a dataset from the received data */
DataSet ds = pdata.getDataSet("dsCustomers");

/** Saving data as a file with init data */
String targetFilename = "./saveFile.bin";
OutputStream target = new FileOutputStream(targetFilename);
PlatformResponse res = new PlatformResponse(target,
    PlatformType.CONTENT_TYPE_BINARY);
res.setData(pdata);
res.sendData();
target.close();
```

## ErrorCode, ErrorMsg 처리하기

예외 상황이 발생했을 때 이를 처리하기 위한 부분입니다. 성공적으로 처리한 경우에는 정상적으로 저장되었다는 메시지를 반환합니다.

```
/** 6.1 Processing ErrorCode and ErrorMsg */
int nErrorCode = 0;
String strErrorMsg = "START";
try {
    /** 6.2 Setting ErrorCode and ErrorMsg for success */
    nErrorCode = 0;
    strErrorMsg = "person list saved complete : row count("+ds.getRowCount()+")";
} catch (Throwable th) {
    /** 6.3 Setting ErrorCode and ErrorMsg for failure */
    nErrorCode = -1;
    strErrorMsg = th.getMessage();
}
```

```

/** 6.4 Saving ErrorCode and ErrorMsg to send them to the client */
PlatformData senddata = new PlatformData();
VariableList varList = senddata.getVariableList();
varList.add("ErrorCode", nErrorCode);
varList.add("ErrorMsg", strErrorMsg);

```

## 결과값 Client에게 보내기

초기값이 정상적으로 파일에 저장되었는지를 사용자에게 전달하기 위하여 PlatformData 객체를 사용합니다. 이때, 앞에서 저장한 ErrorCode와 ErrorMsg가 전달됩니다.

VariableList는 PlatformData의 멤버이므로 수행 결과값은 PlatformData 객체에 들어있습니다. 이제 PlatformData의 데이터를 넥사크로플랫폼에서 처리할 수 있는 XML Format으로 추출하여 전송하는 부분을 구현해 보겠습니다. 데이터를 전송하는 기능을 쉽게 구현하기 위해 PlatformResponse 객체를 만들고 PlatformData 객체에서 데이터를 출력시키는 코드는 다음과 같습니다.

```

/** 7. Sending result data to the client */
HttpPlatformResponse res = new HttpPlatformResponse(response,
    PlatformType.CONTENT_TYPE_XML, "UTF-8");
res.setData(senddata);
res.sendData();

```

## 6.6.3 전체 코드

save\_list.jsp

```

<!-- 1. Designating a Java library -->
<%@ page import="java.io.*" %>
<%@ page import="com.nexacro17.xapi.data.*" %>
<%@ page import="com.nexacro17.xapi.tx.*" %>

<!-- 2. Defining a MIME type -->
<%@ page contentType="text/xml; charset=UTF-8" %>

<%
/** 3. Creating a basic object of Nexacro Platform */
PlatformData pdata = new PlatformData();

/** 6.1 Processing ErrorCode and ErrorMsg */

```

```

int nErrorCode = 0;
String strErrorMsg = "START";

try {
    /** 4. Receiving a request from the client */
    // create HttpPlatformRequest for receive data from client
    HttpPlatformRequest req = new HttpPlatformRequest(request);
    req.receiveData();
    /** 5. Processing data: Loading data from the file */
    /** 5.1 Loading data from the http object */
    pdata = req.getData();

    /** Obtaining a dataset from the received data */
    DataSet ds = pdata.getDataSet("dsCustomers");

    /** Saving data as a file with init data */
    String targetFilename = "./saveFile.bin";
    OutputStream target = new FileOutputStream(targetFilename);
    PlatformResponse res = new PlatformResponse(target,
        PlatformType.CONTENT_TYPE_BINARY);
    res.setData(pdata);
    res.sendData();
    target.close();

    /** 6.2 Setting ErrorCode and ErrorMsg for success */
    nErrorCode = 0;
    strErrorMsg = "person list saved complete : row count("+ds.getRowCount()+")";
} catch (Throwable th) {
    /** 6.3 Setting ErrorCode and ErrorMsg for failure */
    nErrorCode = -1;
    strErrorMsg = th.getMessage();
    System.out.println("ERROR:"+strErrorMsg);
}

/** 6.4 Saving ErrorCode and ErrorMsg to send them to the client */
PlatformData senddata = new PlatformData();
VariableList varList = senddata.getVariableList();
varList.add("ErrorCode", nErrorCode);
varList.add("ErrorMsg", strErrorMsg);

```

```

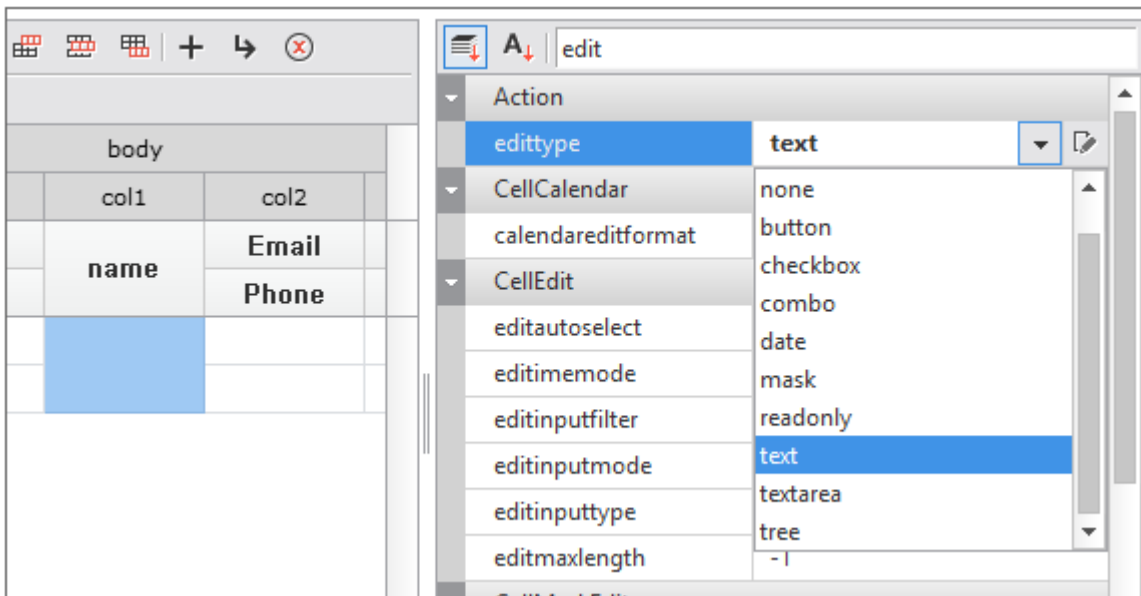
/** 7. Sending result data to the client */
HttpPlatformResponse res = new HttpPlatformResponse(response,
    PlatformType.CONTENT_TYPE_XML,"UTF-8");
res.setData(senddata);
res.sendData();
%>

```

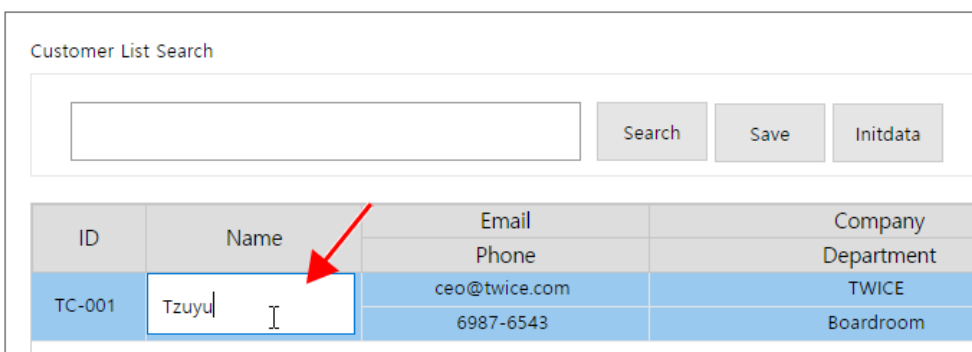
## 6.6.4 데이터 저장 이벤트

[화면 만들기 \(트랜잭션\)](#)에서 만든 화면에 버튼 컴포넌트를 추가하고 아래와 같이 onclick 이벤트를 추가합니다. 화면 내 버튼을 클릭하면 save\_list.jsp 서비스가 호출되면서 화면에서 수정한 데이터셋을 서버로 전송하고 다시 파일로 저장합니다.

데이터를 수정할 수 있는 기능을 간단하게 추가합니다. 화면 내 그리드를 더블클릭하고 Grid Contents Editor를 실행한 다음 Name 항목을 선택한 후 edittype 속성을 'text'로 수정합니다.



데이터 조회 후 해당 항목을 더블 클릭하면 데이터를 수정할 수 있습니다.



데이터 수정 후 저장(save) 버튼을 클릭해 데이터를 서버로 전송합니다.

화면을 새로 고침한 다음 조회(Search) 버튼을 클릭하면 수정한 데이터가 조회되는 것을 확인할 수 있습니다.

```
this.divCommand_btnSaveList_onclick = function(obj:nexacro.Button, e:nexacro.ClickEventInfo)
{
    var id = "save_list";
    var url = "SvcList::save_list.jsp";
    var reqDs = "customers=dsCustomers";
    var respDs = "";
    var args = "";
    var callback = "save_list_received";

    this.transaction(id, url, reqDs, respDs, args, callback);
}

this.save_list_received = function(id, code, message)
{
    if (code == 0) {
        this.alert(message);
        trace(message);
    } else {
        this.alert("Error["+code+"]: "+message);
        trace("Error["+code+"]: "+message);
    }
}
```