

# 넥사크로플랫폼 14 / 개발자 가이드

---

14.0.0.8



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 | 2014/07/07

주소 | (135-873) 서울시 강남구 봉은사로 617 인탑스빌딩 5층

전화 | 02-2140-7700

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

고객지원센터 | [www.nexacro.co.kr](http://www.nexacro.co.kr)

제품기술문의 | 1588-7895 (오전 10시부터 오후 5시까지)

# 차례

---

저작권 및 면책조항 .....	ii
차례 .....	iii
일러두기 .....	viii

## 파트 I. 개요 ..... 1

1. 넥사크로플랫폼 소개 .....	2
1.1 기업에서의 UI/UX .....	2
BUX: Business User eXperience .....	3
1.2 넥사크로플랫폼 14 .....	4
주요 특징 .....	4
적용 대상 시스템 .....	5
2. 넥사크로플랫폼 개요 .....	6
2.1 넥사크로플랫폼 통합 프레임워크 .....	6
2.2 넥사크로플랫폼 구성 요소 .....	8
프로그래밍 언어 .....	8
개발 환경 .....	10
실행 환경 .....	11
배포 환경 .....	12
2.3 시스템 요구사항 .....	13

## 파트 II. 기본 ..... 15

3. 넥사크로플랫폼 애플리케이션 구조 .....	16
3.1 넥사크로플랫폼 애플리케이션 파일 .....	16
ADL File .....	17
Type Definition File .....	20
Global Variable File .....	22
테마 / 스타일 .....	23

FDL 파일	23
3.2 주요 화면 요소	24
프레임을 이용한 화면배치	24
SDI (Single Document Interface)	25
MDI (Multi Document Interface)	25
Form의 구성	26
3.3 서버와의 Data연동	27
X-API	27
<b>4. XML 파일 구조</b>	<b>28</b>
4.1 ADL XML Format	28
4.2 FDL XML Format	29
4.3 Type Definition XML Format	30
4.4 Global Variable XML Format	30
<b>5. 넥사크로플랫폼 스크립트 언어</b>	<b>31</b>
5.1 유효범위(Scope)	31
this	31
Global	33
Expr	35
lookup	36
5.2 이벤트 핸들러	37
메소드	37
타입	38
5.3 Setter	39
set 메소드	39
동적인 속성	41
5.4 기타 변경 사항	42
nexacro 메소드	42
동작 방식 변경	43
오브젝트 명 생성 시 제약	43
<b>6. Frame Tree</b>	<b>44</b>
6.1 표기법	44
box	44
연결선	44
설명	45
용어정의	45
공통으로 사용하는 Syntax	45
6.2 Form	46
관계도	46

Script 예시 화면	47
component / invisible object / bind 의 Script 접근	48
container component에 Script 접근	49
form을 연결한 container component에 Script 접근	49
parent의 Script 사용	49
container component의 element사용	50
6.3 Application	51
관계도	51
Script 예시 화면	51
application에서 form의 script 접근	52
form에서 application의 script 접근	52
<b>7. 스타일(CSS)과 테마 관리</b>	<b>53</b>
7.1 스타일 / 테마 개요	53
스타일의 정의 및 적용	53
테마의 정의	56
스타일/테마의 등록 및 적용	57
스타일과 테마의 적용 대상	58
7.2 스타일의 적용	59
스타일의 생성	59
스타일의 편집	62
스타일의 적용	67
cssclass의 적용	70
7.3 테마의 적용	71
테마의 생성	71
테마의 편집	72
테마의 적용	74
Deploy Theme	76
<b>8. Position</b>	<b>78</b>
8.1 좌표계 설명	78
8.2 컴포넌트의 위치값 설정	80
8.3 폼 디자인	81
트래커	81
눈금자 / Dot Grid	82
컴포넌트 크기 정보	84
포지션 에디터	85
8.4 Position 관련 주의사항	86
<b>9. MLM(Multi Layout Manger)</b>	<b>88</b>
9.1 프로젝트 생성 및 수정	88

프로젝트 생성	88
스크린 정보의 수정	91
레이아웃 템플릿 등록	92
9.2 폼 생성	93
9.3 InitValue	96
9.4 실행	97
9.5 XML 추가/변경 사항	98
XPRJ	98
XADL	98
XFDL	99

## 파트 III. 확장 ..... 102

10. 프로토콜 어댑터	103
10.1 프로토콜 어댑터	103
모듈(Module) 등록	103
프로토콜(Protocol) 등록	105
서비스 등록	108
데이터 호출	109
10.2 PluginElement	110
프로토콜(Protocol) 클래스 코드	110
11. 확장 모듈	113
11.1 개발 환경	113
11.2 프로젝트 생성과 설정	113
프로젝트 생성	113
프로젝트 설정	115
11.3 확장 모듈 제작	115
Entry Funcs	115
Initialize	115
EntryPoint	115
Shutdown	116
11.4 넥사크로플랫폼 API	116
nexacro._addExtensionModule	116
nexacro._clearExtensionModule	117
12. 사용자 컴포넌트	118
12.1 사용자 컴포넌트 구성	118
모듈(Module) 등록	118
오브젝트 정보	120
컴포넌트 스크립트	127

12.2 사용자 컴포넌트 적용 .....	128
컴포넌트 등록 .....	128
컴포넌트 사용 .....	130
찾아보기 .....	131

# 일러두기

---

본 매뉴얼은 넥사크로플랫폼에 대한 기본 개요와 넥사크로플랫폼 애플리케이션을 만드는 개발 과정에 대해 다루고 있습니다.

## 대상 독자

### 대상 독자

넥사크로플랫폼 애플리케이션을 개발하거나 운영하는 개발자가 프로그래밍을 위해 참조할 수 있습니다.

### 넥사크로플랫폼 배포

넥사크로플랫폼 애플리케이션을 배포하거나 실행하는 단계에 대한 세부 내용은 관리자 가이드를 참조해 주세요.

### 넥사크로 스튜디오

넥사크로 스튜디오 설치 및 구성, 옵션과 관련된 세부 내용은 넥사크로 스튜디오 가이드를 참조해 주세요.

### 기타 개발 도구

투비소프트에서 제공하지 않는 웹서버 또는 기타 개발 도구에 대해서는 개략적인 내용만 다루고 있습니다. 각 제조사 매뉴얼을 별도로 참조해 주세요.

## 매뉴얼 구성

세부 항목에 대한 구성은 아래와 같습니다.

### 넥사크로플랫폼 소개

넥사크로플랫폼에 대해 개괄적으로 설명합니다.



## 넥사크로플랫폼 개요

넥사크로플랫폼의 기본 구조와 구성 요소에 대해 설명합니다.

## 기본

넥사크로플랫폼 애플리케이션 구조와 동작 방식, 스크립트 언어, 스타일, 테마 관리, Position, MLM에 대해 설명합니다.

## 확장

기능 확장에 필요한 프로토콜 어댑터, 확장 모듈, 사용자 컴포넌트에 대해 설명합니다.

# 매뉴얼 표기법

본 매뉴얼은 독자의 이해도를 높이고자, 특별한 의미가 있는 단어나 문장은 별도의 표기법으로 표현했습니다. 다음은 그 표기법에 대한 설명입니다.

노트, 팁, 주의는 다음과 같이 제공됩니다.



노트는 본문에 간단하게 추가할 짧은 설명이나 참조, 논평을 제공하기 위해 사용됩니다.



팁은 도움말 등의 팁을 제공하기 위해 사용됩니다.



주의는 독자 또는 사용자의 주의를 환기하는 문장을 제공하기 위해 사용됩니다.

파트 I.

---

개요

# 1.

## 넥사크로플랫폼 소개

업무에 최적화된 시스템, 원하는 정보를 쉽게 접근할 수 있는 시스템, 직관적으로 실행할 수 있는 시스템, 데이터 입력 작업과 같은 번거로운 작업에 시간을 덜 뺏기고 필요한 업무에 집중할 수 있는 시스템은 획기적인 업무 생산성의 향상을 가져올 수 있습니다. 이런 변화는 시스템의 UI/UX 환경을 어떻게 구축하느냐에 달려 있습니다.

UI/UX의 혁신은 시스템을 단지 보기 좋게 만드는 것이 아닌, 효율적으로 총소유비용(TCO, Total Cost of Ownership)을 낮추고 투자자본수익률(ROI, Return On Investment)을 높이면서 사용성도 함께 높여 경영자와 시스템 관리자, 사용자 모두의 만족도를 높여야 합니다.

넥사크로플랫폼은 기능, 성능, 디자인 요소들이 기업의 요구 수준에 맞게 조화를 이루도록 구성되어있으며 오직 기업만을 위해서 태어난 비즈니스 사용자 경험 솔루션이라고 할 수 있습니다.

### 1.1 기업에서의 UI/UX

X인터넷에서 RIA(Rich Internet Application), 그리고 REA(Rich Enterprise Application)에 이르기까지 기업과 개인의 더 풍부한 사용자 경험을 구현하기 위한 웹 애플리케이션의 개념은 계속해서 발전해 왔습니다.

C/S(Client Server) 환경은 처리 속도는 만족스러웠지만 설치, 배포, 유지관리의 어려움이 있었고 웹이 등장하면서 이런 문제는 해결되었지만 데이터 처리 속도와 제한된 UI의 한계를 가지게 되었습니다. 이를 넘어서고자 국내에서는 X인터넷이 기업용 시스템 개발에 널리 도입되면서 안정적인 시스템을 필요로 하는 기업 내 업무의 사용자 인터페이스를 데스크탑 수준의 인터페이스로 제공하는 솔루션으로 자리매김했습니다. 국외에서는 2002년 어도비와 합병된 매크로미디어에서 사용하던 RIA라는 용어가 확산되면서 전체적인 시장이 확대되었습니다. X인터넷은 보다 기업적인 용도를 강조하며 성능과 기능을 향상시키는 데 초점을 맞추었고 RIA는 일반 사용자를 상대로 하는 광고나 디자인, 애니메이션을 강조하며 발전해 왔습니다.

시간이 지나면서 기업용 애플리케이션 시장에서도 좀 더 풍부하고 향상된 사용자 인터페이스를 요구하기 시작했고 기업을 위한 RIA란 의미의 REA 솔루션이 기업용 시장에서 각광을 받기 시작합니다. 하지만 REA는 일반 사용자를 대상으로 하는 RIA와는 근본적인 접근이 달랐습니다. 철저하게 기업 환경의 분석과 복잡한 개발 요구 사항의 이해 그리고 기업에 대한 높은 이해도와 수많은 구축 경험을 기반으로 만들어진 독자적인 영역이라 볼 수 있습니다.

최근 국내외적으로 'UX(User eXperience)'라는 용어가 주목받으면서 기업의 비즈니스 환경에 최적화된 UI/UX 구현

을 통한 기업에 가치 있는 비즈니스 뷰(VIEW)에 대한 관심이 높아졌습니다. 투비소프트에서는 이런 새로운 기업 사용자의 요구를 수용하며 REA 개념을 넘어 확장된 개념으로 BUX(Business User eXperience : 비즈니스 사용자 경험)를 정의했습니다. BUX는 직관적인 업무 환경, 사용자 경험을 기반으로 한 UI 구현을 통해 업무 생산성을 높이고 신속한 의사 결정을 가능하게 함으로써 무엇보다 기업 고객의 비즈니스 가치를 실현하는데 무게 중심을 두고 있습니다.

## BUX: Business User eXperience

투비소프트가 제안하는 BUX는 아래 3가지의 통합을 통해 실현됩니다.

첫 번째는 플랫폼의 통합(Unified Platform)입니다.

다양한 운영체제, 브라우저 및 디바이스는 물론 각기 다른 화면 크기에 대해 하나의 소스로 구동되고 최적화된 결과를 만들어내는 OSMU(One Source Multi Use)를 지향합니다.

두 번째는 개발 환경의 통합(Unified Development Process)입니다.

같은 개발 환경 내에서 인하우스(In House) 개발이나 SAP와 같은 UI 고도화 패키지 개발 작업을 모두 수행할 수 있으며 개발에서 테스트, 배포까지 소프트웨어 개발의 전 과정을 통합할 수 있게 하는 개발도구인 '넥사크로 스튜디오'를 제공함으로써 개발 생산성을 향상시킵니다.

세 번째는 데이터 중심의 통합(Unified Data Frame)입니다.

정형, 비정형 데이터 및 OLTP(Online Transaction Processing)성 업무와 빅데이터의 시각화를 포함한 OLAP(Online Analysis Processing)성 업무를 모두 지원하는 것을 의미합니다. 이는 데이터 지향적인 기업용 시스템 UI/UX에 있어 업무 효율성과 생산성의 향상과 직결되는 것이어서 특히 중요한 의미가 있습니다.

기업 시스템의 UI/UX 개선은 투자 비용 대비 업무 생산성 향상과 만족도 제고 효과가 가장 높은 방법입니다. 기존 시스템을 거의 건드리지 않고도 큰 폭의 개선 효과를 창출할 수 있기 때문입니다. 최소 비용을 통한 최대의 혁신 효과, 그리고 모바일, 클라우드, 빅데이터 기술을 적극 수용하여 기업용 UI/UX를 통합 지원하겠다는 의지가 바로 BUX 플랫폼인 넥사크로플랫폼이 추구하는 방향입니다.

## 1.2 넥사크로플랫폼 14

넥사크로플랫폼 14는 투비소프트가 개발한 BUX 플랫폼으로 기업의 다양한 개발 요구사항을 수용하기 위해 자바스크립트 기반의 자체적인 통합 프레임워크로 개발됐습니다. 넥사크로플랫폼으로 개발된 애플리케이션은 별도의 추가 개발 없이 다양한 운영체제, 브라우저, 디바이스에서 같은 기능을 구현할 수 있습니다.

넥사크로플랫폼 14에서 제공하는 위지윅(WYSIWYG)기반의 개발 도구인 넥사크로 스튜디오는 RTE(Real Time Enterprise) 실현을 위한 개발생산성을 최대화하기 위해 HTML5, 런타임 버전에 상관 없이 개발할 수 있으며 개발자의 편의성을 최대화함과 동시에, 손쉽게 다양한 디자인 효과를 줄 수 있도록 지원합니다.

넥사크로플랫폼 14는 시스템 UI에 접근하는 상황에 따라 HTML5, 런타임 버전으로 구분되며 같은 개발 프로세스와 설정 파일을 사용합니다. HTML5 버전은 별도의 추가 설치 없이 배포 문제에서 매우 자유로운 것이 특징이며 런타임 버전은 가상 엔진 설치를 통해 데스크탑, 스마트폰, 태블릿 등 다양한 환경에서 상대적으로 높은 성능과 안정성을 확보할 수 있습니다. 런타임 버전은 운영체제에 따라 최적화된 가상 엔진을 제공합니다.

### 주요 특징

- 강력한 개발 도구

기존 4GL 개발 도구와 같이 개발에 필요한 다양하고 편리한 기능을 넥사크로플랫폼 전용 개발 도구인 넥사크로 스튜디오에서 제공합니다. 위지윅(WYSIWYG) 방식의 화면 개발과 함께 지능적인 편집기를 제공해 개발자에게 다양한 편의 기능을 지원합니다.

- 표준준수

넥사크로플랫폼에서 사용하는 스크립트 언어는 ECMA, CSS, XML, DOM 등 국제 표준에 따른 범용 기술을 적용함으로써 다른 시스템 또는 솔루션과 유연한 인터페이스를 지원합니다.

표준 기술을 사용해 새로운 언어를 배워야 한다는 부담을 줄일 수 있으며 넥사크로플랫폼 개발 환경을 빠르게 습득해 개발 및 유지보수를 쉽게 할 수 있습니다.

- 통합된 디바이스 이벤트 처리

PC에서의 마우스 이벤트 처리와 스마트 디바이스에서의 터치 이벤트 처리를 표준화해 하나의 코드로 다양한 디바이스를 지원하는 이벤트 처리를 구현할 수 있습니다.

- 빠른 데이터 처리

넥사크로플랫폼은 서버와 클라이언트간 데이터 처리 시 화면과 정보를 뺀 데이터만을 송/수신함으로써 서버의 부하를 줄이고 네트워크 효율을 높여 빠른 통신 속도를 구현할 수 있습니다.

비동기 통신 기술을 이용한 데이터 통신을 지원하며 업무에 따라 최적의 수행 환경을 만들 수 있습니다.

- 기업에 특화된 풍부한 컴포넌트

애플리케이션에서 바로 사용할 수 있는 다양한 기업용 컴포넌트를 제공해, 애플리케이션에 필요한 컴포넌트 개발 기간을 최소화함으로써 개발 기간을 단축하고, 변화하는 업무 환경에 빠르게 대응할 수 있습니다. 특히 가장 많이 활용되는 Grid는 기업에서 주로 쓰는 기능을 내장해 제공하고 있습니다.

넥사크로 스튜디오에서 별도의 코딩 없이 드래그앤드롭만으로 각 컴포넌트에 대한 데이터 바인딩하고 클릭만으로 새로운 이벤트를 생성할 수 있습니다.

- 시각적인 효과  
다양한 시각적 효과를 처리하거나 스타일이나 테마를 편집하는 작업을 넥사크로 스튜디오 내에서 손쉽게 구현할 수 있으며 개발자와 디자이너 모두에게 유연한 협업 프로세스를 지원합니다.
- 기업 전용 브라우저 (런타임 버전)  
Real Time Enterprise(RTE) 실현을 위해서 기업에서 사용하는 애플리케이션에서 기능과 성능은 중요한 요소입니다. 웹 브라우저의 성능이 지속적으로 향상되고 있지만, 기업이 요구하는 안정적인 성능에 대한 기대를 충족시켜주지 못하는 한계를 갖고 있습니다. 이러한 한계를 탈피하기 위해서 런타임 버전에서 제공하는 기업 전용 브라우저 기능을 활용할 수 있습니다.

## 적용 대상 시스템

- 기존 시스템의 UI 고도화를 통한 TCO 절감 및 ROI 향상
  - 메인프레임 환경 등 기존 시스템을 웹으로 전환
  - 사용이 복잡한 SAP UI의 개선/고도화
  - 기존 시스템의 노후화로 UI 구성이 복잡해짐에 따라 사용성이 떨어진 시스템을 개편
  - 콜센터 시스템과 같이 UI 개선/재구축 시 고객만족도 향상, 인당 처리율 향상을 통한 비용 절감
  - ERP, CRM, PLM, SCM 등에 대한 시스템 접근성을 향상
- 클라이언트/서버 시스템을 인터넷 기반 시스템으로 전환
  - 클라이언트/서버 시스템의 설치, 배포, 업데이트 이슈를 근본적으로 해결
  - 순수 웹에서 지원하지 못하였던 오프라인 구동 및 Local DBMS나 Local File과의 연동 업무
- 스마트폰과 태블릿 등 다양한 기기 지원이 필요한 시스템 구축
  - PC와 스마트폰, 태블릿 모두를 지원
  - BYOD(Bring Your Own Device) 환경 지원 시스템 구축
- 수치, 통계 데이터 중심의 대고객 웹 서비스나 응용프로그램 개발
- 대표이사, 임원진 또는 의사결정권자를 대상으로 직관적이고 유려한 EIS나 상황판 등의 시스템 구축
- HTML5를 수용한 차세대 시스템 구축

## 2.

---

# 넥사크로플랫폼 개요

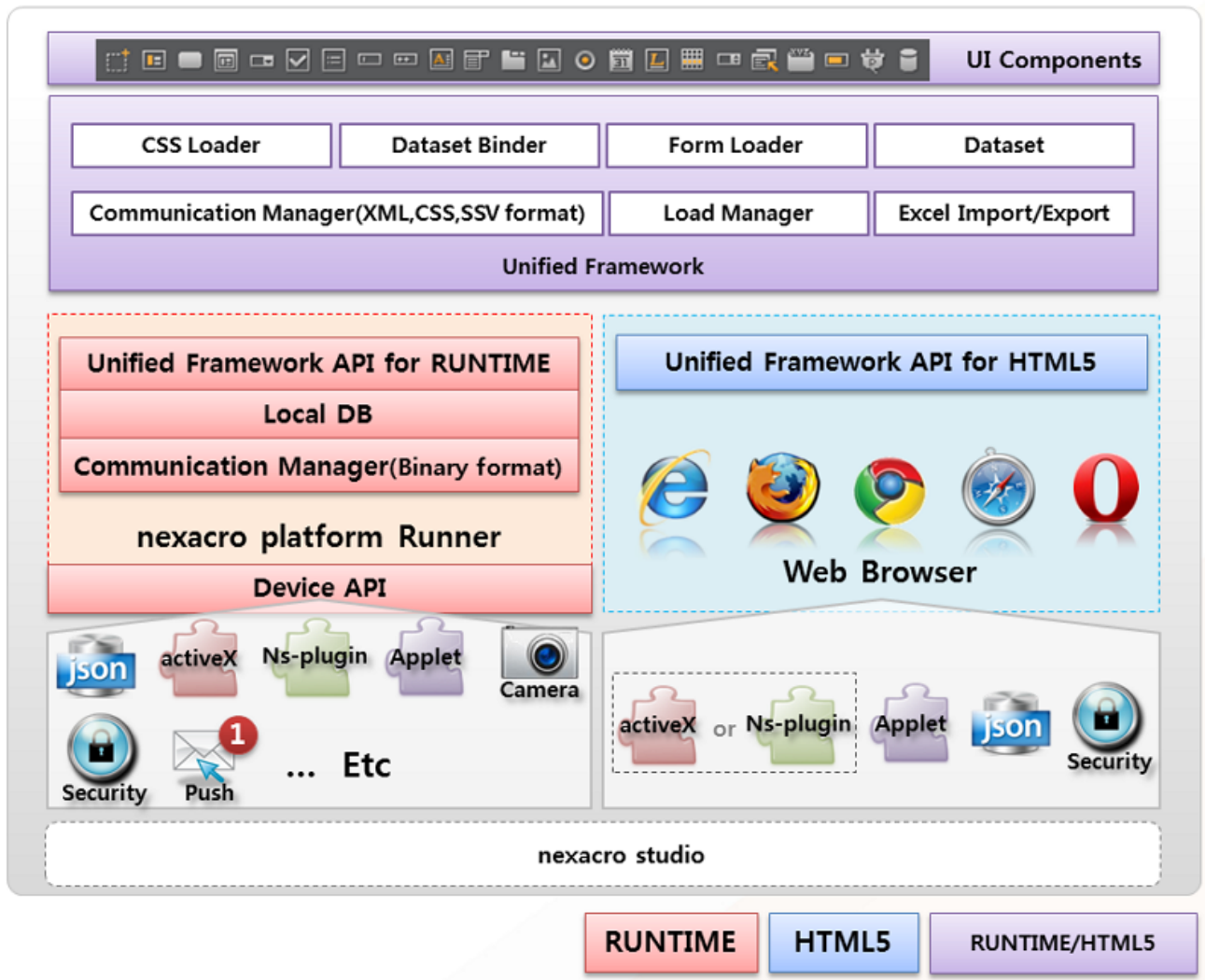
넥사크로플랫폼은 다양한 디바이스와 운영 환경을 지원하기 위해 통합 프레임워크(Unified Framework)와 자바스크립트 기반의 컴포넌트를 제공합니다. 하나의 코드로 어떤 환경에서든지 동작하는 애플리케이션을 만들 수 있으며 각 운영 환경에 따라 최적화된 실행 환경을 제공합니다.

## 2.1 넥사크로플랫폼 통합 프레임워크

넥사크로플랫폼 통합 프레임워크는 다양한 실행 환경에서 최적의 성능을 제공하고 새로운 운영체제가 등장하더라도 즉각적으로 대응할 수 있는 구조로 만들어졌습니다.

버전에 따라 패키징이나 배포 방식은 달라질 수 있지만, 내부적으로 동작하는 애플리케이션은 같기 때문에 어떤 환경에서도 같은 동작을 보장할 수 있습니다. 애플리케이션에서 사용하는 모든 컴포넌트는 자바스크립트 기반으로 만들어져 웹브라우저와 넥사크로플랫폼 러너(Runner)에서 같은 방식으로 동작합니다. 다만 웹브라우저는 사용자가 사용하는 버전에 따라 성능이 달라질 수 있습니다.

통합 프레임워크는 다른 요소들과 함께 넥사크로플랫폼 엔진 역할을 담당합니다. 주요 요소는 아래와 같습니다.



구성요소	설명
통합 프레임워크 (Unified Framework)	프로젝트와 애플리케이션을 구성하는 기본적인 프레임워크를 제공합니다. 데이터 통신(XML, CSV 등), 화면 간의 연계와 같은 작업을 처리하고 설정된 테마에 따라 전체 스타일을 처리하는 기능을 포함합니다.
UI 컴포넌트 (UI Components)	화면 상에 보이는 UI 컴포넌트와 Dataset과 같은 오브젝트를 제공합니다. 내부적으로 UI 컴포넌트는 기본 컴포넌트를 조합해 만들어집니다. - UI 컴포넌트: Button, Calendar, Checkbox, Grid 등 - 기본 컴포넌트: EditBase, FormBase, FrameBase, ScrollBar 등 - 오브젝트: Dataset
통합 프레임워크 API	애플리케이션은 통합 프레임워크로 구성되어 런타임과 HTML5 버전이 같은 방식으로 동작하며 개발자는 어떤 버전을 사용하던 같은 방식으로 개발합니다. 내부 요소인 엘리먼트(Element)는 각 환경에 최적화된 방식으로 개발됐습니다. 런타임 버전은 C를 기반으로 작성되며 HTML5 버전은 자바스크립트를 기반으로 작성됩니다. 하지만 개발자가 사용자 컴포넌트를 만들거나 프로젝트에 필요한 공통 작업을 할 때는 자바스크립트 기반으로 하나의 코드만을 사용합니다. 나머지는 통합 프레임워크 내부에서 사용 환경에 따라 필요한 요소를 적용해 처리합니다.
Device API	서드파티에서 개발한 모듈(Extension DLL)과 연계 시 사용하는 기능입니다.



구성요소	설명
	외부 장비 연동 등의 처리를 지원합니다.
Device API (Camera...)	모바일 기기 사용 시 카메라나 전화 걸기 등 기기 자체의 기능과 연계 시 사용하는 기능입니다. 기계에 따라 지원되는 기능이 제한될 수 있습니다. - 웹브라우저에 따라 일부 기능을 제공하지만 아직은 실험적인 수준이어서 HTML5 버전에서는 지원하지 않습니다.
Local Database	애플리케이션 내부에서 데이터를 저장하고 처리할 수 있는 기능을 지원합니다. SQLite를 지원하며 데스크탑 뿐 아니라 모바일 기기에서도 사용할 수 있습니다. - 웹브라우저에서도 Indexed Database( <a href="http://www.w3.org/TR/IndexedDB/">http://www.w3.org/TR/IndexedDB/</a> )를 지원하지만 아직은 실험적인 수준이어서 HTML5 버전에서는 지원하지 않습니다.
바이너리 데이터 통신	데이터 통신 시 XML, CSV, SSV 등의 형식은 런타임, HTML5 버전 모두 기본 지원하며 런타임 버전에서는 바이너리 통신을 추가적으로 지원합니다. - 웹브라우저에서도 바이너리 데이터 통신 처리가 가능하지만 아직은 실험적인 수준이어서 HTML5 버전에서는 지원하지 않습니다.

## 2.2 넥사크로플랫폼 구성 요소

넥사크로플랫폼은 4가지 요소로 구분됩니다. 프로그래밍 언어와 개발 환경은 제품 버전과 상관없이 같으며 실행 환경과 배포 환경은 버전에 따라 달라질 수 있습니다.

### 프로그래밍 언어

넥사크로플랫폼은 다른 프로그래밍 언어와 달리 사용자에게 보이는 화면을 정의하는 부분과 비즈니스 로직을 처리하는 스크립트로 구분됩니다. 또한, 화면에 원하는 디자인을 적용하기 위해 스타일과 테마를 적용할 수 있는 기능을 제공합니다.

화면을 배치하는 부분은 XML 기반으로 각 컴포넌트의 속성과 바인딩, 이벤트 등의 정보를 관리합니다. 다양한 실행 환경을 지원할 수 있도록 MLM(Multi Layout Manager) 기능을 지원하며 관련된 속성을 사용할 수 있습니다. 애플리케이션 실행에 필요한 환경 정보는 별도의 파일에서 관리합니다.

아래는 넥사크로플랫폼에서 애플리케이션 개발 시 기본적으로 생성되는 파일에 대한 설명입니다.

구분	파일명(확장자)	용도
nexacro platform Project	*.xprj	<ul style="list-style-type: none"> <li>프로젝트 정보</li> <li>TypeDefinition</li> <li>전역 변수</li> <li>ADL (애플리케이션)</li> </ul>
nexacro platform	*.xadi	<ul style="list-style-type: none"> <li>애플리케이션 실행 환경</li> </ul>

구분	파일명(확장자)	용도
Application Definition		<ul style="list-style-type: none"> <li>• TypeDefinition</li> <li>• 전역 변수</li> <li>• 테마 정보</li> <li>• 프레임 속성</li> <li>• 스크린 정보 (ScreenInfo)</li> </ul>
nexacro platform Form Definition	*.xfdl	<ul style="list-style-type: none"> <li>• 화면 레이아웃</li> <li>• 화면 폼 속성</li> <li>• 컴포넌트 속성</li> <li>• 추가 레이아웃</li> <li>• 스크립트</li> </ul>
TypeDefinition	default_typedef.xml	<ul style="list-style-type: none"> <li>• 모듈</li> <li>• 컴포넌트</li> <li>• 서비스</li> <li>• 업데이트</li> </ul>
GlobalVariable	globalvars.xml	<ul style="list-style-type: none"> <li>• 전역 변수</li> </ul>
Theme	*.xtheme	<ul style="list-style-type: none"> <li>• 스타일시트 (프레임, 폼, 컴포넌트 등)</li> <li>• 이미지</li> </ul>

생성된 애플리케이션은 빌드 과정을 거쳐 자바스크립트 기반의 코드로 변환됩니다. 실제 실행 환경에서는 변환된 자바스크립트 코드를 실행하게 됩니다.

## 개발 환경

넥사크로플랫폼은 위지윅(WYSIWYG) 기반의 개발 툴인 넥사크로 스튜디오를 제공합니다. 넥사크로 스튜디오 내에서 실행 환경과 상관없이 애플리케이션을 개발할 수 있으며 생성된 코드는 넥사크로플랫폼 프로그래밍 언어로 저장됩니다.

넥사크로 스튜디오는 마이크로소프트 윈도우 운영체제만을 지원하지만 개발된 애플리케이션은 어떤 운영체제나 어떤 디바이스든 상관없이 최적화된 사용 환경으로 배포할 수 있습니다.



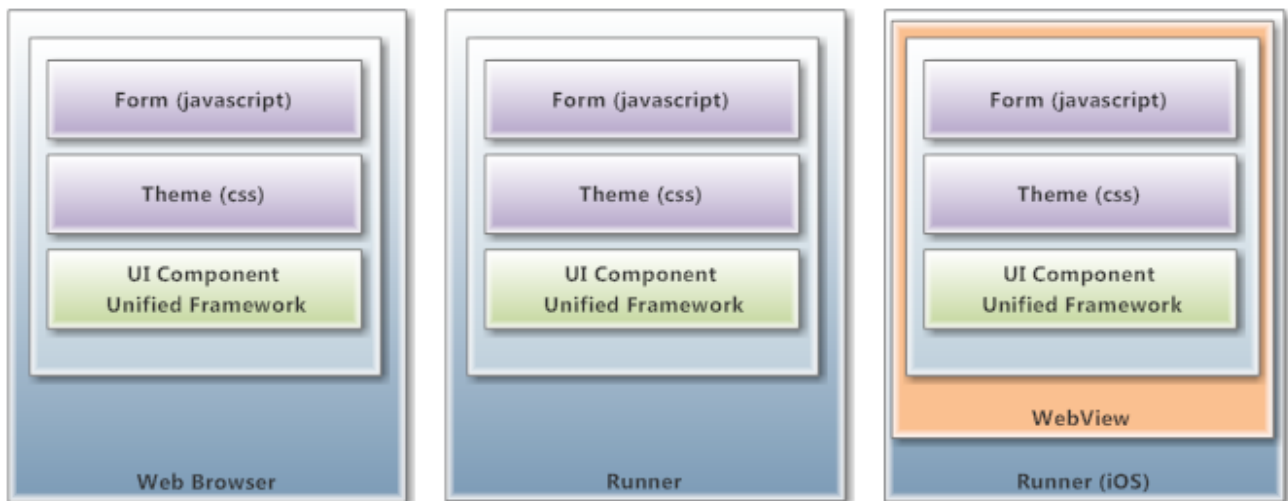
배포 환경에 따라 넥사크로 스튜디오 외 다른 개발 툴에서 추가적인 작업을 필요로 할 수 있으며 별도의 인증을 받아야 할 수 있습니다. 예를 들어 iOS 운영체제를 지원하는 앱을 개발하는 경우에는 배포 시 Xcode 환경이 필요하며 안드로이드 운영체제를 지원하는 앱을 개발하는 경우에는 배포 시 이클립스 환경이 필요합니다.

## 실행 환경

넥사크로플랫폼은 통합 프레임워크(Unified Framework)를 기본으로 애플리케이션이 실행되며 각 실행 환경에 따라 최적화된 구조를 제공합니다.

애플리케이션을 실행하는 주체에 따라 런타임과 HTML5 2가지 버전이 제공됩니다. 런타임 버전은 각 운영체제에 따라 최적화된 실행 환경을 제공합니다. 새로운 운영체제가 나오더라도 필요한 API를 추가해 지원할 수 있습니다. HTML5 버전은 사용자의 웹브라우저 기반으로 동작합니다.

넥사크로플랫폼이 실행되는 환경을 간략하게 표현하면 아래와 같습니다.



런타임 버전은 자체 개발된 렌더링 엔진을 포함한 넥사크로플랫폼 러너(Runner)를 제공합니다. 이 때문에 일반 웹브라우저보다 빠르고 안정적인 성능을 보장할 수 있습니다.



런타임 버전이라도 운영체제상 제약이 있는 경우가 있습니다. 예를 들어 iOS에서 실행되는 앱은 UIWebView라는 오브젝트만 사용해야 합니다. 같은 프레임워크를 사용하지만, 운영체제 제조사 정책 상 UIWebView 오브젝트로 한 번 더 감싸주는 형식을 취합니다.



HTML5 버전은 웹브라우저에 따라 애플리케이션 실행 성능이 달라질 수 있습니다. 특히 IE6는 마이크로소프트에서도 2014년 4월부터는 공식적으로 지원이 종료되며 최신 기술에 대한 성능 보장이 되지 않고 노출된 보안 위협을 더는 막을 방법이 제공되지 않습니다.

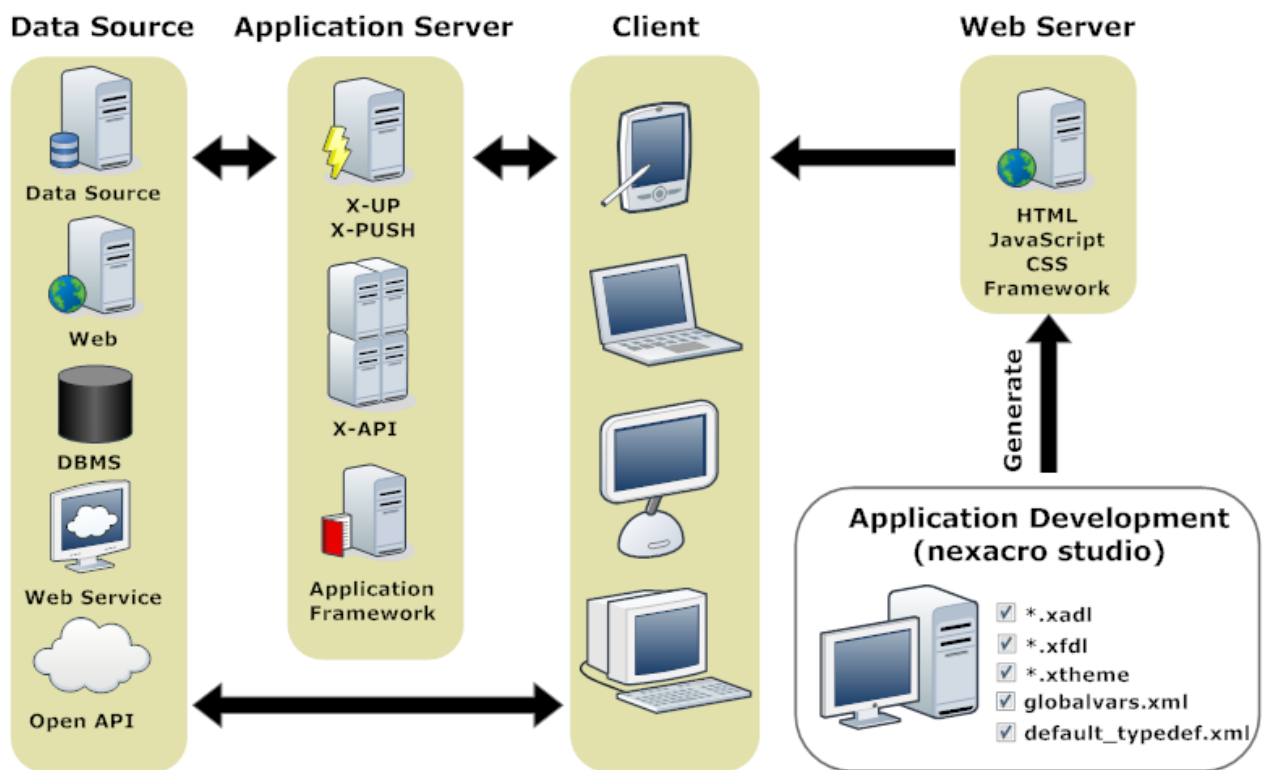
## 배포 환경

배포란 애플리케이션이 실행하는 데 필요한 자원을 클라이언트에 설치하는 일련의 작업을 의미합니다. 사용 환경에 따라 넥사크로플랫폼에서 개발된 애플리케이션과 필요한 모듈을 내려받아 클라이언트에 설치하게 됩니다.

넥사크로플랫폼 애플리케이션은 사용자가 사용하는 클라이언트에서 동작합니다. 하지만 데이터 처리와 같은 작업을 위해 애플리케이션 서버(Application Server)를 필요로 할 수 있습니다. 넥사크로플랫폼은 데이터 처리를 위한 X-API 모듈을 함께 제공하고 있습니다. 또한, 필요에 따라 데이터를 실시간으로 처리해야 한다면 X-PUSH와 같은 추가적인 기술을 사용할 수 있습니다.

기본 배포 작업은 HTTP 프로토콜을 사용합니다. 하지만 인터넷 접속을 지원하지 않는 환경에서는 애플리케이션 실행에 필요한 자원을 별도 매체로 배포해 사용할 수 있습니다.

각 배포 환경에 따라 추가적인 작업이 필요할 수 있습니다. 세부적인 배포와 관련된 내용은 관리자 가이드(Administrator Guide)에서 제공하는 버전별 상세 내용을 참고하세요.



X-PUSH는 런타임 버전만 지원합니다.  
런타임 버전에서는 설치 모듈이 실행환경에 따라 배포됩니다.



테마 파일은 기본 제공되는 테마를 사용할 경우에는 넥사크로 스튜디오에서 따로 만들지 않고 기본 테마를 변환하는 작업만 거칩니다.

## 2.3 시스템 요구사항

- Runtime

항목	사양	Windows	Android	iOS
CPU	최소	Intel® Celeron® Processor 430	ARM Cortex-A8 600 MHz	ARM Cortex-A8 600 MHz
	권장	Intel® Pentium®4 1GHz	ARM Cortex-A8 1GHz 이상	ARM Cortex-A8 1GHz 이상
디스플레이	최소	-	HVGA	HVGA
	권장	-	HVGA 이상	HVGA 이상
메모리	최소	512M	256M	256M
	권장	1G	512M	512M
HDD(ROM)	최소	50M	4M	4M
	권장	100M	10M	10M
플랫폼	최소	Windows2000 SP4	Android 2.3	iOS 4
	권장	Windows XP SP2	Android 2.3	iOS 6



넥사크로플랫폼 지원 단말기는 아래와 같습니다.

- **Android**

Fujitsu Mobile(F-12C), HTC(Nexus One), LG Electronics(Optimus 3D), Motorola(Atrix) NEC(MEDIAS N-01D), Pantech(Vega Racer), Samsung Electronics(Galaxy S2, Galaxy S3, Galaxy Tab 10.1, Galaxy Note2), SHARP(IS05), Sony Mobile(XPERIA arc, Tablet S)

- **iOS**

iPhone4, iPad2, ipad mini

- HTML5

웹브라우저	최소사양	권장사양	참고
Google Chrome	10	11	<a href="http://www.google.com/chrome/">http://www.google.com/chrome/</a>
Firefox	8	8	
Internet Explorer	7	9	
Opera	11	11	
Safari	4	4	



모바일 웹브라우저는 아래 명시된 브라우저만 지원합니다.

- iOS 기본 브라우저
- Android 기본 브라우저
- Android 구글 크롬 브라우저

## 파트 II.

---

## 기본



### 3.

## 넥사크로플랫폼 애플리케이션 구조

넥사크로 스튜디오에서 애플리케이션을 개발할 때는 XML 기반의 넥사크로플랫폼 프로그래밍 언어와 자바스크립트를 사용하며 애플리케이션을 테스트하거나 배포할 때는 자바스크립트로 변환된 파일이 배포됩니다. 이렇게 배포된 애플리케이션은 넥사크로플랫폼 통합 프레임워크를 기반으로 만들어지고 동작합니다.

이번 장에서는 넥사크로플랫폼 개발 시 사용되는 파일과 동작 방식에 대해 설명합니다. 작동원리에 배포과정도 포함되지만, 여기서는 배포에 대한 부분은 언급하지 않습니다. 배포에 대한 설명은 별도로 배포되는 관리자 가이드 문서를 참조하세요.

### 3.1 넥사크로플랫폼 애플리케이션 파일

넥사크로플랫폼 애플리케이션 개발 시 사용하는 파일은 넥사크로 스튜디오에서 사용되는 파일과 변환 이후 사용하는 파일로 구분할 수 있습니다. 모바일 앱 형태로 배포하거나 외부 라이브러리를 사용하는 경우에는 추가적인 파일이 필요할 수 있지만, 여기에서는 기본 프로젝트를 기준으로 설명합니다.

아래 표에서는 넥사크로 스튜디오에서 작성된 파일이 어떤 식으로 변환되어 배포되는지를 보여줍니다.

넥사크로 스튜디오	변환된 파일	비고
A.xprj		넥사크로 스튜디오에서 개발 시에만 참조
A.xadl	A.xadl.js	Build > Generate Application 메뉴에서 변환
default_typedef.xml		A.xadl.js 내 loadTypedefinition 함수에서 처리
globalvars.xml		A.xadl.js 내 on_loadGlobalVariables 함수에서 처리
Form_A.xfdl	./Base/Form_A.xfdl.js	Build > Generate File 메뉴에서 개별 변환 가능
default.xtheme	./_theme_/default/theme.css.js	

넥사크로플랫폼 애플리케이션이 실행 시에는 최초 호출되는 HTML 문서에서 변환된 A.xadl.js 파일을 호출하고 해당 파일에 설정된 로딩 과정을 거쳐 나머지 파일들의 정보를 얻습니다.

변환된 파일에 대한 상세한 설명은 다루지 않으며 여기에서는 넥사크로 스튜디오에서 넥사크로플랫폼 애플리케이션 작성 시 사용되는 파일에 대해 다룹니다.

## ADL File

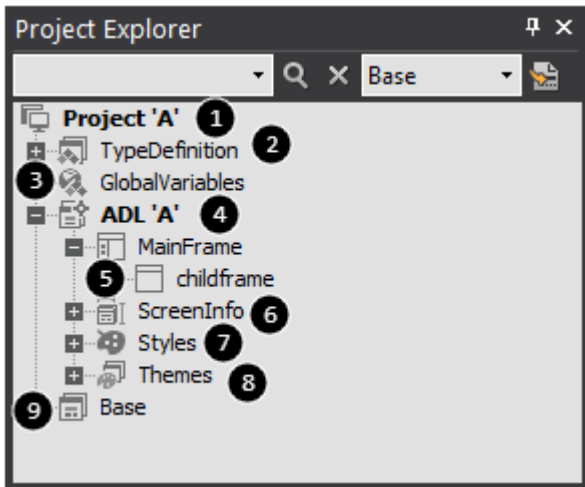
ADL 파일은 Application Definition Language의 약자로, Application구성에 필요한 모든 요소를 정의합니다.

ADL 파일이 정의하는 주요 요소들은 아래와 같습니다.

요소	설명
Object 배포 (link)	<ul style="list-style-type: none"> <li>- 애플리케이션 실행에 필요한 오브젝트들(Dataset, Grid등)을 나열하고 버전 정보를 등록합니다.</li> <li>- 이 정보는 Type Definition 파일에 저장되고, ADL에서는 url 링크 정보만 관리합니다.</li> </ul>
Theme 구성 (link)	<ul style="list-style-type: none"> <li>- 애플리케이션 화면에 보이는 디자인 형식을 정의합니다.</li> <li>- Theme는 이미지파일과 CSS의 조합으로 구성됩니다.</li> <li>- 이 정보는 Theme 파일에 저장되고, ADL에서는 url 링크 정보만 관리합니다.</li> </ul>
CSS (link)	<ul style="list-style-type: none"> <li>- 애플리케이션의 CSS를 정의합니다.</li> <li>- Theme에 등록된 CSS외에 추가로 CSS를 등록할 때 사용합니다.</li> <li>- 이 정보는 CSS 파일에 저장되고, ADL에서는 url 링크 정보만 관리합니다.</li> </ul>
Global Variable (link)	<ul style="list-style-type: none"> <li>- 애플리케이션 내의 여러 화면이 사용하는 공통 변수들을 등록합니다.</li> <li>- 공통 변수의 종류는 variant변수, Dataset 등을 포함한 Object들과 Image가 있습니다.</li> <li>- 이 정보는 Global Variable 파일에 저장되고, ADL에서는 url 링크 정보만 관리합니다.</li> </ul>
Global Script	<ul style="list-style-type: none"> <li>- 애플리케이션이 사용하는 공통 Function들을 정의합니다.</li> <li>- 애플리케이션 자체의 Event Function을 구현합니다.</li> </ul>

ADL파일은 넥사크로 스튜디오에서 자동으로 관리합니다.

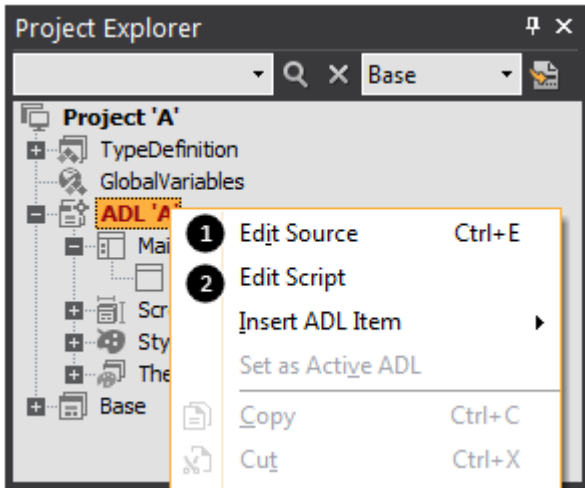
넥사크로 스튜디오상의 프로젝트 탐색 창에서 다루는 내용들이 대부분 ADL 파일에 저장되는 내용입니다. 다음은 넥사크로 스튜디오에서 프로젝트 탐색 창의 화면입니다.



	설명
①	프로젝트명입니다. 하나의 프로젝트에는 여러 개의 ADL이 등록될 수 있습니다
②	Type Definition정보를 저장합니다.
③	Global Variable정보를 저장합니다.
④	ADL명 입니다.
⑤	화면배치입니다. ChildFrame하나만 등록되었으므로 단일 프레임 모델입니다.
⑥	Screen 구성을 저장합니다.
⑦	CSS 구성을 저장합니다.
⑧	테마 구성을 저장합니다.
⑨	Form들을 정의하고 등록합니다.

Project Explorer에서 정의한 내용은 하나의 파일에만 저장되는 것이 아니라, 여러 개의 파일로 분리하여 저장됩니다. 그 중에 Root에 해당하는 파일이 ADL 파일 입니다.

ADL 파일은 프로젝트 탐색 창 외에 소스나 스크립트로 직접 편집이 가능합니다.



	설명
①	ADL에서 마우스 오른 버튼을 클릭한 후, “Edit Source”를 클릭하면 ADL의 내용을 편집할 수 있습니다.
②	ADL에서 마우스 오른 버튼을 클릭한 후, “Edit Script”를 클릭하면 ADL의 Global Script를 편집할 수 있습니다.

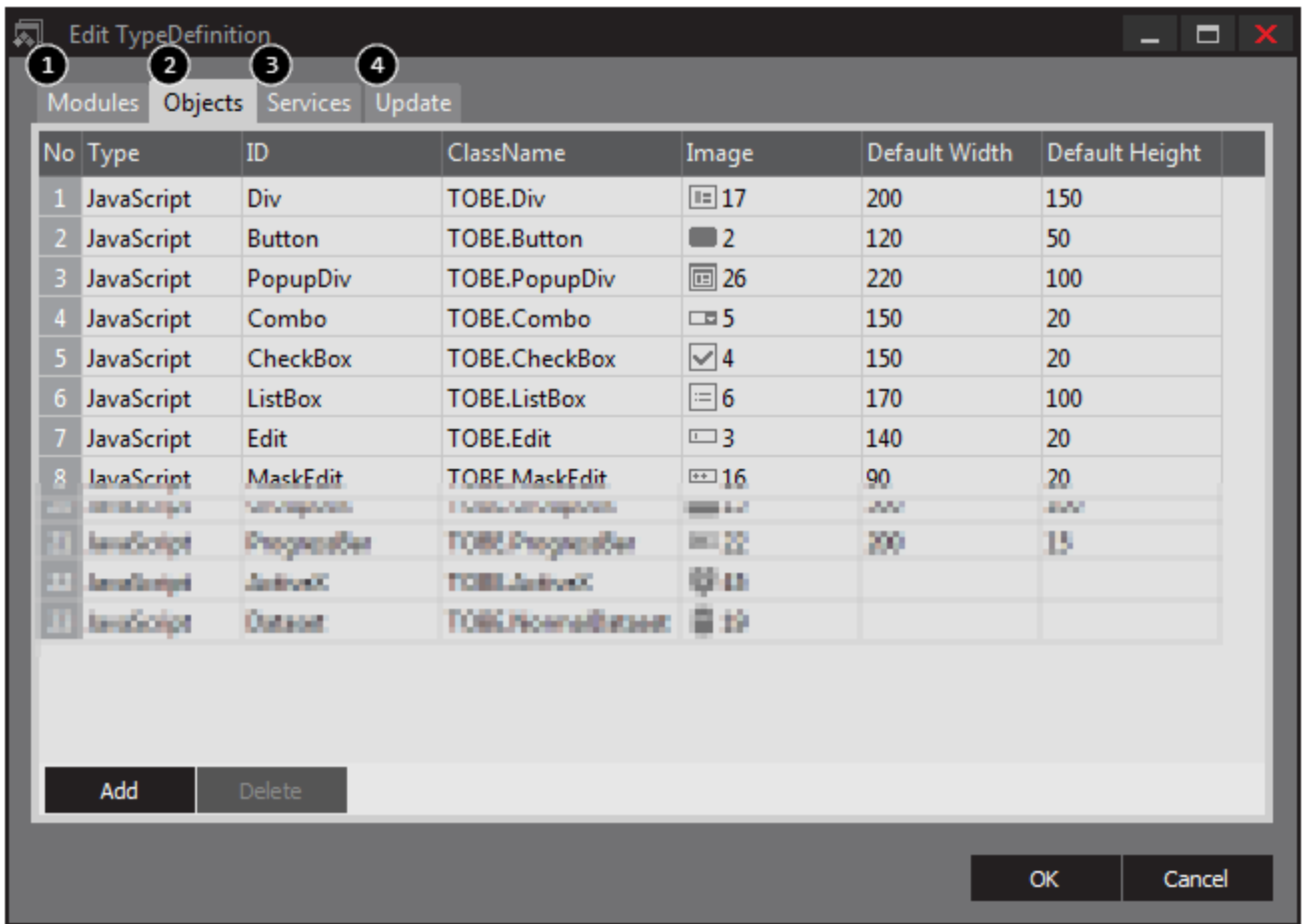
다음은 ADL의 내용을 편집하는 화면입니다.



## Type Definition File

Type Definition 파일은 넥사크로플랫폼 애플리케이션이 필요로 하는 파일에 관련된 정보를 갖고 있습니다. 넥사크로플랫폼에서 사용하는 컴포넌트는 자바스크립트를 기반으로 만들어졌습니다. 로직이 구현된 자바스크립트 파일을 모듈이라고 하고 이 중에서 애플리케이션 개발에 사용할 오브젝트를 개발 시 선택할 수 있습니다. 그 외 애플리케이션 개발 시 서비스 그룹을 정의하고 배포와 관련된 옵션을 설정합니다.

다음은 Type Definition 편집 창의 화면입니다.



Type Definition 파일이 정의하는 주요 요소들은 아래와 같습니다.

	요소	설명
①	모듈 정보	자바스크립트로 개발된 모듈 정보를 담고 있습니다. - 자체 개발된 모듈을 추가할 수 있습니다. - js 또는 json 파일 형태로 등록할 수 있습니다. 각 파일간에 의존성을 가지는 경우에는 순서에 맞게 등록해야 합니다.
②	오브젝트 정보	애플리케이션 실행에 필요한 오브젝트 정보를 담고 있습니다. - 지정된 오브젝트가 배포에 포함됩니다. - 각 오브젝트는 모듈을 기반으로 지정됩니다.
③	서비스 그룹	프로젝트 규모에 따라 내부적으로 서비스 그룹을 정의합니다.

	요소	설명
		- 각 서비스 그룹은 개별적으로 캐시 수준을 설정할 수 있습니다.
4	배포 옵션	애플리케이션의 배포 관련 정보(런타임 버전)를 지정합니다.

## Global Variable File

Global Variable 파일은 넥사크로플랫폼 애플리케이션의 여러 화면들이 사용하는 공통 변수들을 등록합니다.

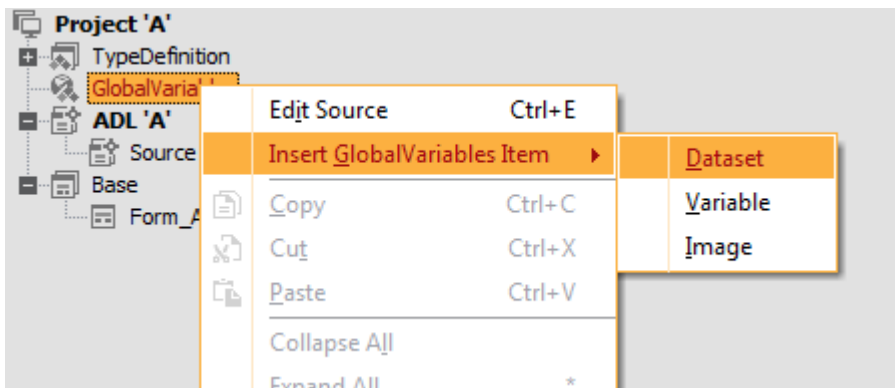
공통 변수로 등록할 수 있는 변수의 종류로는 Variant 변수, Dataset 등을 포함한 Object들과 Image가 있습니다.

Global Variable 파일이 정의하는 변수는 아래와 같습니다.

요소	설명
Dataset 변수	- Dataset을 등록하고 element들을 설정합니다.
Variable 변수	- Variant Type의 변수를 등록합니다.
Image 변수	- 공통으로 사용될 이미지를 등록합니다.

다음은 Global Variable중 Dataset을 등록하는 화면입니다.

Project > GlobalVariable > Insert GlobalVariables Item > Dataset



Global Variable 파일을 일괄적으로 편집하고자 할 때에는 아래 메뉴에서 소스를 열어 편집할 수 있습니다.

Project > GlobalVariable > Edit Source

## 테마 / 스타일

넥사크로플랫폼에서 스타일과 테마란 화면 상에 보이는 UI 요소들의 디자인을 정의하는 것을 의미합니다. 버튼을 예로 들면, 투명도, 폰트, 색상, 그림자, 번짐과 기울임 등의 효과를 버튼 컴포넌트에게 주는 것을 의미합니다.

테마와 스타일의 차이점은 이미지 파일의 포함 여부입니다. 테마는 스타일에 이미지를 추가한 형태입니다. 그러므로 이미지를 사용하는 테마는 스타일과 비교하면 좀 더 다양한 UI 디자인을 할 수 있습니다.

## FDL 파일

FDL 파일은 사용자 화면 Form을 정의합니다. 사용자에게 보이는 화면 배치뿐만 아니라 UI 로직 수행을 위한 스크립트를 포함한 다양한 정보를 담고 있습니다.

FDL 파일은 ADL 파일과 더불어 넥사크로플랫폼 애플리케이션에서 가장 중요한 파일입니다.

FDL 파일이 정의하는 주요 요소는 다음과 같습니다.

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 Function외에도 필요한 Function들을 스크립트로 작성합니다.
Style	- Form에서 사용하는 Style을 등록합니다. 여기서 사용하는 Style은 ADL에서 등록한 Style보다 우선권을 갖습니다.
BindItem	- Form, 컴포넌트, Invisible Object들과 Dataset을 연결하는 BindItem Object를 설정합니다.



## 3.2 주요 화면 요소

넥사크로플랫폼 애플리케이션은 사용자가 조작할 수 있는 애플리케이션 UI를 보여주기 위해 여러 가지 요소를 포함하고 있습니다. 기본적인 화면 요소는 아래와 같습니다.

### 프레임을 이용한 화면배치

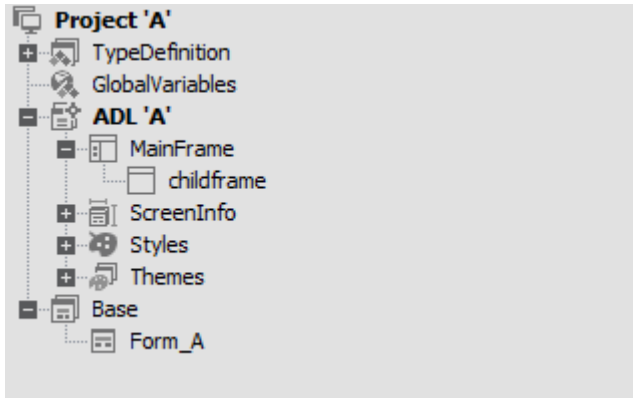
넥사크로플랫폼 애플리케이션은 화면 처리를 위한 기본 프레임 구조를 제공합니다. 최상위 MainFrame 아래에 ChildFrame을 가지고 그 아래에 화면을 배치합니다. 프레임은 Form을 보여주기 위한 무대장치라고 할 수 있습니다. 각 화면을 이루는 Form이 따로 떨어져 있지만 같은 무대 위에서 보여주기 때문에 공통으로 사용할 수 있는 요소를 배치해 화면 간 다양한 요소를 공유할 수 있습니다.

기본 제공되는 프레임은 MainFrame, ChildFrame으로 나뉘어 지며 화면 레이아웃에 따라 FrameSet, VFrameSet, HFrameSet, TileFrameSet을 추가할 수 있습니다.

프레임 종류	Frame	설명
Root Frame	MainFrame	<ul style="list-style-type: none"> <li>- 최상위 프레임입니다. 즉, Root Frame에 해당합니다.</li> <li>- 하위로 Node Frame 또는 ChildFrame을 가질 수 있습니다.</li> <li>- 하위에 ChildFrame을 가진 경우에는 프레임을 추가할 수 없습니다.</li> </ul>
Terminal Frame	ChildFrame	<ul style="list-style-type: none"> <li>- 하위로 어떤 프레임도 가질 수 없습니다.</li> <li>- 하위로 하나의 폼만 가질 수 있습니다.</li> </ul>
Node Frame	FrameSet	<ul style="list-style-type: none"> <li>- 특별한 형태 없이 하위 프레임을 배치합니다.</li> <li>- 2개 이상의 하위 프레임이 추가되면 계단식으로 배치하며 위치 속성값을 지정하면 해당 위치에 배치합니다.</li> <li>- 하위로 Node Frame 또는 ChildFrame을 가질 수 있습니다.</li> </ul>
	VFrameSet	<ul style="list-style-type: none"> <li>- 세로 형태로 하위 프레임을 배치합니다.</li> <li>- separatesize 속성으로 하위 프레임 배치 비율을 지정합니다.</li> <li>- 하위로 Node Frame 또는 ChildFrame을 가질 수 있습니다.</li> </ul>
	HFrameSet	<ul style="list-style-type: none"> <li>- 가로 형태로 하위 프레임을 배치합니다.</li> <li>- separatesize 속성으로 하위 프레임 배치 비율을 지정합니다.</li> <li>- 하위로 Node Frame 또는 ChildFrame을 가질 수 있습니다.</li> </ul>
	TileFrameSet	<ul style="list-style-type: none"> <li>- 바둑판 형태로 하위 프레임을 배치합니다.</li> <li>- separatetype, separatecount 속성으로 가로, 세로 방향에 배치될 하위 프레임을 지정합니다.</li> <li>- 하위로 Node Frame 또는 ChildFrame을 가질 수 있습니다.</li> </ul>

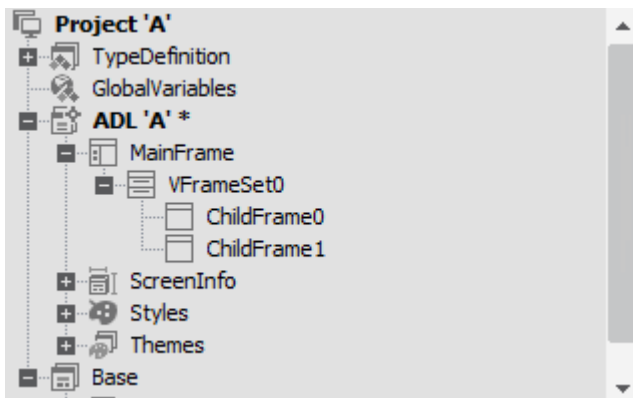
## SDI (Single Document Interface)

프로젝트 생성 시 Default 템플릿을 선택하고 다른 추가적인 작업을 하지 않았다면 하나의 프레임만으로 프로젝트가 구성됩니다.

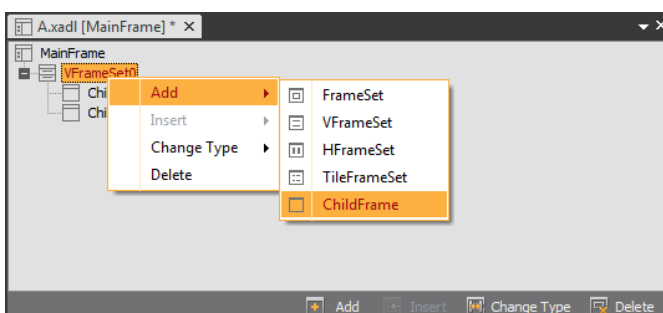


## MDI (Multi Document Interface)

상단 메뉴, 하위 메뉴, 콘텐츠, 툴바 형식으로 화면을 구성하는 경우에는 전체적인 레이아웃 구조를 원하는 형식에 맞게 구성해야 합니다. 넥사크로플랫폼에서 MDI 구성은 Default 템플릿 선택 후 필요한 프레임을 추가하거나 HFrame, VFrame 템플릿 또는 사용자가 지정한 프로젝트 템플릿을 선택해 만들 수 있습니다.



프레임 추가는 프로젝트 탐색기에서 MainFrame 또는 Node Frame 선택 후 컨텍스트 메뉴에서 [Insert Frame] 항목을 선택하거나 MainFrame 또는 Node Frame 항목을 더블 클릭 후 나타나는 편집화면 컨텍스트 메뉴에서 [Add] 항목을 선택해 추가할 수 있습니다.



## Form의 구성

Form은 실질적인 화면 UI를 구성하는 요소입니다. 프레임 위에 Form이 올려져서 사용자가 조작할 수 있는 화면으로 보입니다.

Form은 Visible 영역, Invisible 영역과 Logic 영역으로 크게 3개의 영역으로 구성되어 있습니다.

- Visible 영역은 화면에 보이는 부분으로 visible Object(이하 컴포넌트)들의 배치로 만들어집니다.
- Invisible 영역은 Invisible Object들의 조합으로 만들어집니다.
- Logic 영역은 Script로 직접 코딩하는 부분으로 컴포넌트와 Invisible Object들을 제어하는 부분입니다.

## 3.3 서버와의 Data연동

넥사크로플랫폼 애플리케이션은 기업 내에서 사용하는 데이터를 사용자가 쉽게 조작할 수 있도록 화면 상에 표현해주는 역할을 담당하기도 합니다. 이런 상황에서 데이터를 관리하는 서버와 애플리케이션 간 연동하는 작업을 필요로 합니다.

### X-API

X-API는 넥사크로플랫폼 애플리케이션과 연동할 수 있는 데이터 형식을 생성해주는 라이브러리입니다. X-API를 사용하지 않아도 정해진 명세에 따라 Data Service를 개발한다면 넥사크로플랫폼 애플리케이션과 해당 서비스 데이터와 연동할 수 있습니다.



X-API는 Java 개발 환경을 기반으로 제공됩니다. 상세한 문서는 X-API 라이브러리에 포함된 문서를 확인하세요.

## 4.

# XML 파일 구조

애플리케이션 실행을 위해 넥사크로플랫폼 런타임은 Form 파일 외에 많은 파일이 있어야 하고, 넥사크로 스튜디오는 이 파일들을 자동으로 생성 관리합니다.

이 파일들은 모두 XML Format으로 되어 있습니다. 여기서는 넥사크로플랫폼이 제시하는 XML 구조를 설명합니다.

## 4.1 ADL XML Format

```
<ADL>
  <TypeDefinition/>
  <GlobalVariables/>
  <Application></Application>
  <Script><![CDATA[ ... ]]></Script>
</ADL>
```

```
<Application>
  <Style/>
  <Layout>
    <MainFrame>
      <ChildFrame/>
    </MainFrame>
  </Layout>
</Application>
```

```
<Application>
  <Style/>
  <Layout>
```

```

    <MainFrame>
      <FrameSet>
        <ChildFrame/>
      </FrameSet>
    </MainFrame>
  </Layout>
</Application>

```

## 4.2 FDL XML Format

```

<FDL>
  <TypeDefinition/>
  <Form>
    <Style/>
    <Layouts>
      <Layout>
        <!-- UI Component -->
      </Layout>
    </Layouts>
    <Objects>
      <!-- Invisible Object -->
    </Objects>
    <Bind>
      <BindItem/>
    </Bind>
    <Script><![CDATA[ ]]></Script>
  </Form>
</FDL>

```

## 4.3 Type Definition XML Format

```
<TypeDefinition>
  <Modules>
    <Module/>
  </Modules>
  <Components>
    <Component/>
  </Components>
  <Services>
    <Service/>
  </Services>
  <Update>
    <item/>
  </Update>
</TypeDefinition>
```

## 4.4 Global Variable XML Format

```
<GlobalVariables>
  <Images>
    <Image/>
  </Images>
  <Dataset/>
  <Variable/>
</GlobalVariables>
```

## 5.

# 넥사크로플랫폼 스크립트 언어

넥사크로플랫폼에서 애플리케이션 개발 시 사용하는 스크립트는 HTML5 명세서에서 사용하는 스크립트를 기준으로 개발되었습니다. 따라서 기존 웹 개발자가 별도로 스크립트 언어를 익히지 않아도 넥사크로플랫폼 스크립트를 작성할 수 있습니다.

하지만 HTML5은 계속 개발되고 있으며 아직 제공되지 않지만, 기업 내에서 필요한 기능을 지원하기 위해 넥사크로플랫폼 자체적으로 지원하는 스크립트도 같이 제공됩니다. 그래서 최적의 성능을 내는 애플리케이션을 만들기 위해서는 기본적인 스크립트 언어와 함께 넥사크로플랫폼 스크립트 언어에 대한 기본적인 지식이 필요합니다.

이번 장에서는 기존 버전에서 변경된 스크립트 언어를 설명합니다.

## 5.1 유효범위(Scope)

유효범위는 접근 가능한 변수의 범위와 관련된 내용을 설명합니다. 유효범위 또는 영역이라고 표기하며 영문 표기 그대로 사용할 때도 많습니다. 업무에 사용하는 애플리케이션은 여러 개의 화면을 동시에 조작하고 하나의 변수를 여러 화면에서 참조하는 경우가 많은데 이러면 어떤 식으로 원하는 자원에 접근할 수 있는지 유효범위를 통해 알 수 있습니다.

### this

ADL 또는 폼에서 사용되는 항목에 대해 유효범위를 지정할 때는 항상 this를 사용합니다. 변수뿐 아니라 속성, 폼 간의 참조 시에도 적절한 유효범위를 지정해주어야 합니다.

폼 내에서 변수는 아래와 같이 선언합니다.

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

함수를 선언할 때도 유효범위를 지정해주어야 합니다. 또한, 함수 내에서 폼 내에 선언된 변수를 참조할 때도 유효범



위를 지정해주어야 합니다.

```
this.formvalue = 4;

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

trace나 alert과 같은 메소드는 지정된 유효범위에 따라 다른 메소드가 실행됩니다. alert()만 사용하는 것과 this.alert()을 사용하는 것은 서로 별개의 메소드를 실행하는 것입니다.

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

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

함수 선언 시에도 유효범위를 지정해 해당하는 애플리케이션 또는 품의 멤버로 포함되도록 합니다.

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

eval 함수로 품을 접근하는 경우에도 this를 사용해야 합니다.

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



이전 버전에서 사용하던 것처럼 함수를 직접 선언할 수도 있지만, Global로 처리되며 이후 지원이 중단될 수 있으므로 권장하지 않습니다.

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

## Global

넥사크로플랫폼 스크립트 내에서 유효범위를 지정하지 않은 변수나 함수는 모두 최상위 Global 멤버로 처리됩니다.

아래와 같이 유효범위를 지정하지 않고 사용된 스크립트는 모두 Global로 처리됩니다. var 는 변수를 선언한다는 의미가 있을 뿐 유효범위를 지정하는 것은 아닙니다.

```
globalvar = 2;
var localvar = 3;
```

함수 내에 사용된 변수라도 유효범위를 지정하지 않으면 모두 Global로 처리됩니다.

```
this.test = function()
{
  value = 3;
  obj.prop = 3;
  localvar = 3;
}
```

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:Button, e:nexacro.ClickEventInfo)
{
  test();
  this.test();
}
```

아래와 같이 유효범위 없이 지정된 함수는 Global로 처리됩니다.

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

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



Alert(), Confirm(), Trace() 메소드를 Global로 사용할 수 있습니다. 이전 버전처럼 폼 메소드로 사용하려면 this로 유효범위를 지정해주어야 합니다.



dialog(), open(), exit(), transaction() 과 같은 애플리케이션 또는 폼 메소드는 해당하는 유효범위를 지정해주어야 합니다.

```
//application
application.open("");
application.exit("");
application.transaction("");

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

//ADL
this.open("");
this.exit("");
this.transaction("");
```



애플리케이션 전체에서 사용하고 싶은 변수는 Global 변수가 아닌 애플리케이션 변수로 등록해서 사용하는 것을 권장합니다.

## Expr

expr 스크립트는 넥사크로플랫폼 내부적으로 바인딩 된 Dataset을 기준으로 처리되기 때문에 별도의 유효범위를 지정하지 않아도 사용할 수 있습니다. 단 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'");
```

바인딩 된 Grid, Dataset, Cell을 지정하는 경우에는 아래와 같은 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.



폼 내에 있는 함수에 접근할 때 comp.parent.func() 형식을 사용하지 않고 this를 사용하게 되면 오류로 처리됩니다. expr 스크립트에서 this는 cell을 가리킵니다.

```
<Cell text="expr:this.func01()"/>
```

컴포넌트 속성으로 Expr과 Text가 같이 있는 경우에 화면에 보이는 텍스트를 반환하는 displaytext 속성을 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.displaytext); // "2"
```

## 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 );
Frame.lookup( strid );
Application.lookup( strid );
EventSyncObject.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 );
```

## 5.2 이벤트 핸들러

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

### 메소드

스크립트 내에서 이벤트 처리를 위해 이벤트 핸들러를 추가, 설정하거나 삭제할 수 있는 메소드를 제공합니다. 기존 버전에서는 addHandler, setHandler, removeHandler 3개의 메소드를 제공했지만, 해당 메소드는 더는 지원하지 않습니다

1. addEventHandler( eventid, funcobj, target )
2. addEventHandlerLookup( eventid, funcstr, target )
3. setEventHandler( eventid, funcobj, target )
4. setEventHandlerLookup( eventid, funcstr, target )
5. removeEventHandler( eventid, funcobj, target )
6. 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 애플리케이션 성능에 영향을 미칠 수 있으므로 필요한 경우만 사용을 권장합니다.



btn00.onclick.addHandler(button00\_onclick) 형식은 더는 지원하지 않습니다.



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.

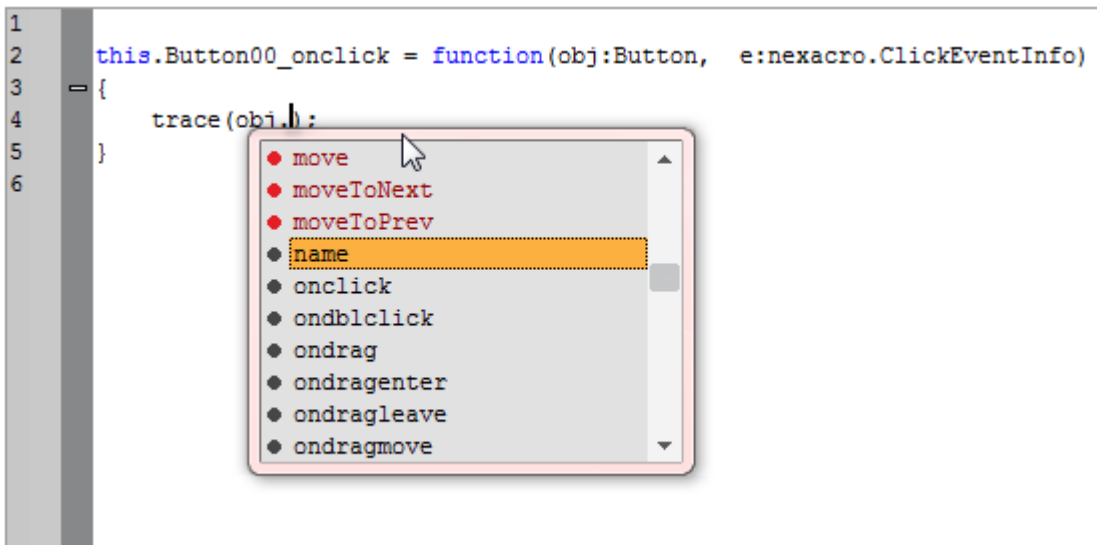
```
this.Button00.addEventHandler("onclick", this.Button00_onclick, this);
```

## 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:Button, e:nexacro.ClickEventInfo)
{
    trace(obj.text);
}
```

입력된 타입 값을 기준으로 넥사크로 스튜디오에서 코드 힌트 기능을 지원합니다.



해당 코드를 자바스크립트 코드로 변환할 때는 타입 값을 제거합니다.

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



오브젝트 타입을 지정하지 않아도 애플리케이션이 실행하는데 영향을 미치지 않습니다.



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

## 5.3 Setter

ECMAScript 5부터는 Setter/Getter를 사용해 변수의 접근 제한을 처리할 수 있습니다. 하지만 IE8 이하 웹브라우저에서 지원할 수 없는 명세서이기 때문에 넥사크로플랫폼에서는 별도의 set 메소드를 제공합니다.

### 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 속성값을 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 속성에 접근하는 방식도 set 메소드를 사용해야 합니다. 추가된 set 메소드에는 하위 속성까지 포함하고 있습니다. 기존 속성은 값을 가져올 때만 사용할 수 있습니다.

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



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

단, CSS 코드는 기존과 동일합니다.



```
Button
{
    background : red;
}
```



액티브X(플러그인)나 애플릿 컴포넌트를 사용할 때 속성을 지정하거나 메소드를 호출하는 방식도 아래와 같이 사용하게 됩니다.

```
this.ActiveX00.setProperty("prop1", 3 );
var v = this.ActiveX00.getProperty("prop1");
```

```
this.ActiveX00.callMethod("method1", arg1, arg2 );
var v = this.ActiveX00.callMethod("method2");
```

## 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.

```
System.getCursorX();  
System.getCursorY();  
System.getScreenWidth();  
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)라고 하며 속성 자체를 문자열로 출력하면 [object ButtonControl]라고 표시됩니다.

## 5.4 기타 변경 사항

### nexacro 메소드

ECMAScript에서 지원하지 않는 메소드를 넥사크로플랫폼 자체적으로 추가해 사용하던 것을 표준에 따라 변경하면서 더는 지원하지 않습니다. 대신 해당 메소드는 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
Export.export() → Export.exportData()
Buffer.delete() → Buffer.remove()
VirtualFile.delete() → VirtualFile.remove()
```

## 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.

## 오브젝트 명 생성 시 제약

컨테이너의 멤버인 속성, 메소드명과 같은 ChildName을 만들 수 없습니다. 예를 들어 폼은 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>
```

## 6.

---

# Frame Tree

이 장에서는 Form의 상호관계를 좀 더 구체적으로 도식화하여 설명합니다.

## 6.1 표기법

Frame의 상호관계를 도식화할 때 사용하는 표기법을 설명합니다.

### box

collection 상의 Item으로 실제 이름을 의미하지는 않습니다.



### 연결선

- 1:1 연결을 표기합니다. 원 모양의 연결 아이콘이 있는 부분이 Child가 됩니다.



- 1:N 연결을 표기합니다. 까마귀발(Crow's foot) 연결 아이콘이 있는 부분이 Child가 됩니다.



## 설명

- 아이템 간의 스크립트 문법은 연결선 위에 표기합니다.
- Parents에서 Child로의 스크립트는 연결선 상단에 표기합니다.
- Child에서 Parents로의 스크립트는 연결선 하단에 표기합니다.



위 표기는 다음을 의미합니다.

- parent와 child가 1:N의 관계입니다.
- parents.id 스크립트로 child에 접근할 수 있습니다.
- parents.child[index] 스크립트로 child에 접근할 수 있습니다.
- child.parent 스크립트로 parent에 접근할 수 있습니다.

## 용어정의

용어	설명
element	component의 property, method, event를 통칭하여 component의 element라고 합니다.

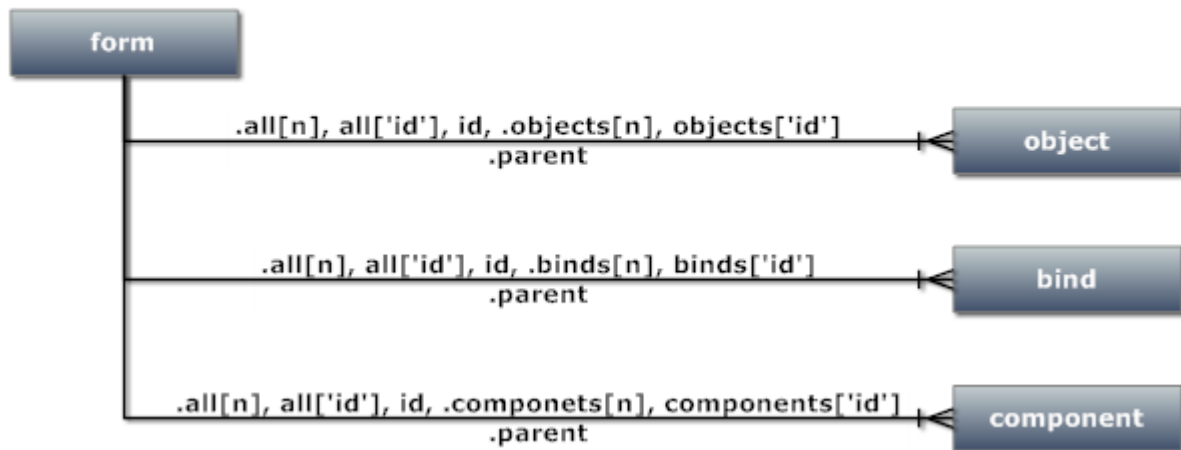
## 공통으로 사용하는 Syntax

- '[', ']' 내에는 index 또는 'id'가 올 수 있습니다.  
즉, this.all[0]과 this.all['btn0'] 둘 다 동일 component를 가리킬 수 있습니다.

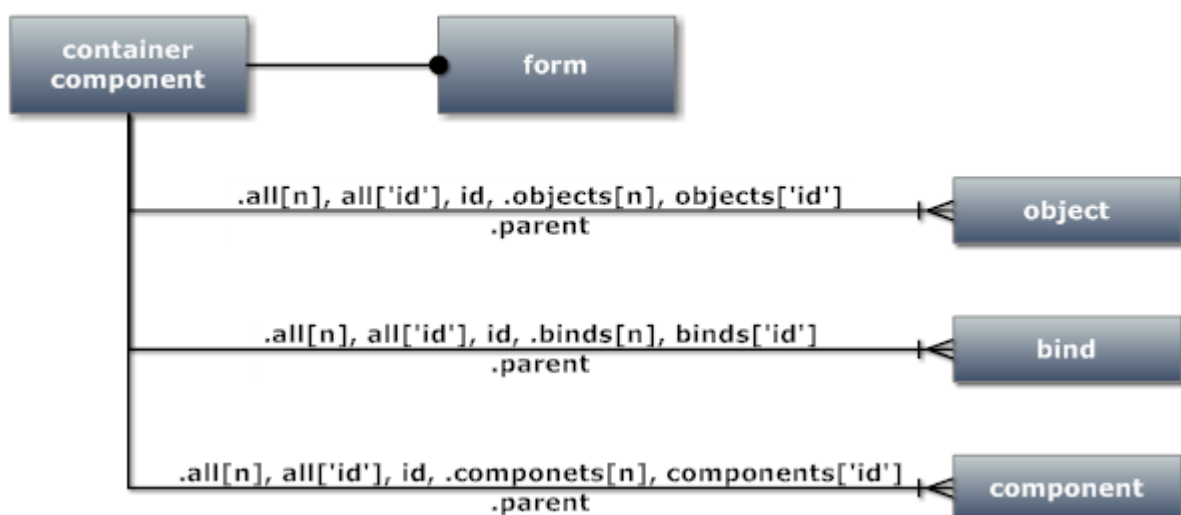
## 6.2 Form

### 관계도

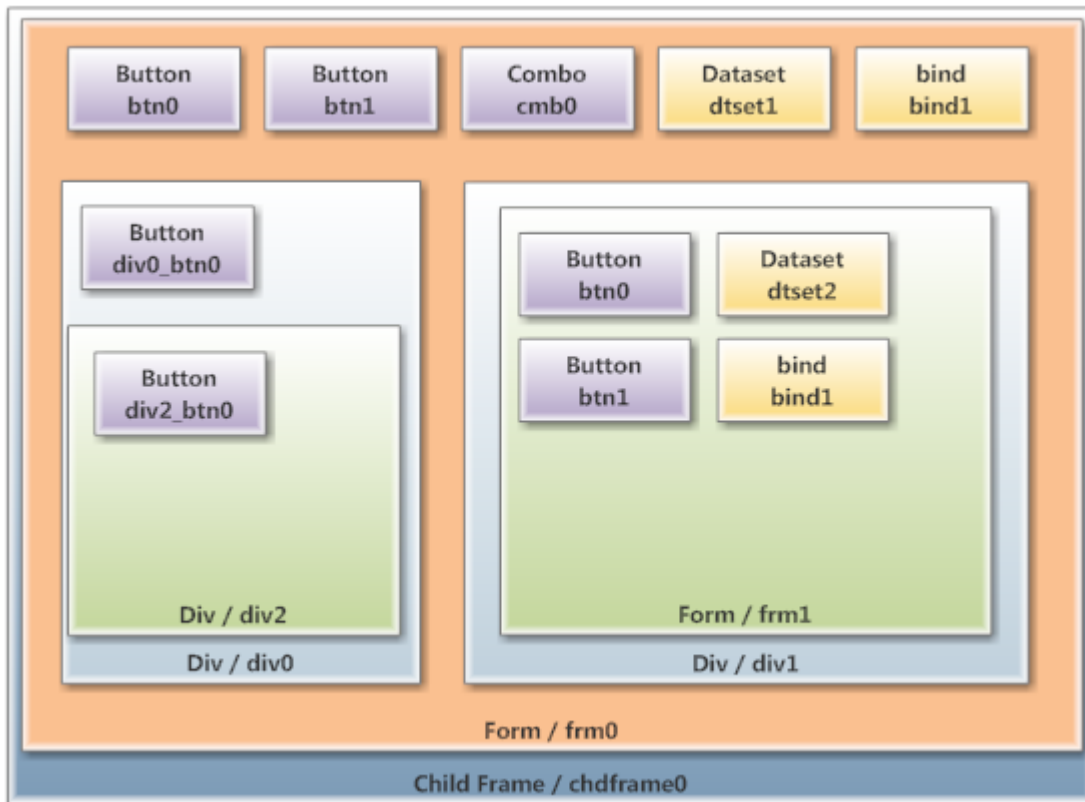
- 일반 Form인 경우



- container component인 경우



## 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 없다. 즉, btn0와 div0\_btn0와 div2\_btn은 동일 Level이 아니므로, 같은 id로 지정하여도 무방합니다.



## component / invisible object / bind 의 Script 접근

```
this.classname = "frm0"
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
```

## container component에 Script 접근

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

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

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

## form을 연결한 container component에 Script 접근

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

```
// (frm0안의 script에서 접근할 경우) frm1에 있는 dtset1의 접근
this.name = "frm0"
this.div1.btn0.name = "btn0" // "frm1"을 흡수한 "div1"으로만 접근할 수 있습니다.
```

## parent의 Script 사용

```
this.div1.dtset2.parent.parent.classname = "frm0"
this.div1.dtset2.parent.name = "div1" // frm1은 div1에 연결되면서 그 특성을 잃게 됩니다.
this.div1.dtset2.parent.classname = undefined // frm1은 div1에 연결되면서 그 특성을 잃게 되어
classname이 존재하지 않게 됩니다.
this.div1.dtset2.parent.name = "div1"
```

## container component의 element사용

form을 연결한 container component는 form처럼 components, invisible object, bind (이하 element)를 가질 수 있습니다.

form을 연결하지 않은 container component는 invisible object, bind를 가질 수 없고 component만 가질 수 있습니다.

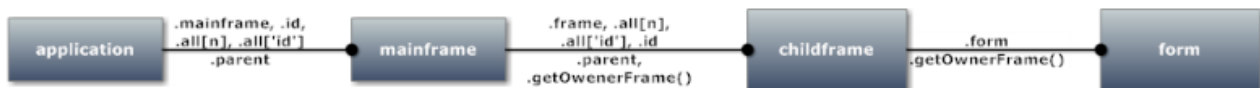
container 기능이 없는 component와 invisible object, bind는 element를 가질 수 없습니다.

container component는 연결된 form의 특성을 가집니다.

- form script에서 this로 접근할 경우, 연결된 form 내부의 element뿐만 아니라 container component의 element에도 접근할 수 있습니다.
- container component가 속한 form에 this.component\_name으로 접근할 경우, 연결된 form의 모든 elements에 접근할 수 있습니다.

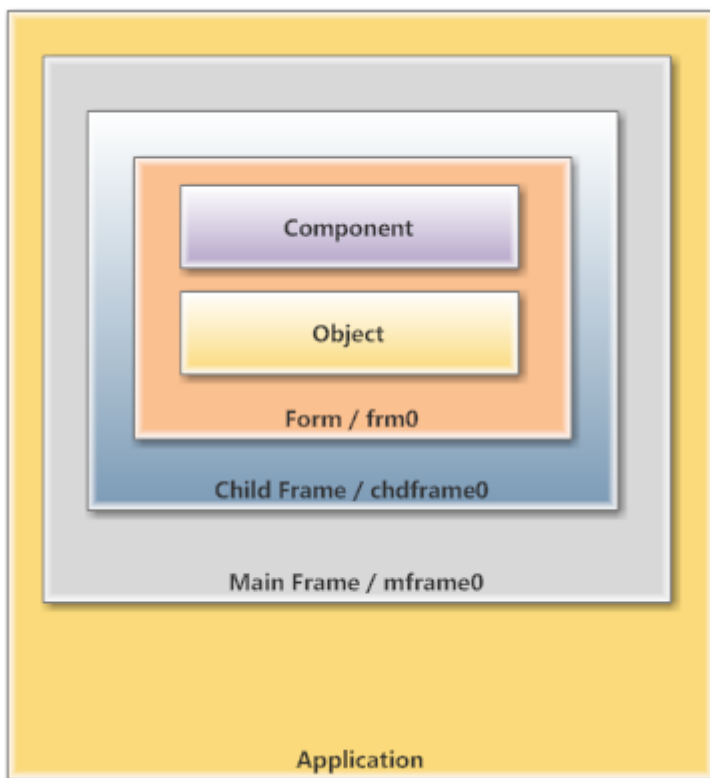
## 6.3 Application

### 관계도



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

### Script 예시 화면



## application에서 form의 script 접근

```
// frm0로의 접근 (ChildFrame인 경우)
application.mainframe.frame.form.name == "frm0"
application.mframe0.frame.form.name == "frm0"
application.mframe0.chdframe0.form.name == "frm0"
application.all[0].all[0].form.name == "frm0"
application.all["mframe0"].all["chdframe0"].form.name == "frm0"
this.mainframe.frame.form.name == "frm0"
```

## form에서 application의 script 접근

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

## 7.

# 스타일(CSS)과 테마 관리

스타일(CSS)과 테마는 넥스코로플랫폼 화면을 구성하는 화면요소들을 디자인하는 기능을 의미합니다.

스타일과 테마를 적용할 수 있는 화면 요소들은 컴포넌트, 화면 Form, Frame, TitleBar, StatusBar, ScrollBar등이 있습니다.

## 7.1 스타일 / 테마 개요

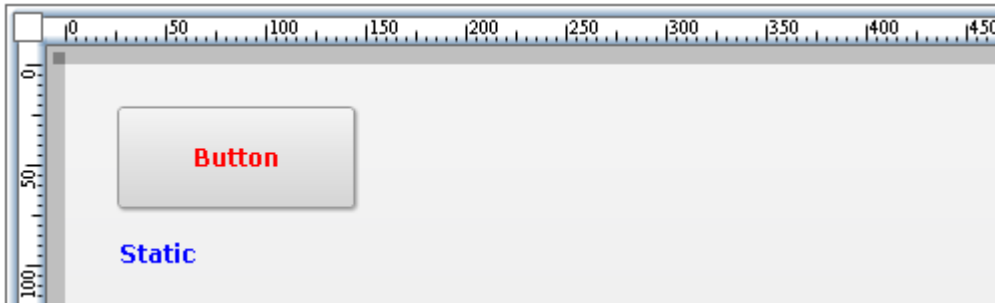
넥스코로플랫폼에서 스타일과 테마는 화면 상에 보이는 UI 요소들의 디자인을 정의하는 것을 의미합니다. 버튼을 예로 들면, 투명도, 글꼴, 색상, 그림자, 번짐, 기울임 등의 효과를 버튼 컴포넌트에 지정하는 것을 의미합니다.

### 스타일의 정의 및 적용

스타일은 컴포넌트별로 정의합니다.

각각의 컴포넌트는 스타일 적용이 가능한 속성이 있으며, 해당 속성값을 설정해 스타일을 정의합니다.

```
Button
{
    color : "red";
}
Static
{
    color : "blue";
}
```



스타일을 정의할 때는 선택자(Selector)를 사용합니다. 기본적인 CSS에 대한 사전지식이 있다는 가정하에 설명합니다.

넥사크로플랫폼이 지원하는 선택자는 다음과 같습니다.

### 기본 선택자

Type Selector, Class Selector, ID Selector, Child Selector

### 기타

Grouping, Pseudo-Classes

- 타입선택자(Type Selector) : “Button”과 같이 컴포넌트명을 지정합니다.

Syntax	Name (Element)
예	Button {color : green;}
의미	모든 Button의 color는 green으로 정의합니다.

- 클래스선택자(Class Selector) : 특정 클래스로 지정된 항목의 스타일을 적용합니다.

```
<Button id="Button00" text="Button00" left="50" top="20" width="100" height="40"
cssclass="BlueButton"/>
```

Syntax	.(Period)
예	.BlueButton {color : blue;}
의미	BlueButton으로 지정된 클래스에 대해 blue 생상을 적용합니다.



아래와 같이 컴포넌트별로 클래스 선택자를 지정할 수도 있습니다.

```
Button.BlueButton {color : blue;}
```

- ID 선택자(ID Selector) : “Button00”과 같이 컴포넌트의 ID를 지정합니다.

```
<Button id="Button01" text="Button00" left="50" top="20" width="100" height="40"/>
```

Syntax	#
예	Button#Button00 {color : yellow;}
의미	모든 Button중에 Button00이라는 ID를 가진 Button의 color는 yellow로 정의합니다.

- 자식선택자(Child Selector) : 모든 컴포넌트의 하위를 지정합니다.

Syntax	>
예	Div>Button { color : blue; }
의미	모든 Div 안의 Button의 color는 blue로 정의합니다.



자식선택자는 최상위 레이아웃에 속한 컴포넌트만 사용할 수 있습니다. 예를 들어 Div 내에 포함된 Button의 스타일을 컴포넌트 수준의 child 선택자로 지정할 수는 없습니다. 필요한 경우에는 ID 선택자를 사용해야 합니다.

```
Form>Button (O)
Form>Div>Button (X)
Form>#div00>#button00 (O)
```

- 유사클래스(Pseudo-classes) : 선택자로 지정할 수 없는 상태 정보를 지정합니다.  
컴포넌트에 따라 선택할 수 있는 상태 정보는 달라집니다.
  - UI 요소 유사클래스  
enabled(default), disabled, focused, mouseover, pushed, selected, readonly
  - 구조적 유사클래스  
root, nth-child(), nth-last-child(), first-child, last-child, only-child, empty

Syntax	:
예	Button:focus {color : black;}
의미	모든 Button에 focus가 되면 해당 Button의 color는 black으로 정의합니다.

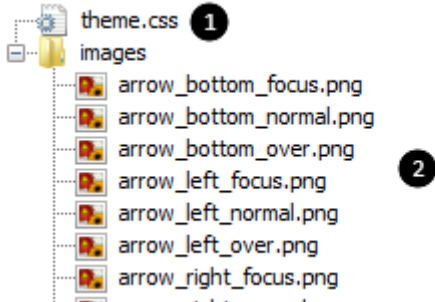
- 그룹핑(Grouping) : 같은 값(declarations)을 갖는 여러 개의 선택자를 컴포넌트명으로 지정합니다.

Syntax	,(Comma)
예	Button, Combo, Calendar {font : Dotum,9;}
의미	모든 Button에 Button, Combo, Calendar의 font는 “Dotum 9”으로 정의합니다.



## 테마의 정의

테마는 스타일과 이미지 파일이 조합된 형태입니다.



1. 테마에서 사용하는 스타일을 정의합니다.
2. 스타일에서 사용하는 이미지를 포함합니다.



애플리케이션은 하나의 테마만을 사용할 수 있습니다.

넥사크로플랫폼은 default, black, gray, blue의 4가지 Theme파일을 제공합니다.

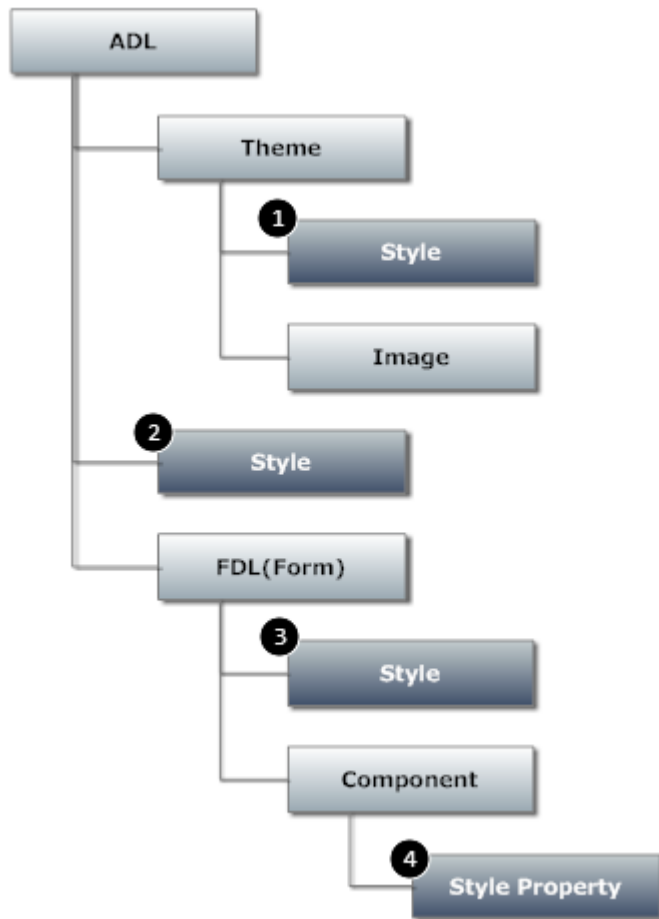
제공되는 테마는 넥사크로 스튜디오의 Theme, Style Editor창을 통해 필요한 부분을 수정하고 다른 이름으로 저장해 사용할 수 있습니다.

테마는 넥사크로플랫폼에서 꼭 필요한 기본정보를 가지고 있으므로 필요한 부분만 추가, 수정해야 합니다.

기본정보가 삭제되거나 변경할 경우, 화면 출력에 문제가 생길 수 있습니다. 예를 들면 “#combolist” 와 같이 목록 표시를 처리하는 설정이 없으면 Combo 컴포넌트를 펼쳤을 때 목록이 정상적으로 나타나지 않는 경우가 발생합니다.

따라서 가능한 테마는 새로 생성하는 것보다는 기본으로 제공되는 테마를 수정해서 다른 이름으로 저장하거나 복사해서 사용하도록 합니다.

## 스타일/테마의 등록 및 적용



1. ADL에 속한 테마 내의 스타일
2. ADL에 속한 스타일
3. FDL에 속한 스타일
4. FDL내의 컴포넌트 스타일

컴포넌트에 스타일이 적용되는 순서는 1 > 2 > 3 > 4 입니다. 그러므로 중복된 스타일 값을 정의한 경우, 마지막 스타일인 4번 스타일이 적용되어 화면에 보입니다.

## 스타일과 테마의 적용 대상

Style은 하나의 Form에 선언해서 사용할 수 있으며, Global하게 사용하기 위해 Project(ADL)에 등록하거나 Theme에 포함해 사용할 수도 있습니다.

Style의 적용대상은 단순한 컴포넌트뿐만 아니라, 다양한 화면요소에 적용됩니다. 다음은 그 적용 대상들을 표로 나타낸 것입니다.

다음은 Style 적용이 가능한 대상들입니다.

종류	적용 대상
컴포넌트 (Visible Object)	Button, Calendar, CheckBox, Combo, Div, Edit, Grid, GroupBox, ImageViewer ListBox, MaskEdit, Menu, Progressbar, Radio, Spin, Static, Tab, TextArea
화면 Frame	Form, MainFrame, ChildFrame, FrameSet, HFrameSet, VFrameSet, TileFrameSet TitleBarControl, StatusBarControl
기타	ScrollBar

## 7.2 스타일의 적용

스타일의 개요에 대해 앞에서 간단하게 언급했습니다.

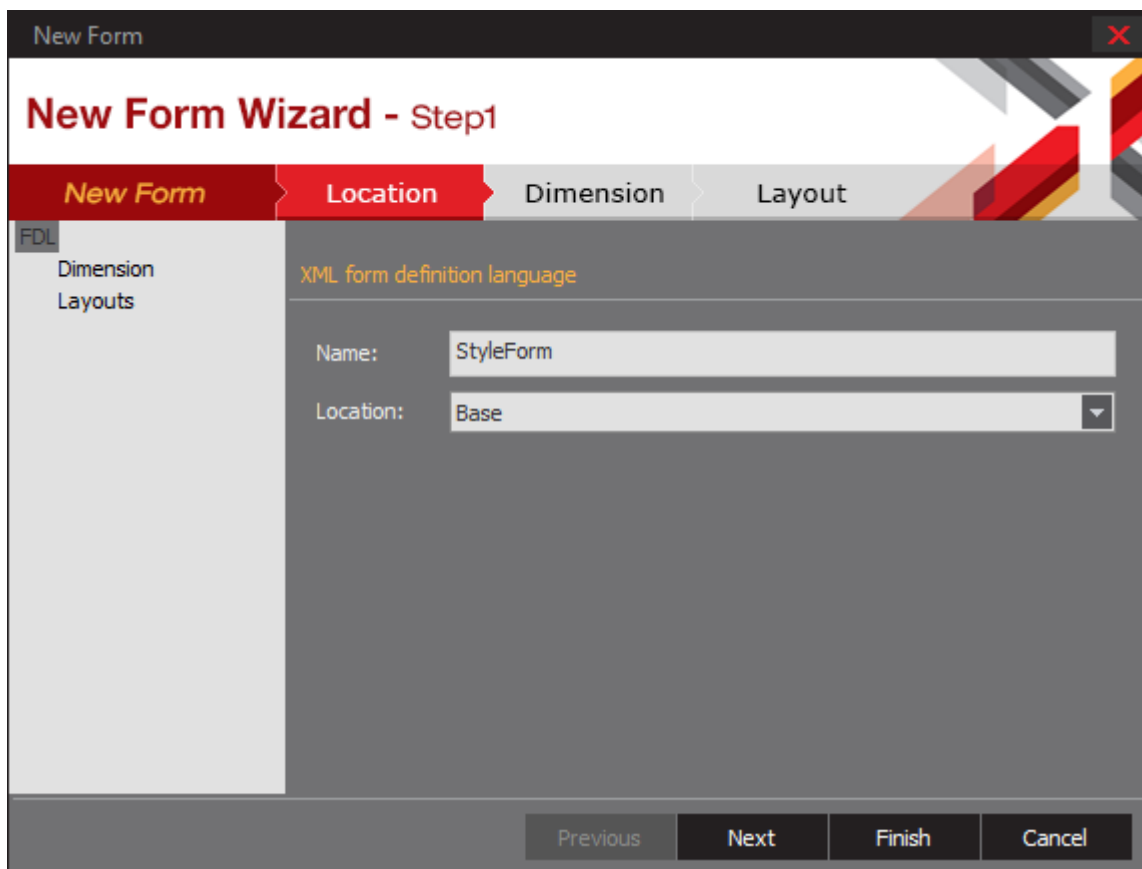
이제부터 스타일의 작성방법 및 적용, 활용하는 방법에 대하여 설명합니다.

### 스타일의 생성

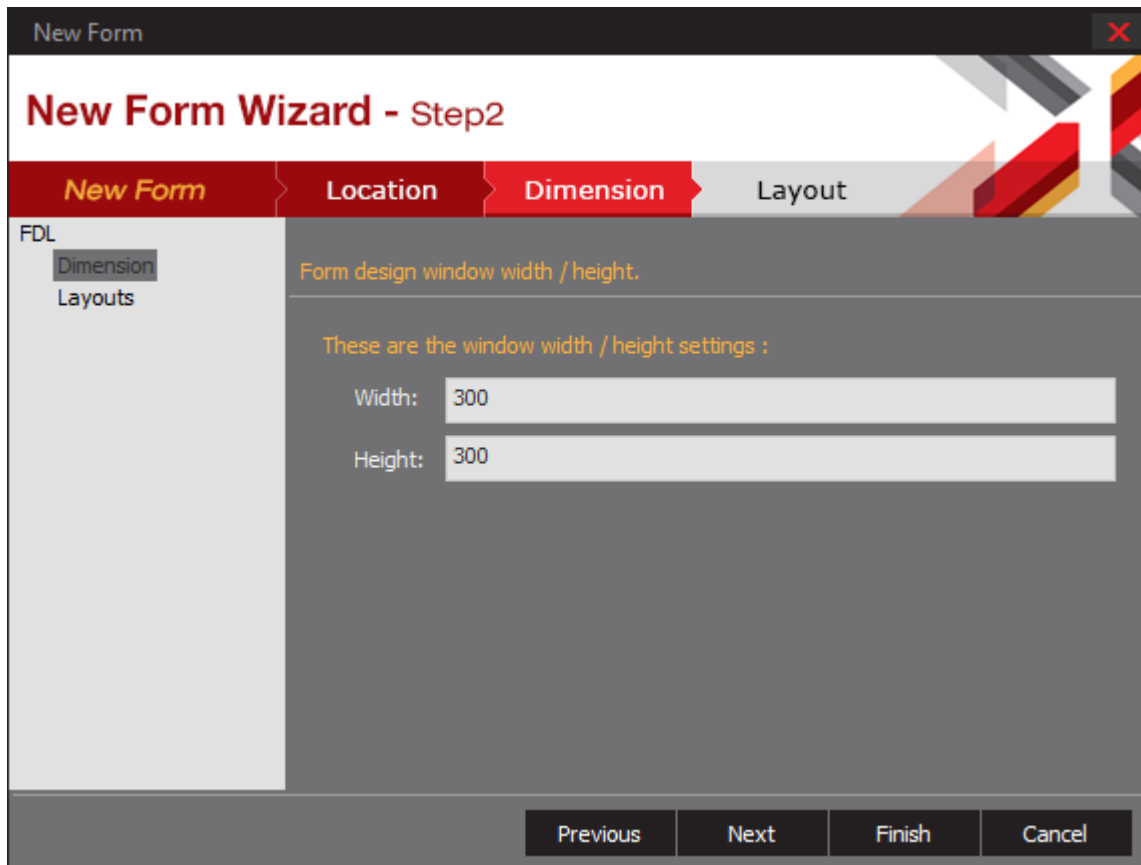
ADL에서 테마를 활용하지 않을 경우나 Form에 직접 스타일을 지정하고 싶을 때 새로 생성하거나 기존에 작성된 스타일 파일을 등록합니다.

1단계: 새로운 Form을 만듭니다.

New Form 메뉴에서 Form을 만들고 Form명을 StyleForm으로 지정합니다.

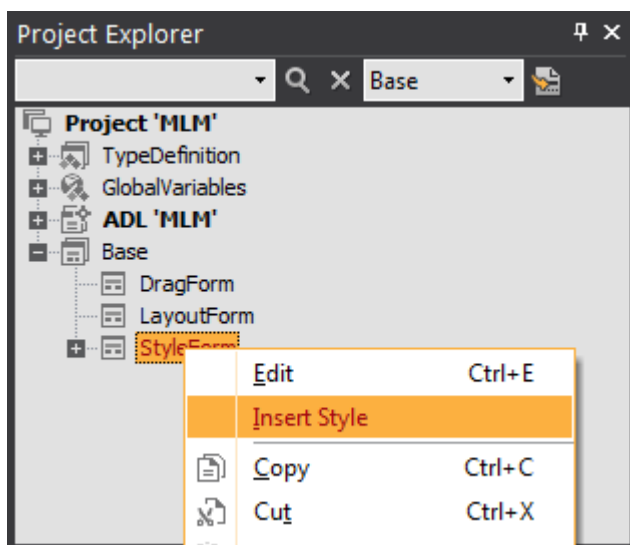


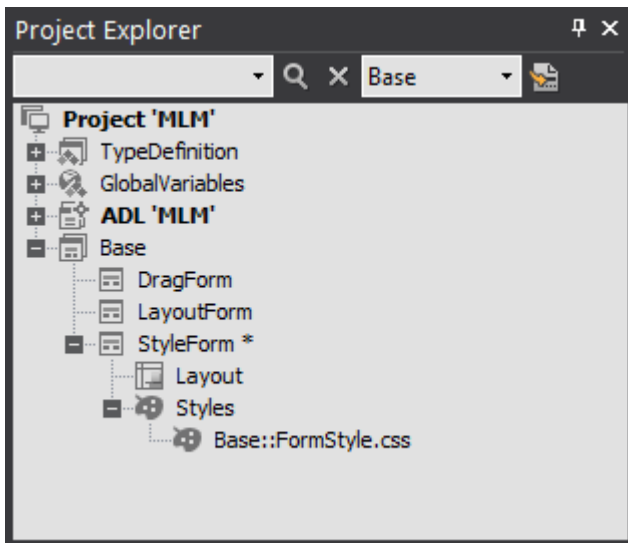
The image shows a 'New Form Wizard - Step1' dialog box. The title bar is 'New Form'. The main title is 'New Form Wizard - Step1'. The wizard has four steps: 'New Form', 'Location', 'Dimension', and 'Layout'. The 'New Form' step is currently selected and highlighted in red. On the left side, there is a tree view with 'FDL' selected, and 'Dimension' and 'Layouts' are listed below it. The main area is titled 'XML form definition language'. It contains two input fields: 'Name:' with the value 'StyleForm' and 'Location:' with a dropdown menu showing 'Base'. At the bottom, there are four buttons: 'Previous', 'Next', 'Finish', and 'Cancel'.



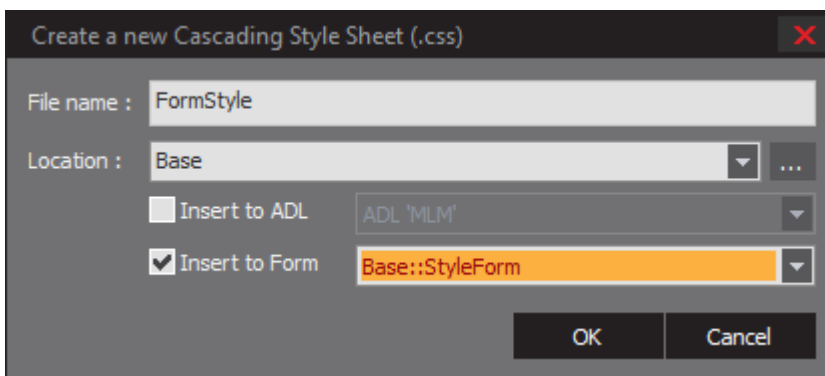
2단계: 만들어진 Form에 새로운 스타일 파일을 생성합니다.

만들어진 StyleForm 컨텍스트 메뉴에서 Insert Style 항목을 선택하고 새로운 스타일 파일을 FormStyle.css 라는 이름으로 추가합니다. Form이 열려있으면 스타일 파일을 추가할 수 없습니다.



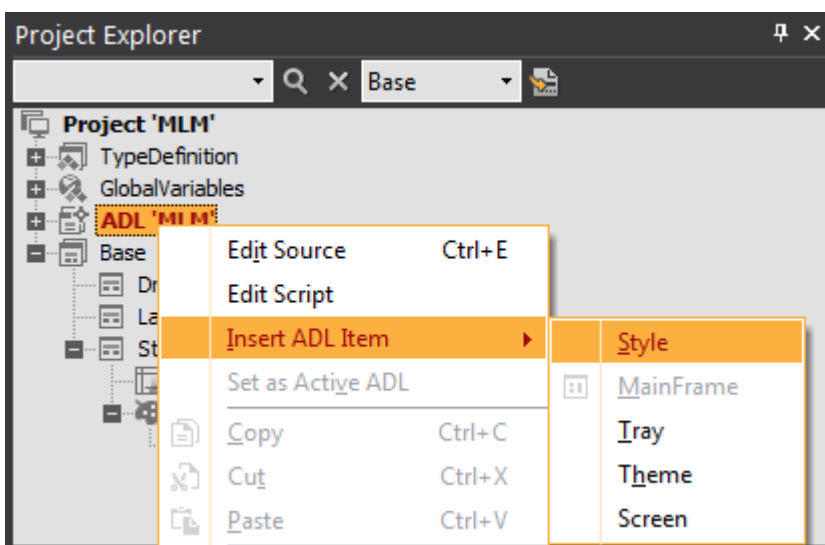


메인 메뉴(File > New > File > Style Sheet)에서 새로운 CSS를 만들어 Form에 포함하고자 한다면 'Insert to Form' 항목을 체크하고 추가하기 원하는 Form 이름을 지정해주어야 합니다.



3단계: ADL에 스타일 파일을 생성합니다.

ADL에 Insert ADL Item으로 하나의 스타일을 생성하고 이름은 ADLStyle.css로 지정해 추가합니다.





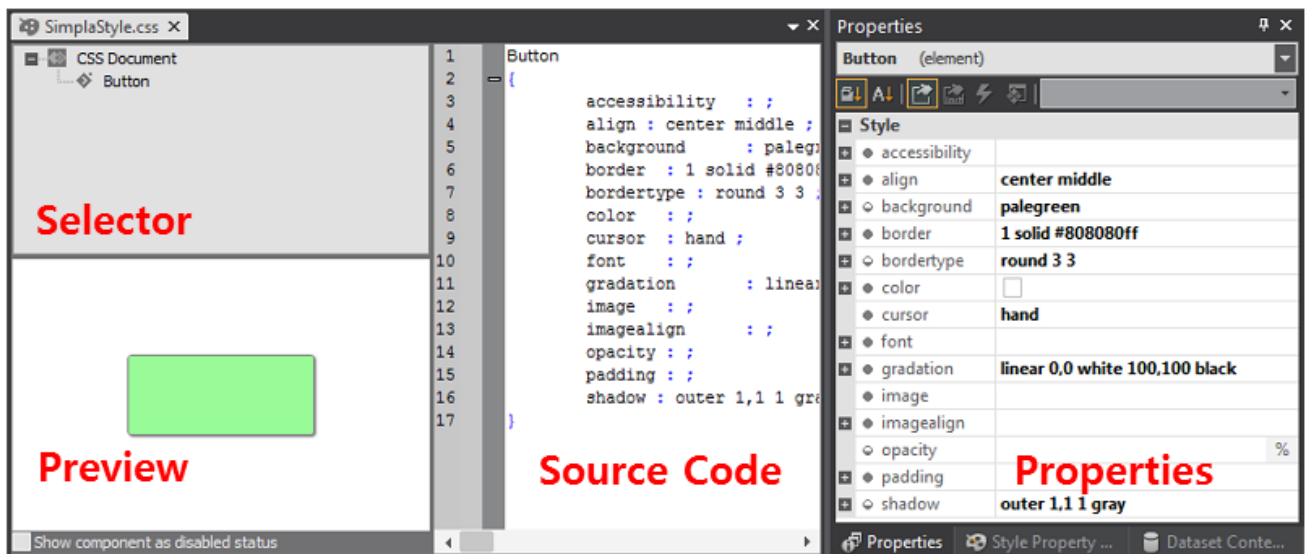
ADL에 추가된 스타일은 프로젝트를 Refresh 해주어야 적용이 됩니다. 스타일을 추가하고 프로젝트 컨텍스트 메뉴에서 Refresh를 선택해주세요.

## 스타일의 편집

스타일의 편집방법은 Theme, ADL, Form이 모두 같습니다.

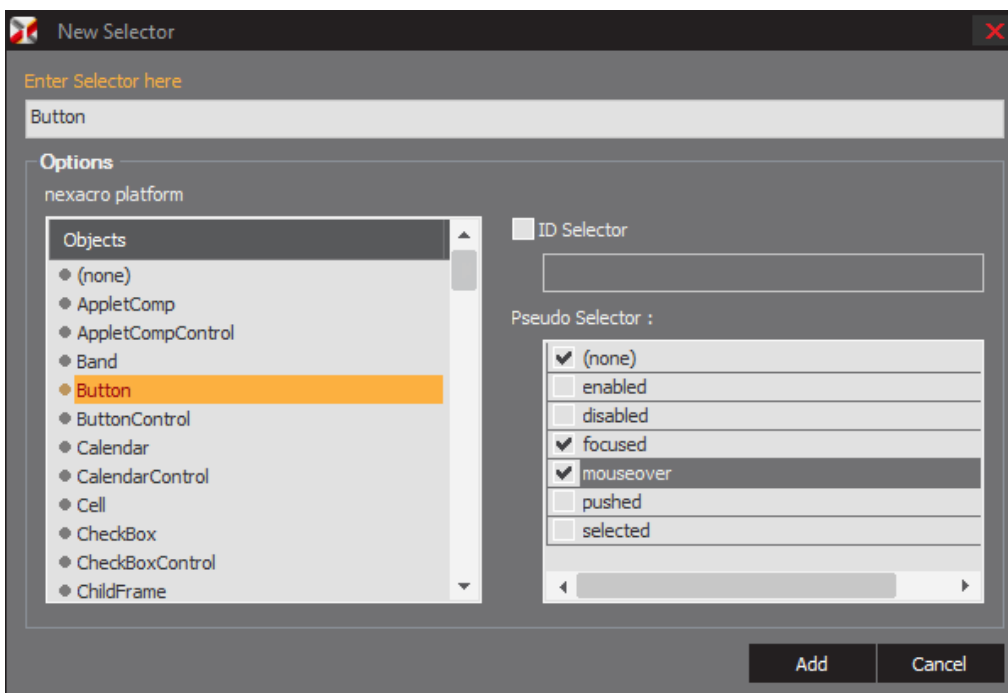
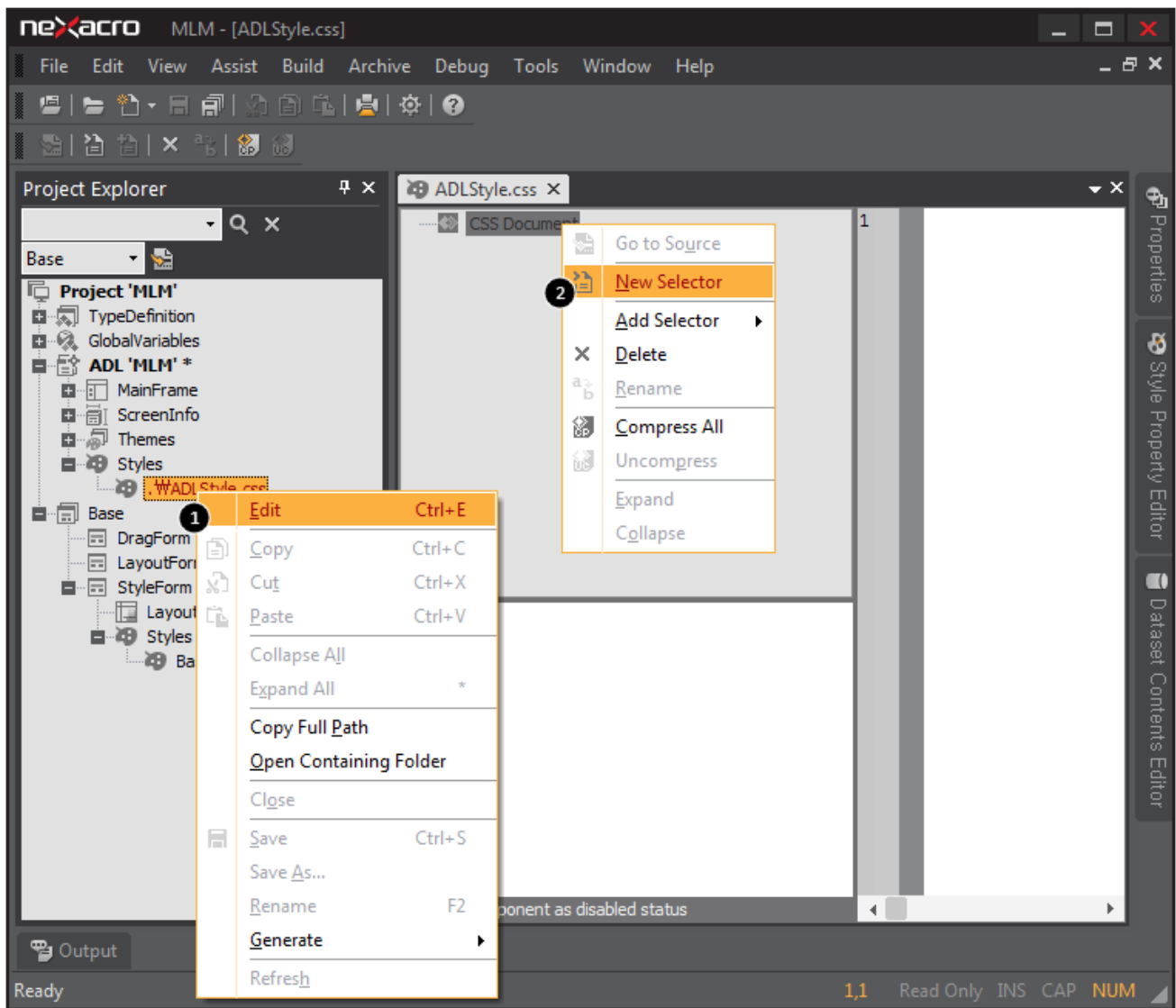
스타일의 편집은 별도로 제공되는 스타일 편집 창에서 할 수 있습니다. 스타일 편집 창은 선택자 목록을 트리 형태로 제공해 추가/삭제가 가능하며, 소스코드 창에서 직접 내용을 수정할 수 있습니다. 또한 속성 창에서 쉽게 값을 입력하거나 수정할 수 있습니다.

속성 창에 입력된 내용은 소스코드 창에 바로 반영되어 확인할 수 있습니다.



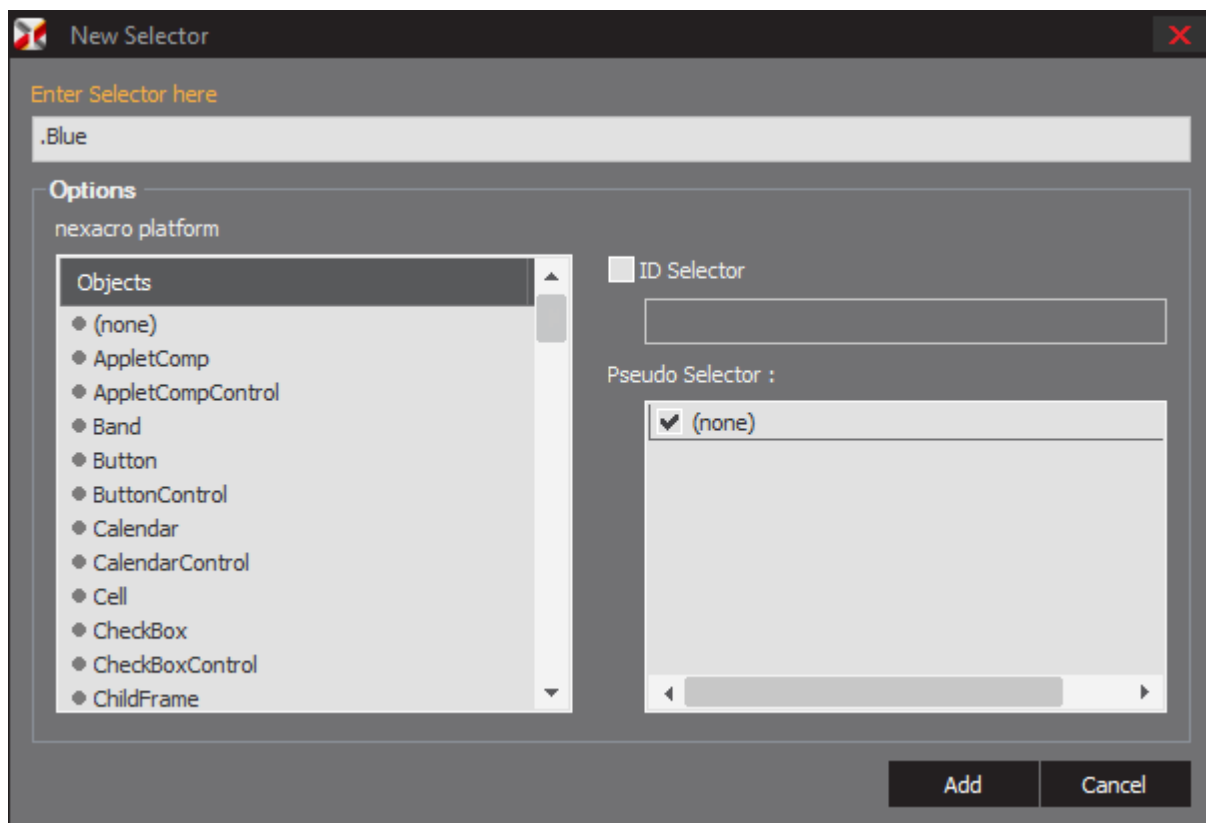
1단계: 스타일 편집 창을 통해 새로운 선택자를 등록합니다.

생성한 ADLStyle.css 파일에 Button 컴포넌트 선택자(Type Selector, Pseudo Selector, Class Selector)를 등록합니다. 먼저 Type Selector와 Pseudo Selector(mouseover, focused)를 등록합니다.





Class Selector를 등록합니다.



생성된 소스 코드 형식은 아래와 같습니다.

```
Button
{
...
}

Button:focusd
{
...
}

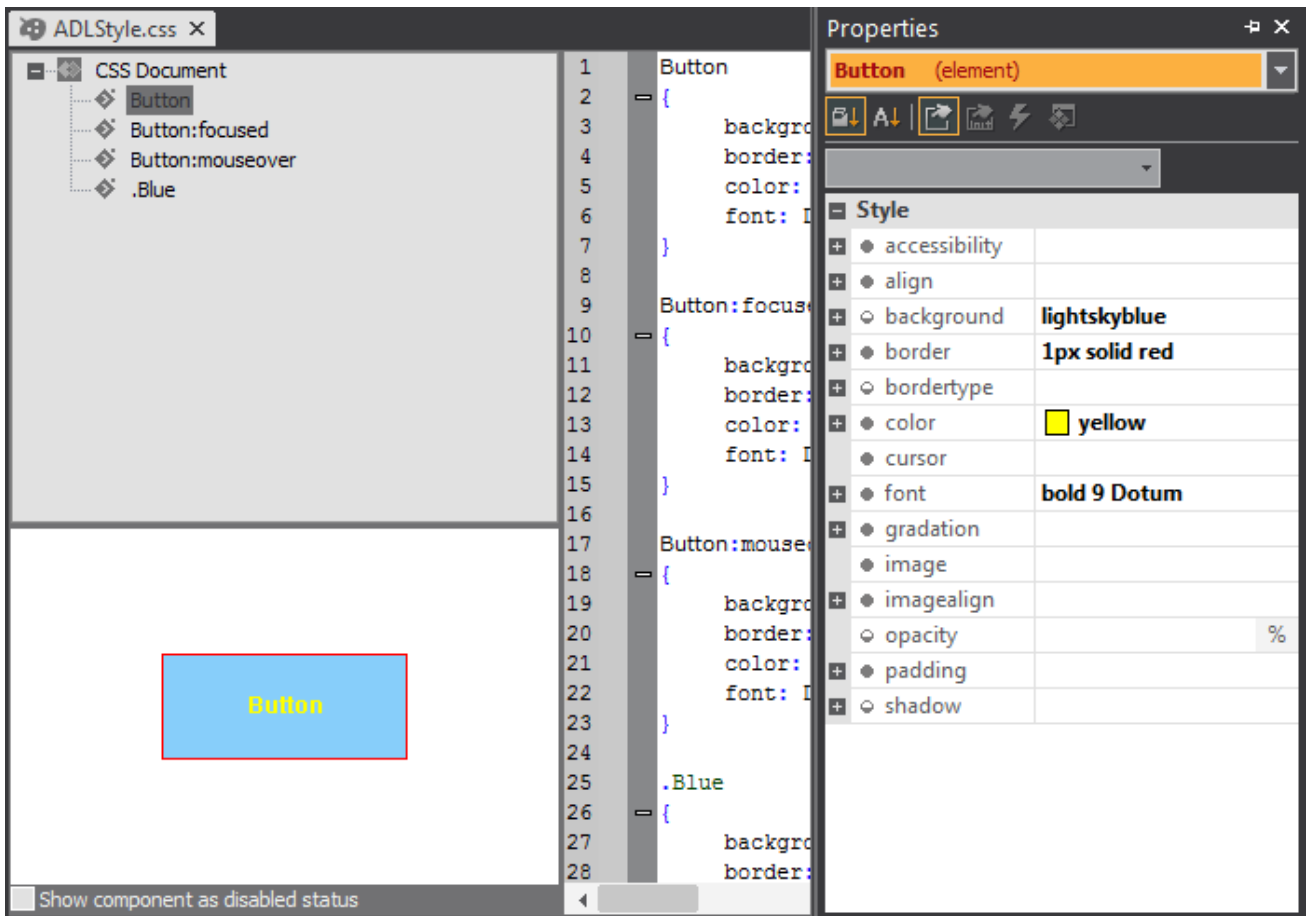
Button:mouseover
{
...
}

.Blue
{
```

```
}
```

2단계: 스타일 편집창에 등록된 선택자의 내용을 작성합니다.

생성한 Button 선택자(Type Selector, Pseudo Selector, Class Selector)의 정보를 작성합니다.



편집할 선택자를 선택하고 소스 코드를 직접 편집하거나 속성 창에서 해당하는 항목을 수정할 수 있습니다. 소스 코드는 아래와 같이 수정합니다.

```

Button
{
    background: lightblue;
    border: 1px solid red;
    color: yellow;
    font: Dotum,9,bold ;
}

Button:focus
{
    background: lightblue;
    border: 1px solid yellow;
  
```

```
    color: black;
    font: Dotum,9,underline;
}

Button:mouseover
{
    background: lightskyblue;
    border: 1px solid black;
    color: red;
    font: Dotum,9;
}

.Blue
{
    background: lime;
    border: 1px solid blue;
    color: blue;
    font:Dotum,10,bold;
}
```

## 스타일의 적용

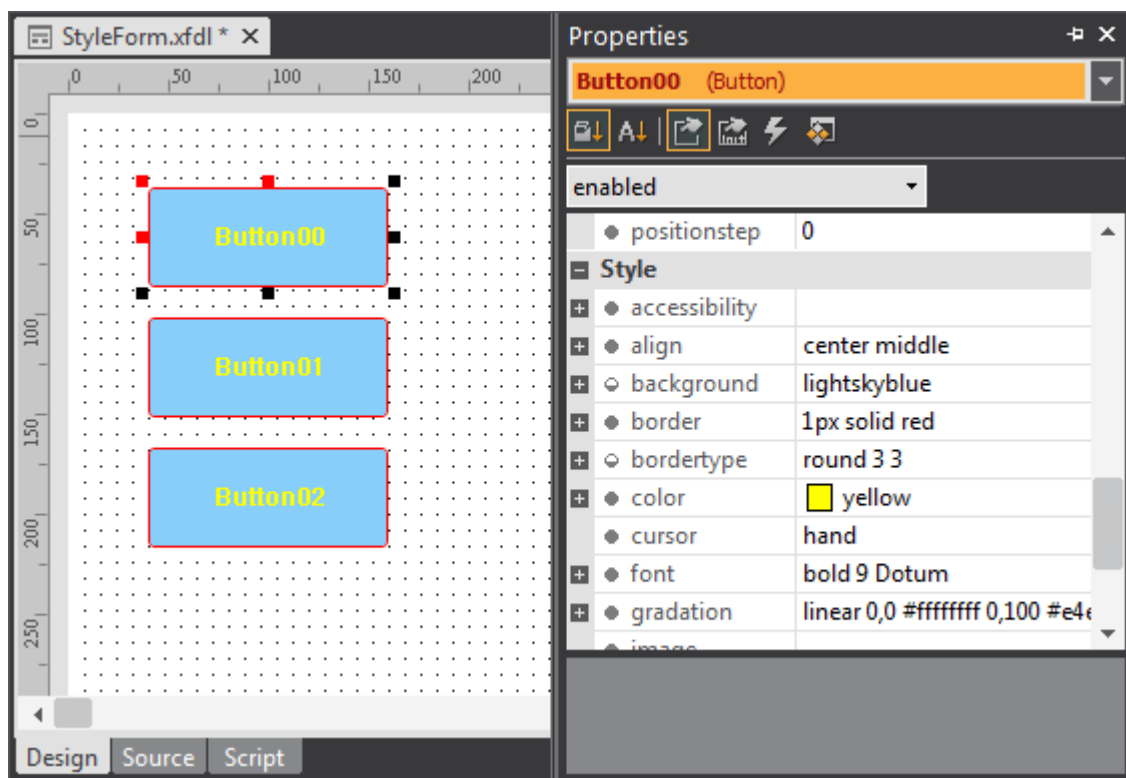
Form에 Button을 생성해 ADL에 있는 스타일과 Form에 있는 스타일에 따라 화면 디자인에 반영되도록 합니다.

### 1단계: 새로운 Form을 만듭니다.

New Form으로 Form을 만들고 이름은 StyleForm으로 지정합니다 ("스타일의 생성" 단계에서 만든 Form을 활용합니다.)

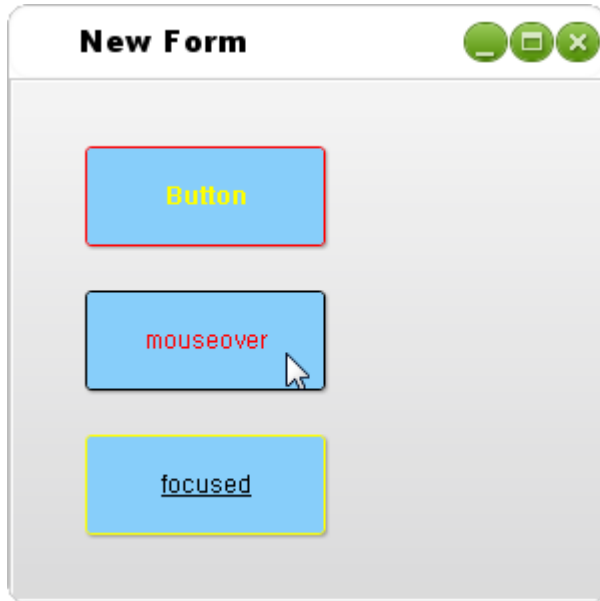
### 2단계: Button을 생성해 스타일의 적용을 확인합니다.

StyleForm에 Button을 생성해 ADL에 작성된 스타일에 맞추어 디자인이 적용되는지 확인합니다.



3단계: Form을 실행해 Button의 Pseudo Selector의 적용을 확인합니다.

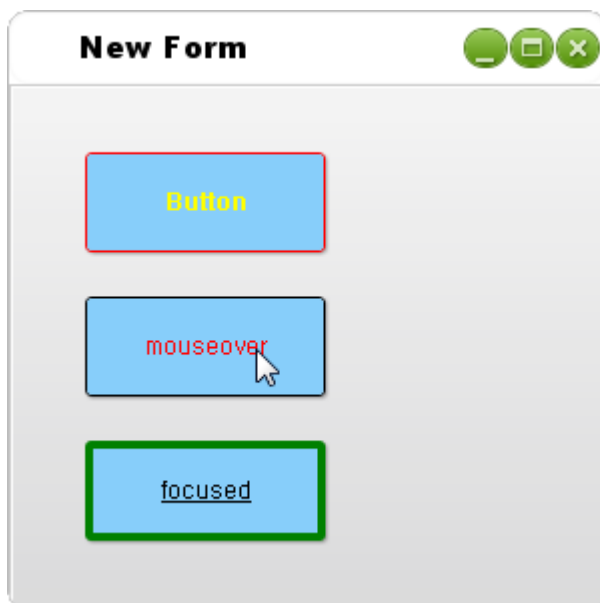
ADL 스타일에 설정한 Pseudo Selector(mouseover, focused)가 Form에 적용되는 것을 QuickView를 실행해서 확인합니다.



**4단계: 앞서 Form에 생성한 Form 스타일을 편집하고, 적용을 확인합니다..**

Form 스타일에 작성된 Pseudo Selector 중 focused 정보를 ADL 스타일과 다르게 설정해 다르게 적용되는 것을 QuickView로 실행해서 확인합니다.

```
Button:focusd
{
    background: lightskyblue;
    border: 4px solid green;
    color: black;
    font: Dotum,9,underline;
}
```



## cssclass의 적용

같은 Form에서 같은 컴포넌트에 서로 다른 Style을 적용하기 위해 등록된 Class 선택자를 사용해 표현을 다르게 할 수 있습니다.

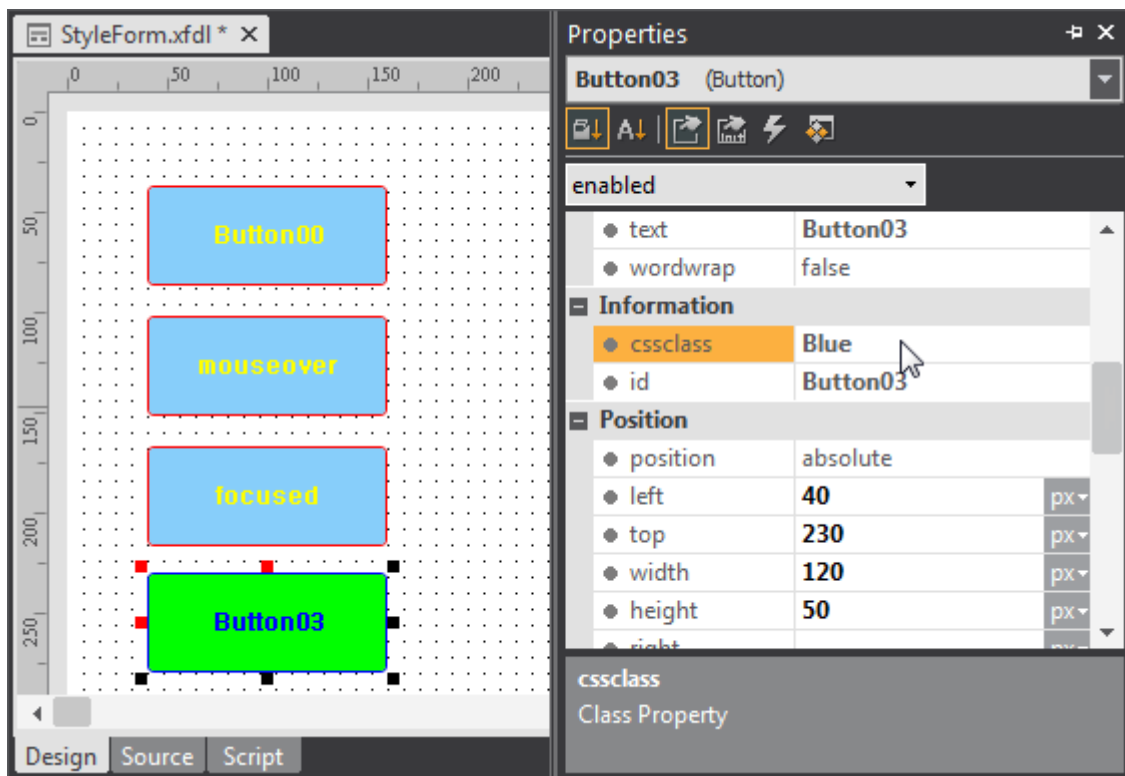
**1단계: ADLStyle에 생성한 Class 선택자를 편집하고, 적용을 확인합니다..**

ADLStyle에 Class Selector(Blue)를 편집합니다.

```
.Blue
{
    background: lime;
    border: 1px solid blue;
    color: blue;
    font:Dotum,10,bold;
}
```

**2단계: Form의 Button에 cssclass 속성을 활용해 Style의 적용을 확인합니다.**

Form의 cssclass 속성에 ADLStyle에 작성된 Class 선택자인 Blue를 설정하고, 스타일의 적용을 확인합니다.



## 7.3 테마의 적용

테마의 개요에 대해 앞에서 간단하게 언급했습니다.

이제부터 테마의 작성방법 및 적용, 활용하는 방법에 대하여 설명합니다.

### 테마의 생성

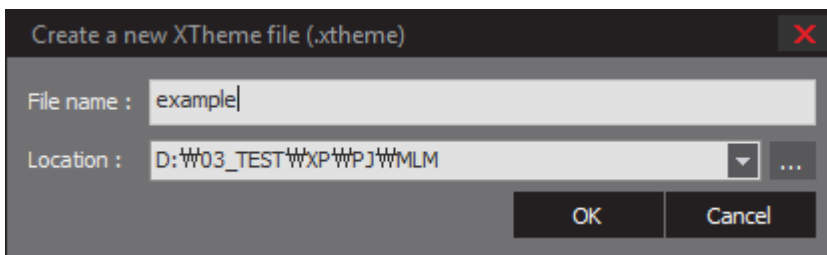
테마의 생성방법은 Insert Theme(Insert ADL Item)을 통해서 미리 작성된 테마를 선택해 등록할 수 있습니다.

테마 편집 창에서는 이미지나 스타일 파일을 추가, 삭제할 수 있으며 스타일 파일은 스타일 편집 창에서 관리할 수 있습니다.

스타일 파일이 변경되는 경우에는 반드시 테마도 저장해 주어야 합니다.

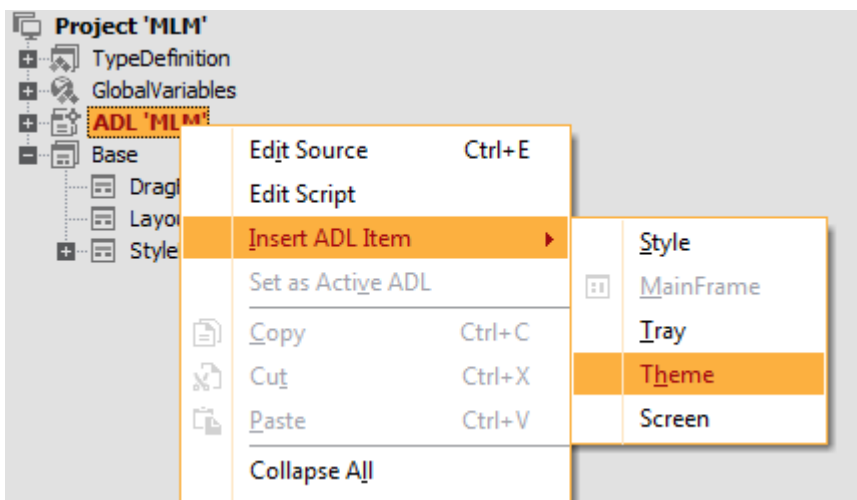
#### 1단계: 새로운 테마를 만듭니다.

메뉴(File > New > File > XTheme)에서 example이라는 이름으로 새로운 테마를 만듭니다. 새로 만들어진 테마는 default 테마를 기본으로 복사됩니다.



#### 2단계: 미리 작성된 테마를 지정합니다.

ADL에 Insert ADL Item 메뉴에서 테마를 지정해 추가해줍니다.





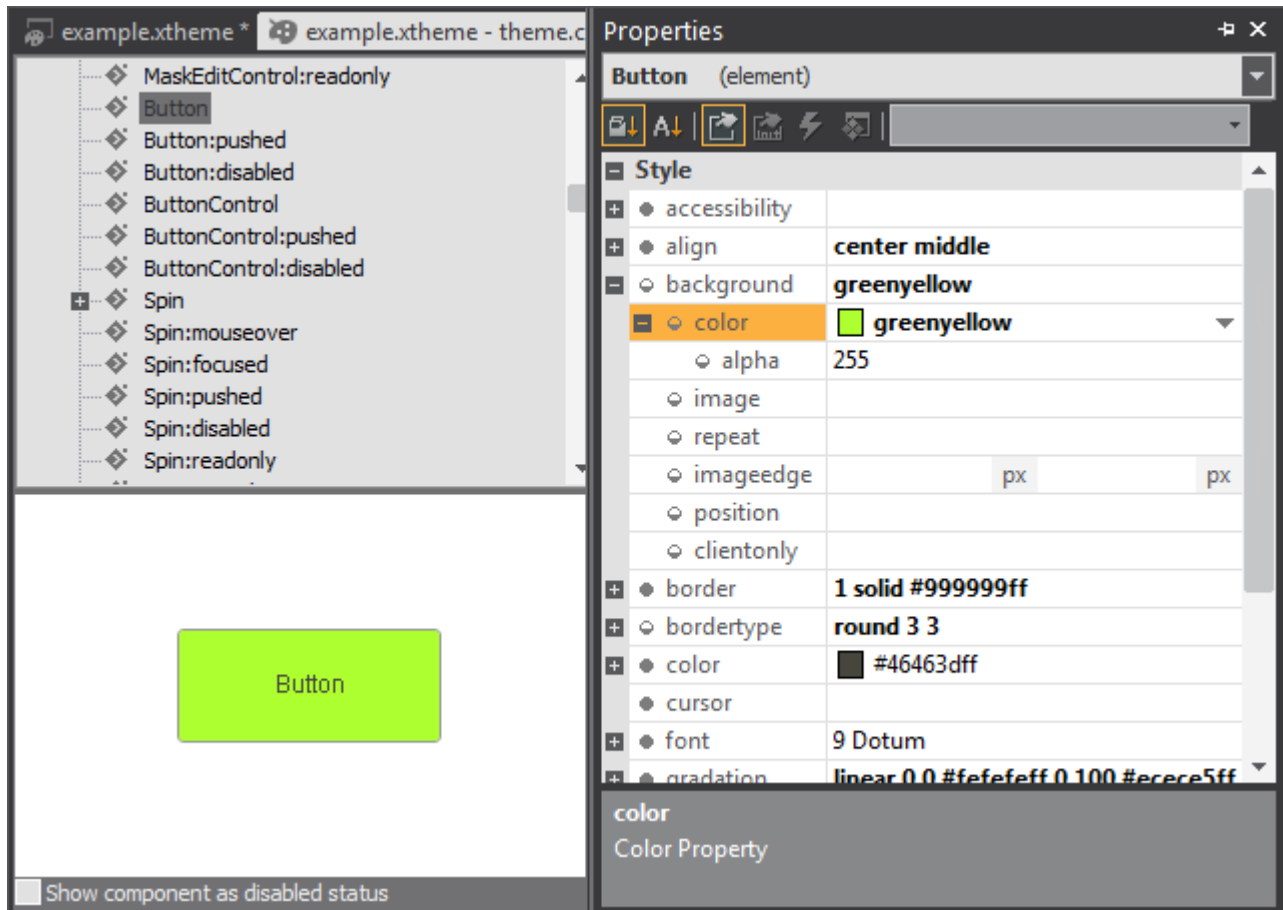
## 테마의 편집

테마 파일은 이미지와 스타일 파일이 서로 분리되어 있어 별도로 편집합니다.

폴더의 생성, 스타일의 생성, 이미지, 스타일 파일 추가/삭제는 테마 편집 창 하단의 버튼을 활용합니다.

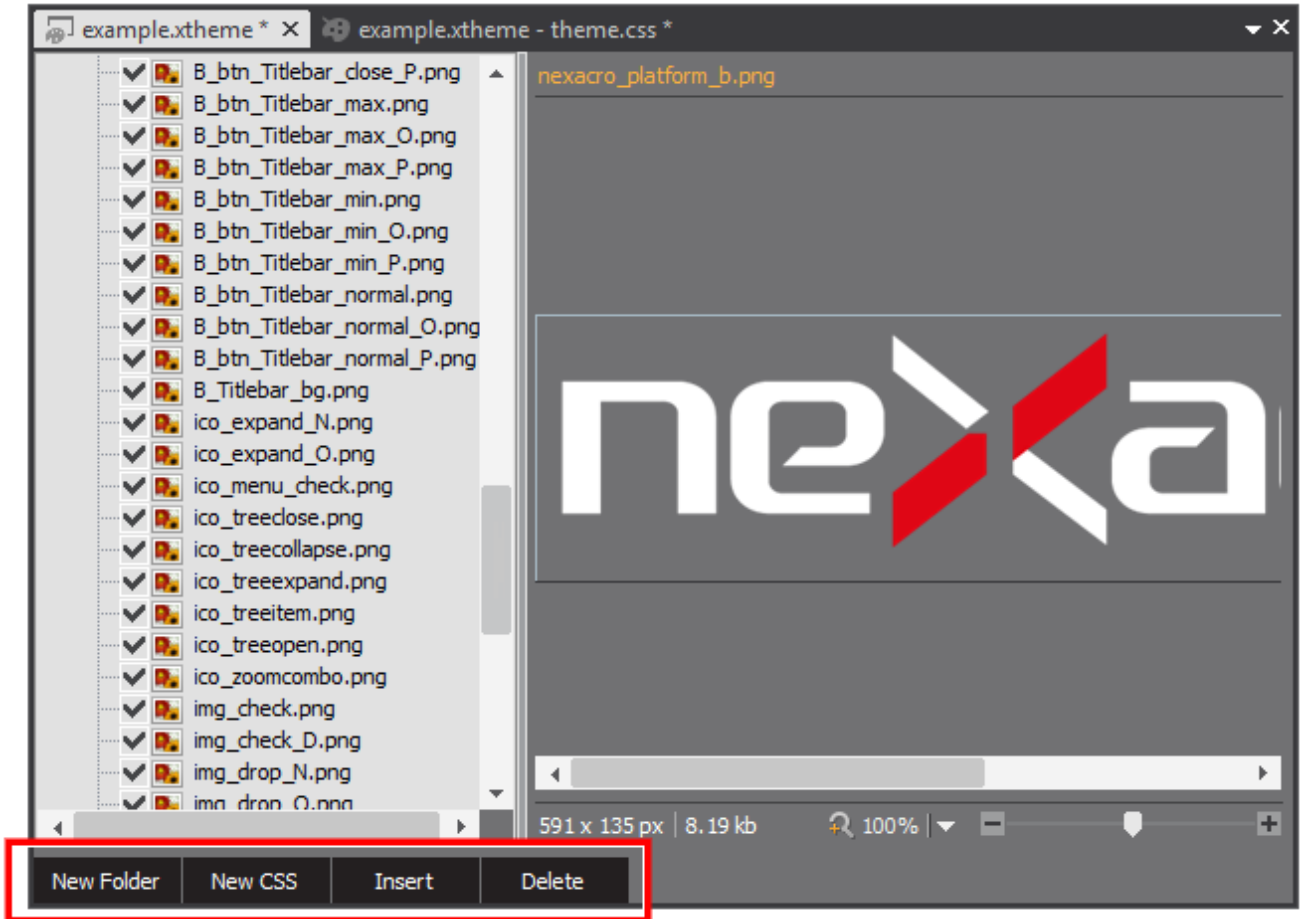
**1단계: Theme Treeview의 내용을 확인하고 스타일 파일을 편집합니다.**

테마 파일인 example.xtheme를 열어 Theme Treeview를 통해 이미지 목록과 스타일 파일(theme.css)을 확인하고, Style 편집 창에서 Style 파일을 편집합니다.



2단계: 테마 편집 창에서 이미지, 폴더, 스타일 파일의 생성/추가/삭제 작업을 합니다.

Theme Treeview 하단의 버튼들을 통해 테마를 편집합니다.



메뉴	기능
New Folder	테마 내에 새로운 폴더를 생성
New CSS	테마 내에 새로운 스타일 파일을 생성
Insert	이미지나 스타일 파일 추가
Delete	Treeview에서 선택된 파일 삭제

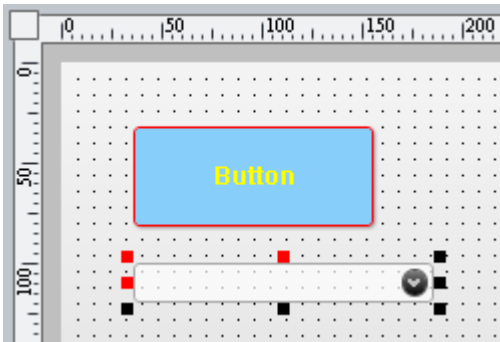
## 테마의 적용

기본으로 제공되는 default Theme의 이미지와 스타일이 반영된 디자인이 활성화된 테마를 변경할 때 컴포넌트에 반영되는 것을 확인할 수 있습니다.

작성은 Combo 컴포넌트를 기준으로 작성하도록 하겠습니다.

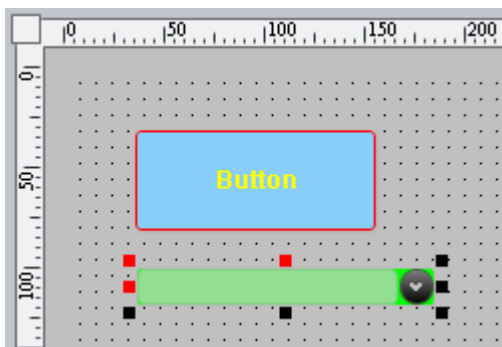
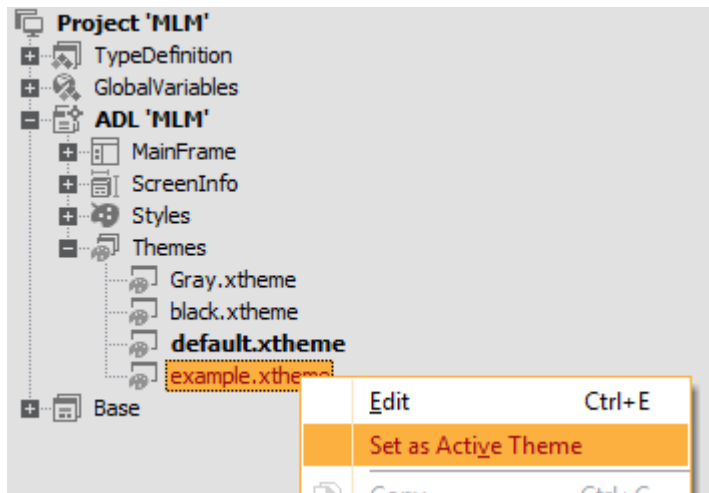
**1단계: default Theme의 Combo 컴포넌트의 스타일을 확인합니다.**

StyleForm을 열어 변경되기 전의 Combo를 생성해 스타일을 확인합니다.



**2단계: 활성화된 테마를 example Theme로 변경해 변화를 확인합니다.**

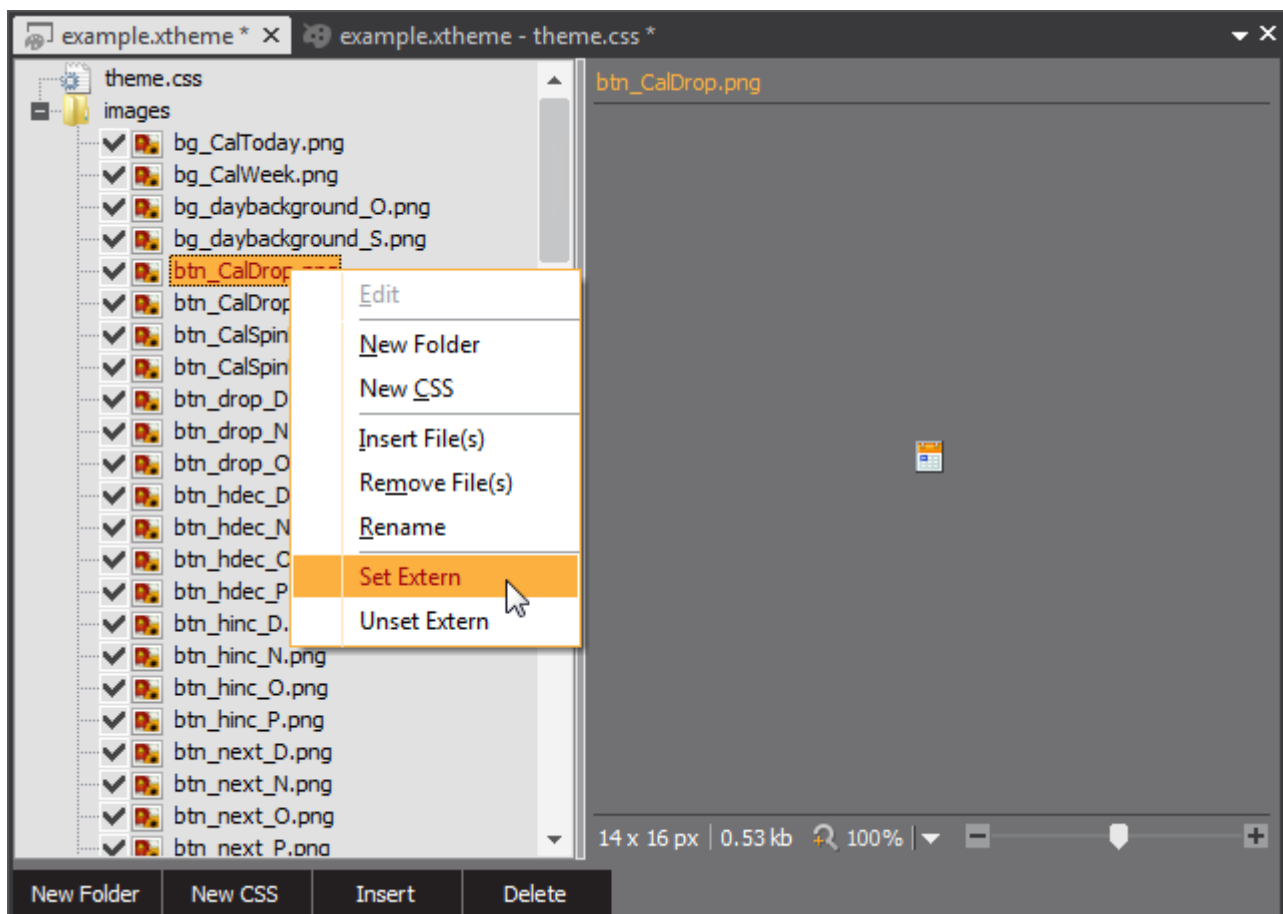
새로 생성한 example Theme의 컨텍스트 메뉴에서 Set Active Theme를 선택해 해당 테마를 활성화하고, StyleForm의 스타일의 변화를 확인합니다.



## Deploy Theme

테마는 이미지와 스타일이 결합되어 있어 그 크기가 크다는 단점이 있습니다. Deploy Theme 기능은 xtheme 파일에서 이미지를 분리해 테마의 크기를 줄이기 위해 만든 기능입니다. 이미지는 선택적으로 분리할 수 있습니다. 이렇게 분리된 xtheme 파일을 “Deploy Theme”라 하고 분리된 이미지는 “Extern File”이라 합니다.

- Extern File로 분리할 이미지 선택
  - 체크박스를 이용해 Extern File로 분리할 이미지를 선택할 수 있습니다.
    - 체크박스를 선택하지 않은 이미지만 extern file로 분리되며 체크된 이미지는 테마에 포함됩니다. 선택하지 않은 이미지 각각을 Extern Item이라 합니다.
    - default.xtheme와 같이 기본적으로 제공되는 Theme는 체크박스가 나타나지 않습니다. 즉, 기본으로 제공되는 테마는 이미지를 분리할 수 없습니다.
    - 스페이스바를 이용해서 체크박스를 선택할 수도 있습니다. 다중 선택도 가능하며 선택한 첫 번째 아이템의 상태를 기준으로 변경됩니다.
    - 마우스 우측 버튼을 클릭 후 컨텍스트 메뉴에서 체크박스를 선택할 수도 있습니다. Set Extern 메뉴를 선택하면 체크박스가 해제되고 Unset Extern 메뉴를 선택하면 체크가 됩니다.



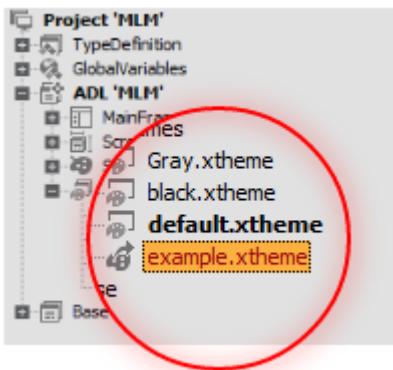
- 저장

Deploy Theme의 저장은 기존 파일 저장 방식처럼 저장 버튼을 누르면 됩니다. 저장 시 Deploy Theme와 Extern File의 저장내용 및 경로는 다음과 같습니다.

구분	저장내용	저장경로
Deploy Theme	Extern File을 제외한 테마	기존 theme경로 및 파일명과 동일 예) 기존 theme를 C:\W\theme\Wa.xtheme라 하면 C:\W\theme\Wa.xtheme로 저장됨
Extern File	분리된 이미지	기존 theme경로/images/분리한 Image들(파일명동일) 예) 기존 theme를 C:\W\theme\Wa.xtheme라 하고 분리할 Image가 b.png라 하면 C:\W\theme\images\Wa\Wb.png로 저장됨

- Deploy Theme표시

Extern Item이 있는 상태로 저장을 하면 프로젝트 탐색 창에 Deploy Theme 표시가 생기며 Extern Item을 제거하고 저장을 하면 Deploy Theme 표시도 사라집니다.



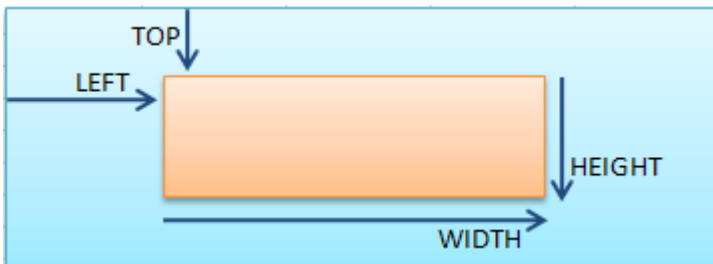
## 8.

# Position

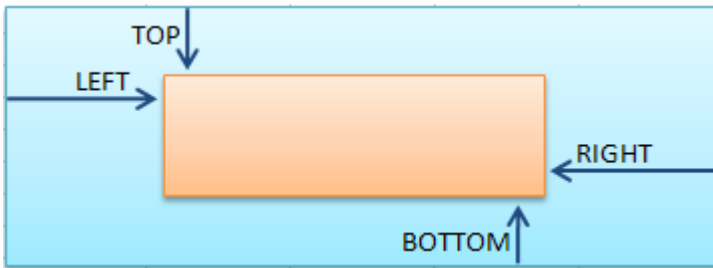
넥사크로플랫폼에서는 폼이나 각 컴포넌트의 위치를 left, top, right, bottom width, height 속성값을 지정해 지정할 수 있습니다. 각 속성값은 픽셀 단위(px) 또는 퍼센트(%) 중 하나를 선택해서 사용할 수 있습니다. 넥사크로 스튜디오에서는 포지션 에디터, 눈금자, DotGrid(화면 배경으로 레이아웃 배치를 위해 보이는 점), 가이드 라인 등 레이아웃 배치를 지원하는 기능을 제공합니다.

## 8.1 좌표계 설명

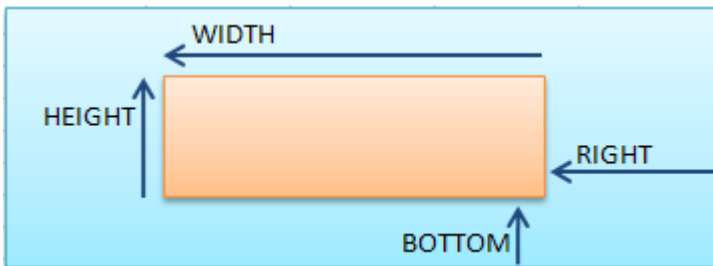
넥사크로플랫폼에서 사용하는 좌표계는 아래 그림과 같이 left, right, width, top, bottom, height 중 속성값 중 4개 속성값을 선택해 사용할 수 있습니다.



```
<Button id="Button00" text="Button00" width="120" height="50" left="20" top="20"/>
```



```
<Button id="Button00" text="Button00" left="20" top="20" right="260" bottom="230"/>
```



```
<Button id="Button00" text="Button00" right="260" bottom="230" width="120" height="50"/>
```



## 8.2 컴포넌트의 위치값 설정

컴포넌트의 위치를 설정할 때는 left, right, width, top, bottom, height 중 속성값 중에서 4개의 값을 선택해 사용하며 남은 위치 설정 없이 width, height 2개의 속성값만 설정합니다.

Position		
position	absolute	
left	84	px
top	61	px
width	120	px
height	50	px
right		px
bottom		px
positionstep	0	

Position		
width	1024	px
height	768	px

속성 창에서 Position 관련 속성의 단위인 픽셀(px)과 퍼센트(%)를 쉽게 변경할 수 있습니다. 속성 단위가 변경되면 해당 속성값이 자동으로 변경됩니다.

Position		
position	absolute	
left	84	px
top	61	px
width	120	px
height	50	px
right		px



폼은 퍼센트 단위를 지정할 수 없으며 픽셀로만 크기를 지정합니다.




스크립트에서 위치를 지정할 때는 아래와 같이 setter 메소드를 사용합니다. 단위를 지정할 수 있고 지정된 단위가 없으면 픽셀값으로 처리됩니다.

```
this.Button.set_left(10);
this.Button.set_left("10px");
this.Button.set_left("30%");
```

## 8.3 폼 디자인

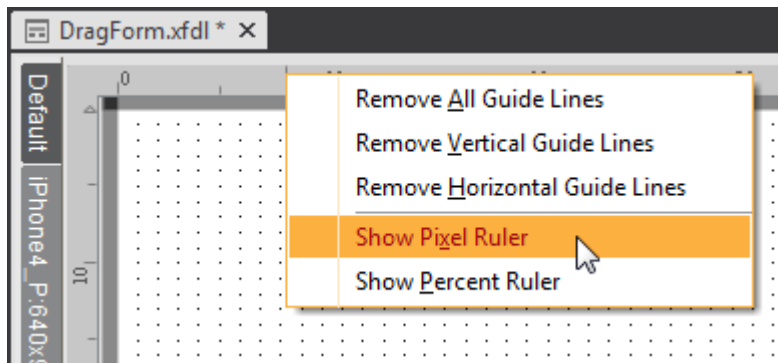
### 트래커

마우스로 컴포넌트를 선택했을 때 컴포넌트의 고정된 변에 따라 트래커가 붉은색으로 표시됩니다. 트래커를 보면 해당 컴포넌트의 속성값이 어떻게 지정되었는지 알 수 있습니다.

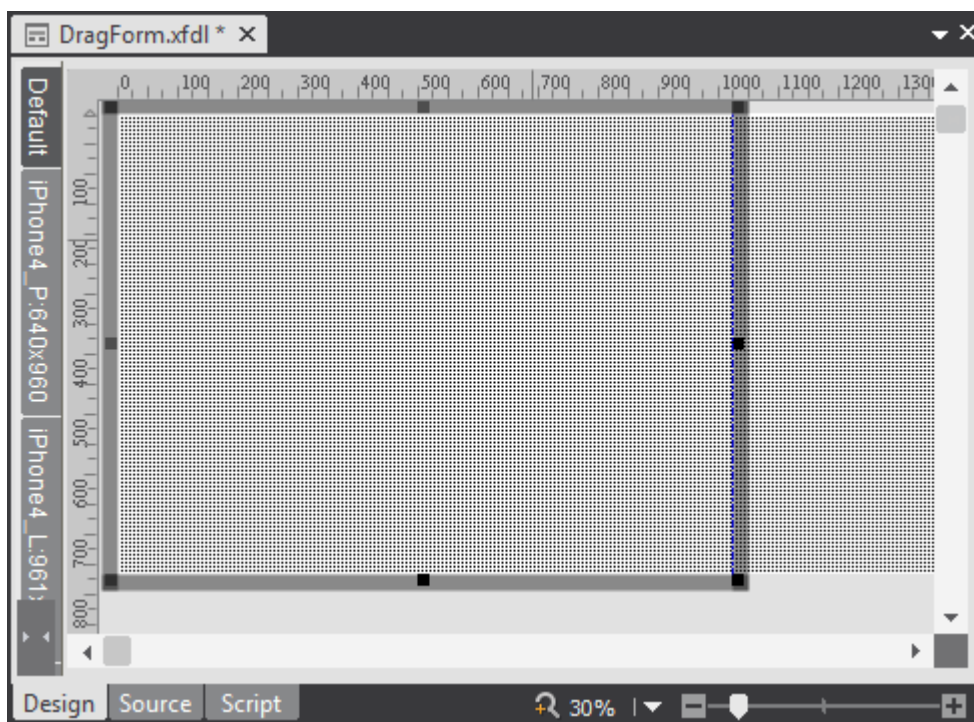
트래커	속성값
	<code>left="20" top="20" width="120" height="50"</code>
	<code>left="20" top="90" right="260" bottom="160"</code>
	<code>width="120" height="50" right="260" bottom="90"</code>

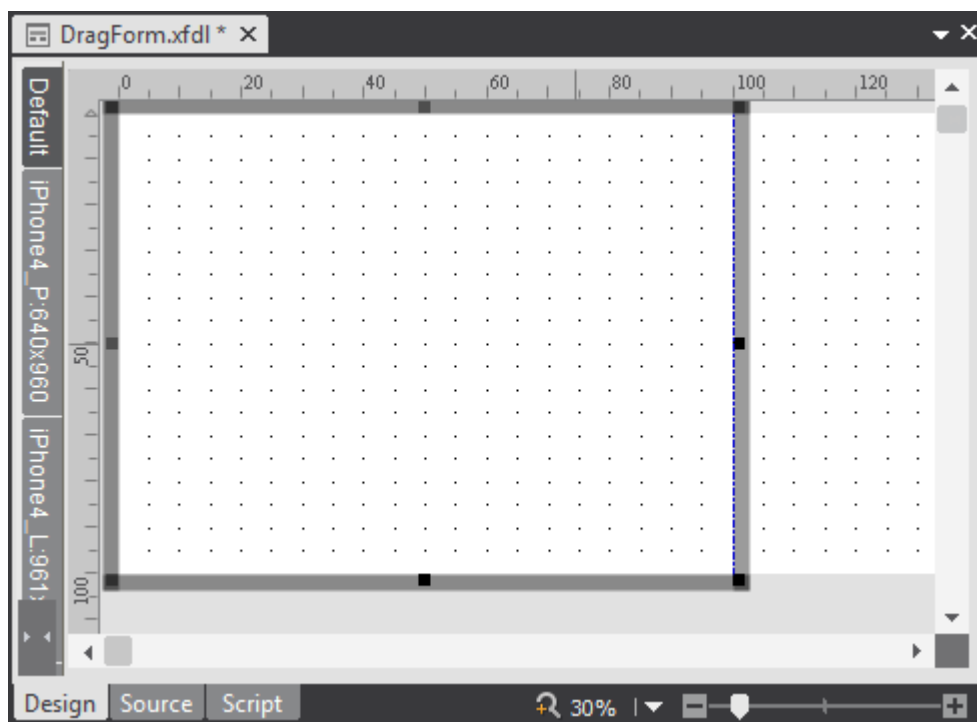
## 눈금자 / Dot Grid

폼 디자인상에서 위치를 지정하면서 사용한 단위에 따라 레이아웃을 쉽게 배치할 수 있도록 눈금자의 단위도 픽셀과 퍼센트 둘 중의 하나를 선택할 수 있습니다.



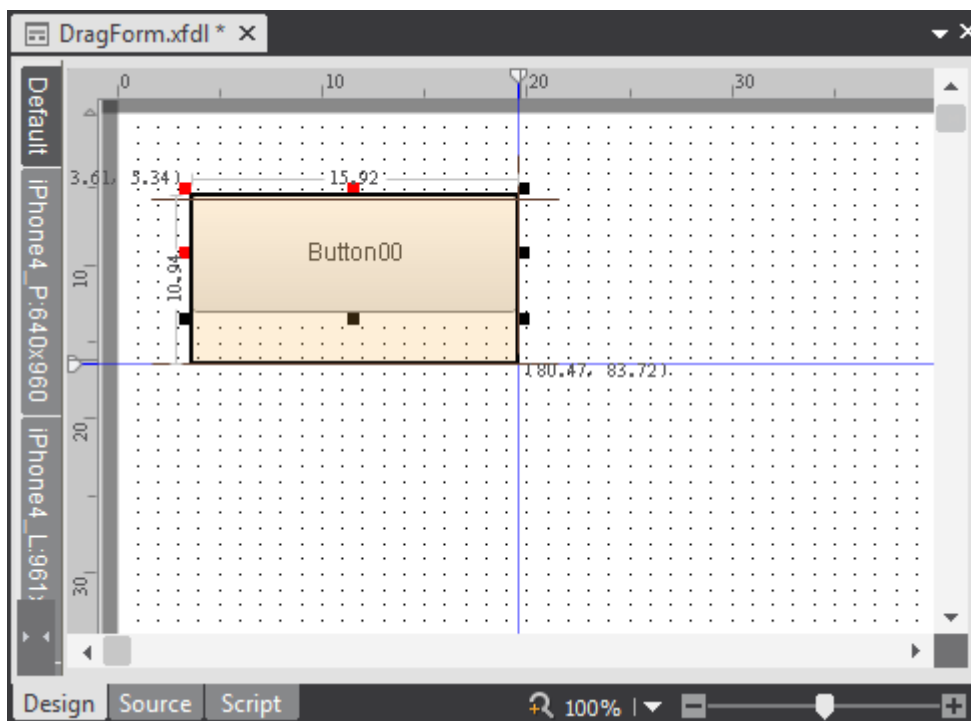
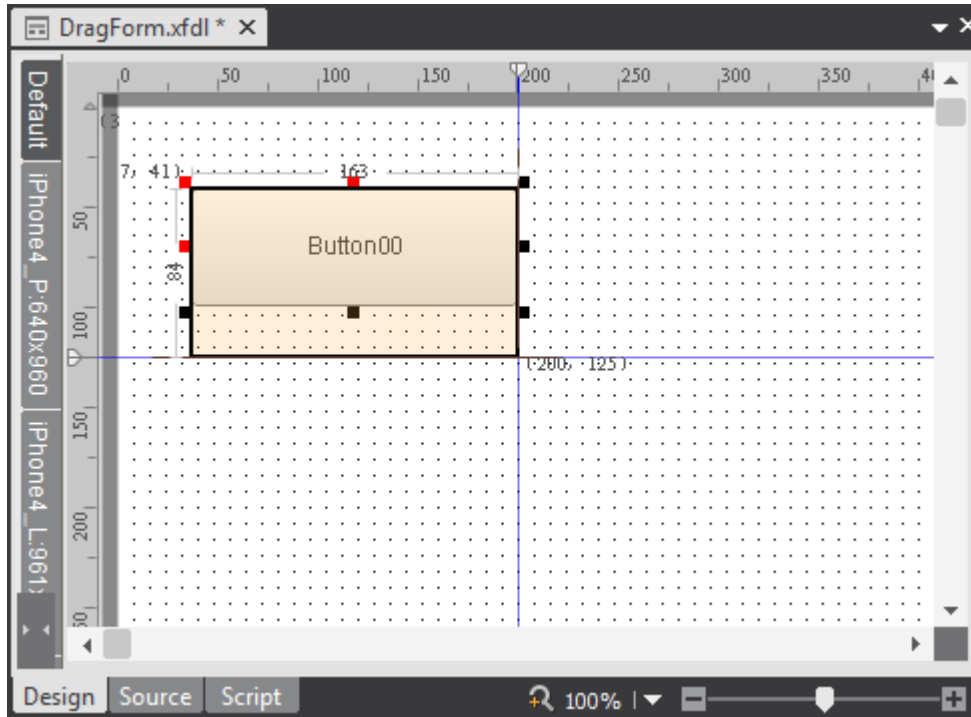
눈금자 컨텍스트 메뉴에서 Show Pixel Ruler / Show Percent Ruler를 선택하면 눈금자의 표시 단위와 Dot Grid의 표시 방법이 변경됩니다.





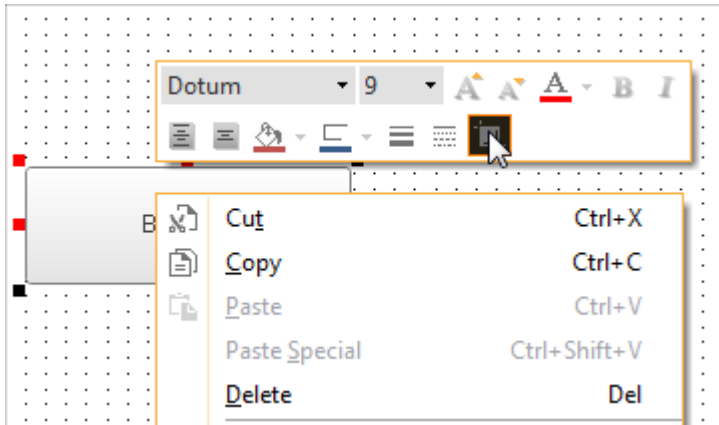
## 컴포넌트 크기 정보

컴포넌트를 하나만 선택하고 크기를 수정할 때는 크기 정보를 표시해줍니다. 이때 표기되는 크기 단위는 눈금자의 표시 단위에 따라 해당하는 단위로 표시해줍니다.

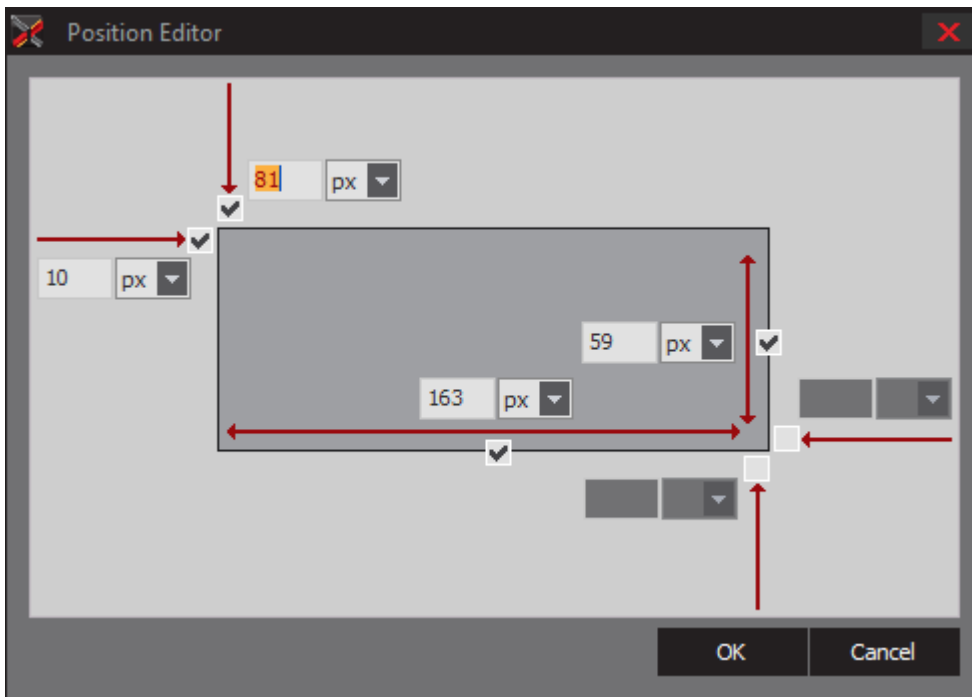


## 포지션 에디터

사용자가 쉽게 Position 정보를 설정할 수 있도록 별도의 에디터를 제공합니다. 포지션 에디터는 컴포넌트를 선택했을 때 나타나는 미니 툴바에서 실행할 수 있습니다.



미니 툴바는 자주 사용되는 스타일 속성 메뉴를 바로 수정할 수 있는 기능을 제공합니다.

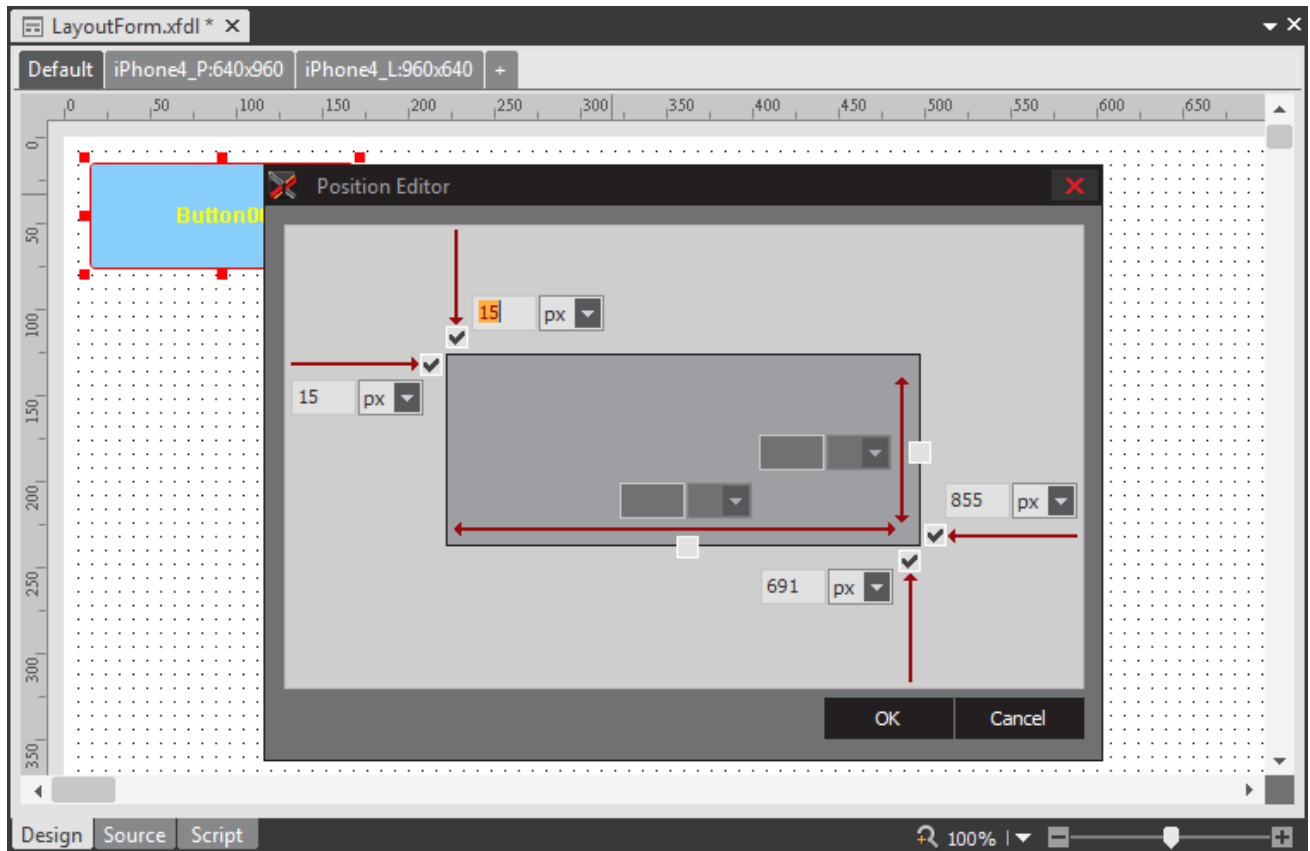


포지션 에디터에서 수정된 값은 해당 컴포넌트의 left, top, right, bottom, width, height 속성값에 반영됩니다.

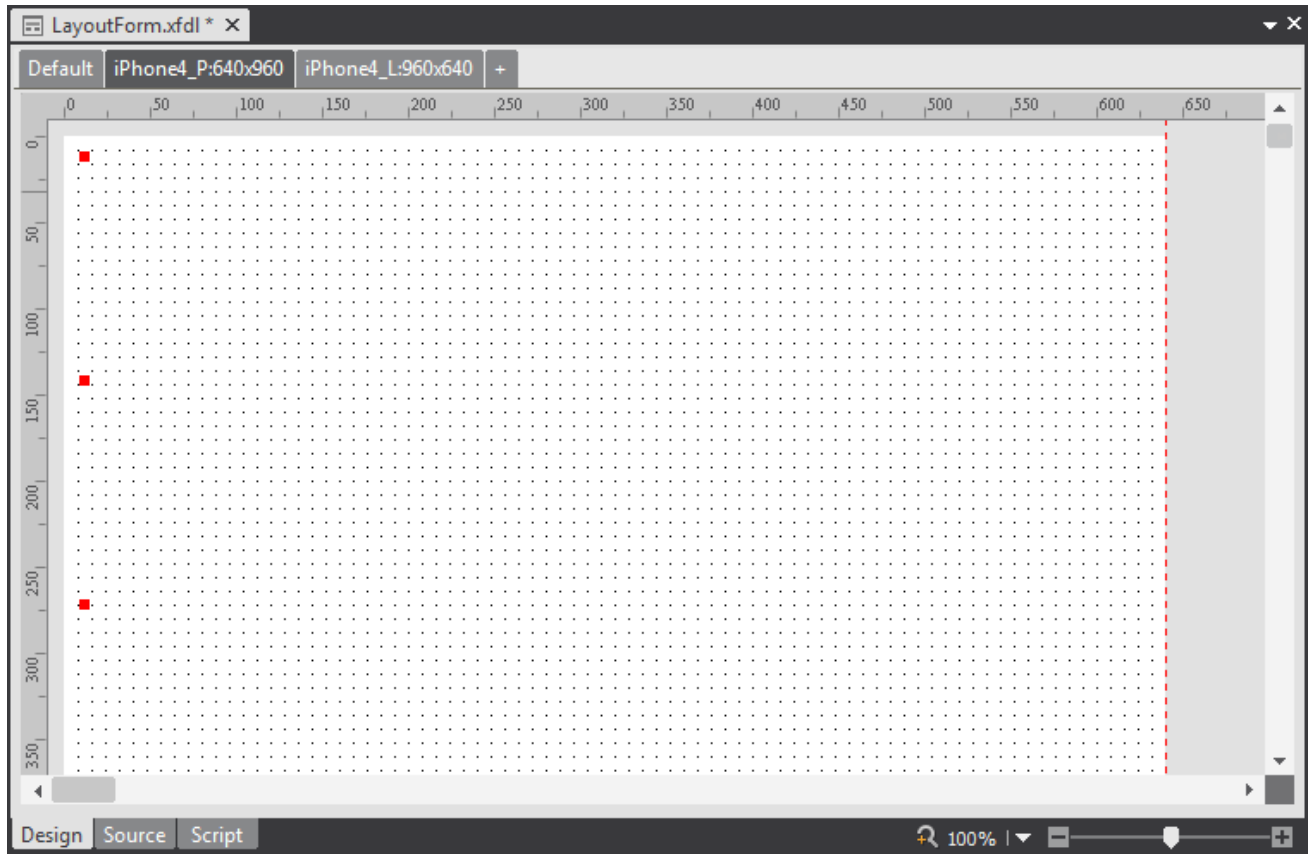
## 8.4 Position 관련 주의사항

레이아웃 매니저 기능을 사용해 추가적인 레이아웃을 사용하는 경우, left, top, right, bottom 속성만으로 좌표값을 지정하게 되면 추가된 레이아웃의 크기에 따라 컴포넌트가 보이지 않을 수 있습니다.

예를 들어, 아래와 같이 left, top, right, bottom 속성을 지정해 버튼 컴포넌트의 위치를 지정하면 Default 레이아웃에서는 정상적으로 컴포넌트가 보입니다.



하지만 추가 레이아웃에서는 right, bottom 속성에 지정된 값이 폼의 크기를 벗어나기 때문에 정상적으로 버튼이 보이지 않습니다. 컴포넌트의 left, top 위치가 고정된 상태에서 폼 보다 큰 right, bottom 속성값이 적용되기 때문에 화면에서 사라진 것처럼 보이게 됩니다.





## 9.

---

# MLM(Multi Layout Manger)

개발된 애플리케이션을 사용하는 환경이 다양해지면서 각 디바이스에 최적화된 디자인과 기능을 제공해야 하는 문제가 생기고 있습니다. 요구되는 디바이스 수대로 애플리케이션을 개발한다면 그 비용과 시간을 기존 시스템에서 감당하기 어렵습니다. 넥사크로플랫폼에서는 이런 요구에 따라 하나의 소스로 다양한 상황에 맞는 디자인과 기능을 구현할 수 있는 멀티 레이아웃 매니저 기능을 지원합니다.

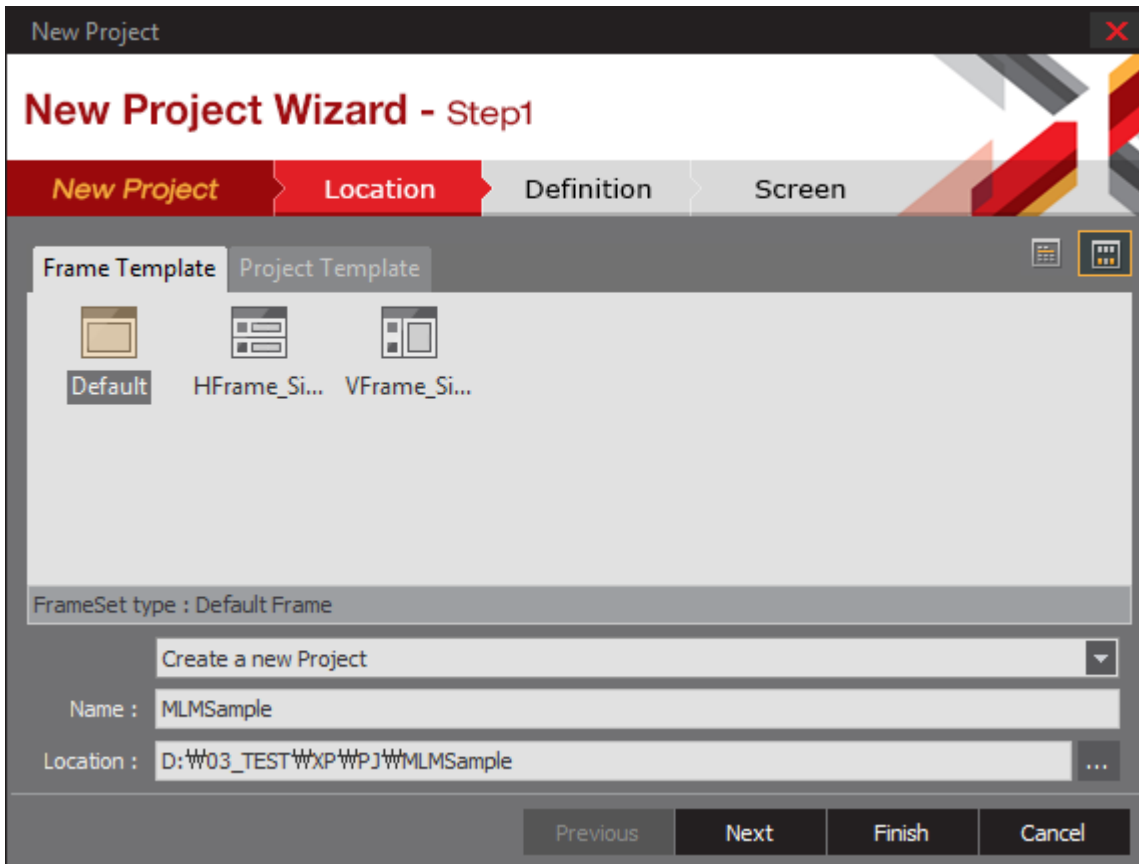
## 9.1 프로젝트 생성 및 수정

### 프로젝트 생성

'iPhone', 'iPad', 'GalaxyTab'의 레이아웃 정보를 가진 프로젝트를 생성하는 방법을 예로 들어 설명합니다. 넥사크로 스튜디오에서 File > New > Project 메뉴를 호출하면 아래와 같이 프로젝트를 생성하는 마법사가 표시되며, 단계별로 정보를 입력하여 생성할 수 있습니다.

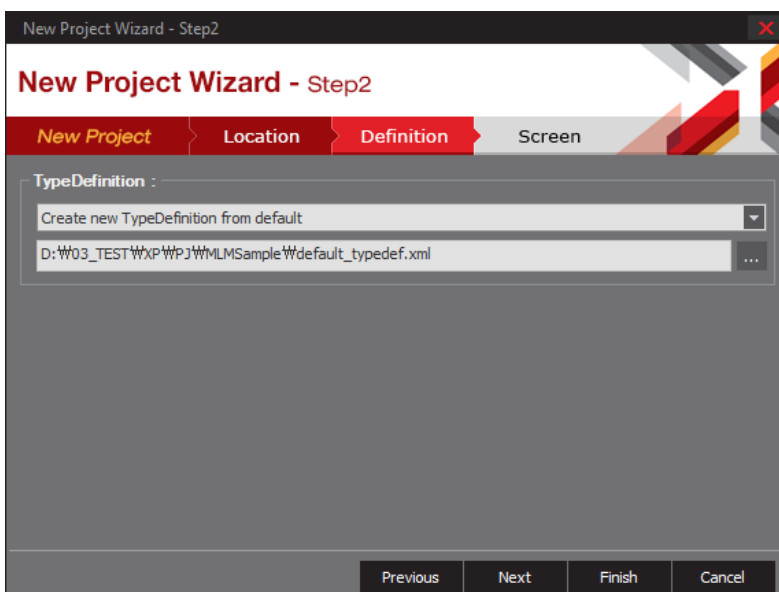
### New Project Wizard - Step1

생성될 프로젝트의 프레임 템플릿을 설정하고 프로젝트를 저장할 경로 및 이름을 입력하는 단계입니다. 프로젝트명은 반드시 입력해야 하는 필수 항목이며, 생성될 경로에 같은 프로젝트명이 존재하면 생성할 수 없습니다.



### New Project Wizard - Step2

TypeDefinition정보를 설정하는 단계입니다.



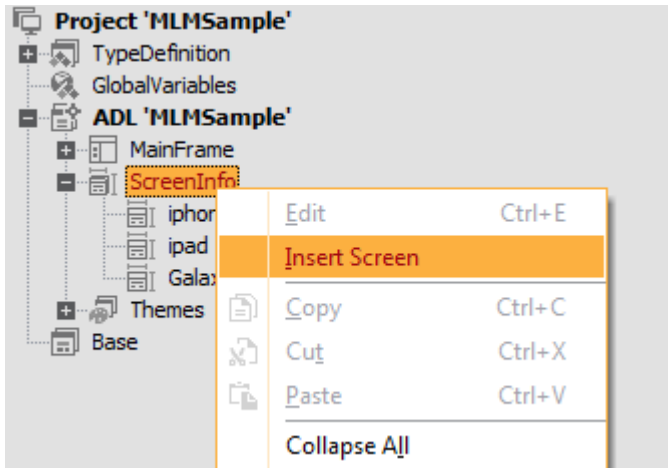
### New Project Wizard - Step3

프로젝트에서 사용할 스크린 정보를 입력하는 단계입니다. iPhone, iPad, GalaxyTab에 맞는 애플리케이션을 개발할 목적이기 때문에 각각의 항목을 입력하고, 해당 스크린에서 사용할 테마를 지정합니다. 'Finish'를 클릭 하면 'iPhone', 'iPad', 'GalaxyTab' 스크린 정보를 가진 프로젝트가 생성됩니다.

속성	설명
name	스크린 이름
type	사용 장비 형태 (desktop, phone, tablet) - 여러 항목 선택 가능
width	스크린 너비 (입력 제한: 0~65536)
orientation	스크린 방향 (landscape, portrait)
autozoom	지정된 스크린 크기에 따라 전체 화면 스케일을 확대, 축소할지 선택 (true/false)
themeid	스크린에 사용할 테마 id

## 스크린 정보의 수정

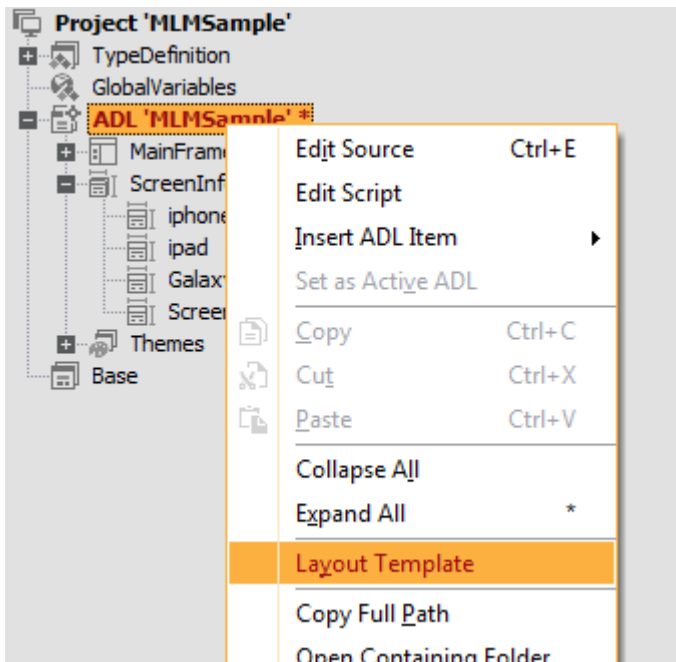
‘New Project Wizard’에서 입력한 스크린 정보를 수정하거나 새로운 스크린 정보는 Project Explorer > ADL 컨텍스트 메뉴를 사용하여 새로운 스크린 정보를 추가하거나 ADL 하위 정보로 표시되는 ScreenInfo 항목을 선택해 편집할 수 있습니다.



No	name	type	width	orientation	themeid	hithemeid	aut
1	iphone	phone			default.xtheme		
2	ipad	tablet			default.xtheme		
3	GalaxyTab	tablet			default.xtheme		
4	Screen0						

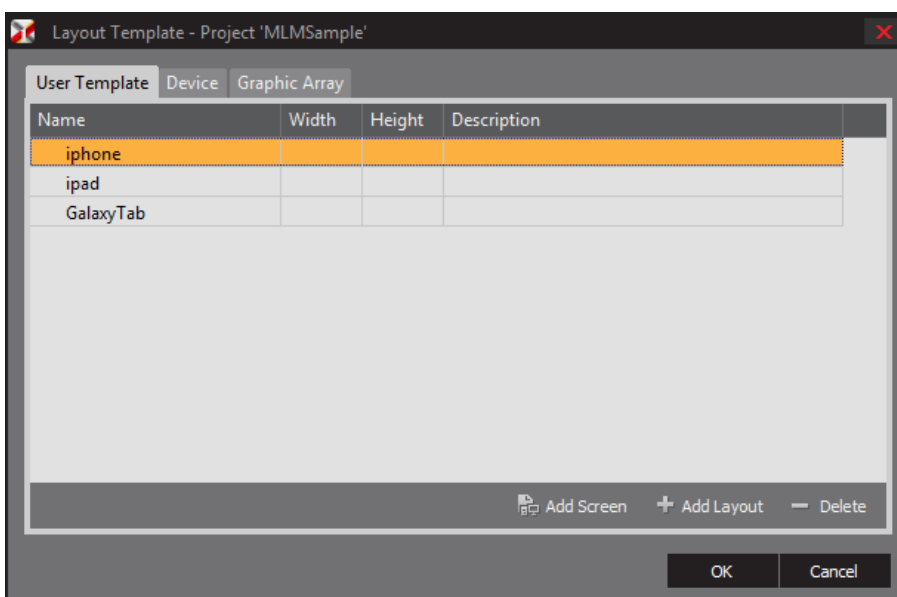
## 레이아웃 템플릿 등록

기본적으로 제공되는 레이아웃 템플릿을 사용할 수 있습니다. 또한, 자주 사용되는 레이아웃 정보를 사용자가 직접 템플릿으로 등록할 수 있습니다.



레이아웃 템플릿 대화상자에서 기본적으로 제공되는 템플릿인 Device, Graphic Array 템플릿과 사용자가 직접 등록하여 사용하는 User Template 3개의 탭으로 구성되어 있습니다.

- User Template : 사용자가 직접 등록하여 사용하는 레이아웃 템플릿 목록
- Device : 현재 사용되고 있는 주요 모바일 기기 목록
- Graphic Array : 해상도로 사용되고 있는 Graphic Array 목록



## 9.2 폼 생성

‘iPhone’, ‘iPad’, ‘GalaxyTab’ 3가지 디바이스에서 사용할 폼을 만드는 방법을 예로 들어 설명합니다. File > New > Item > Form 메뉴를 선택하면 아래와 같이 FDL을 생성하는 마법사가 실행되며, 단계별로 정보를 입력하여 생성할 수 있습니다.

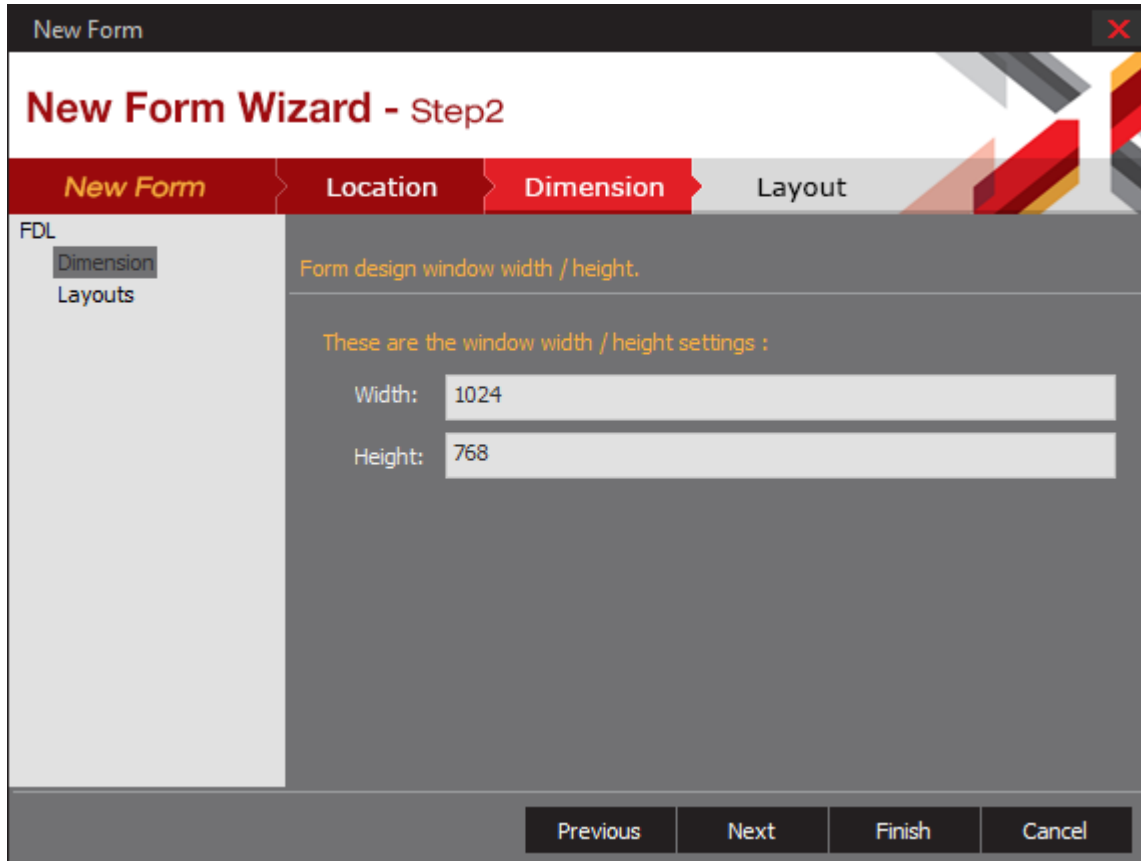
### New Form Wizard - Step1

생성될 폼의 경로와 이름을 입력하는 단계입니다. 폼의 이름은 반드시 입력해야 하는 필수 항목이며, 생성될 경로에 같은 이름이 존재하면 생성할 수 없습니다.

The screenshot shows the 'New Form Wizard - Step1' dialog box. The title bar is 'New Form'. The main title is 'New Form Wizard - Step1'. Below the title is a progress bar with four steps: 'New Form' (selected), 'Location', 'Dimension', and 'Layout'. On the left, there is a sidebar with 'FDL' (selected), 'Dimension', and 'Layouts'. The main area is titled 'XML form definition language' and contains two input fields: 'Name:' with the value 'LayoutForm' and 'Location:' with a dropdown menu showing 'Base'. At the bottom, there are four buttons: 'Previous', 'Next', 'Finish', and 'Cancel'.

### New Form Wizard - Step2

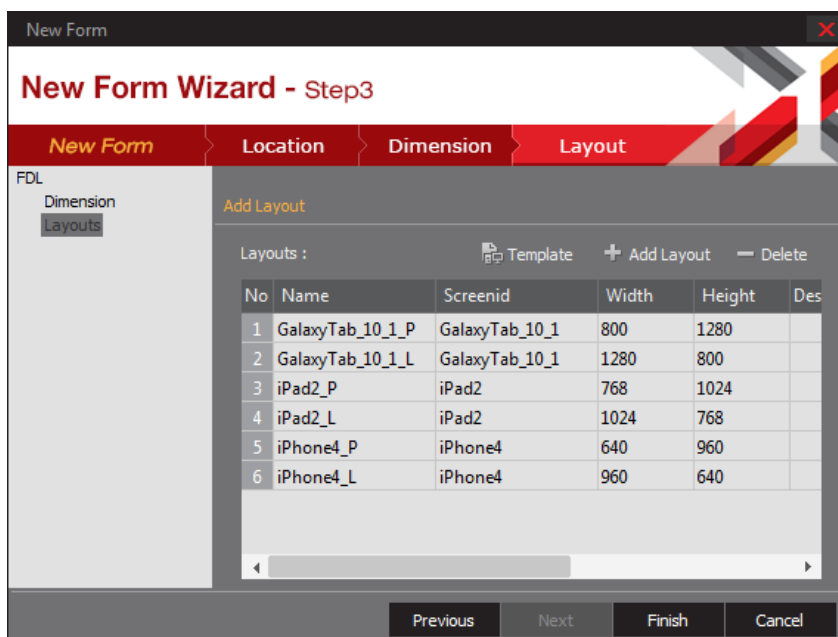
생성될 품의 'Default' 레이아웃 크기를 입력하는 단계입니다. 초기 입력값은 속성 창에서 변경할 수 있습니다.



The screenshot shows the 'New Form Wizard - Step2' window. The 'Dimension' tab is selected in the top navigation bar. The left sidebar shows 'FDL' with 'Dimension' and 'Layouts' options. The main area displays 'Form design window width / height.' and 'These are the window width / height settings :'. Below this, there are input fields for 'Width: 1024' and 'Height: 768'. At the bottom, there are buttons for 'Previous', 'Next', 'Finish', and 'Cancel'.

### New Form Wizard - Step3

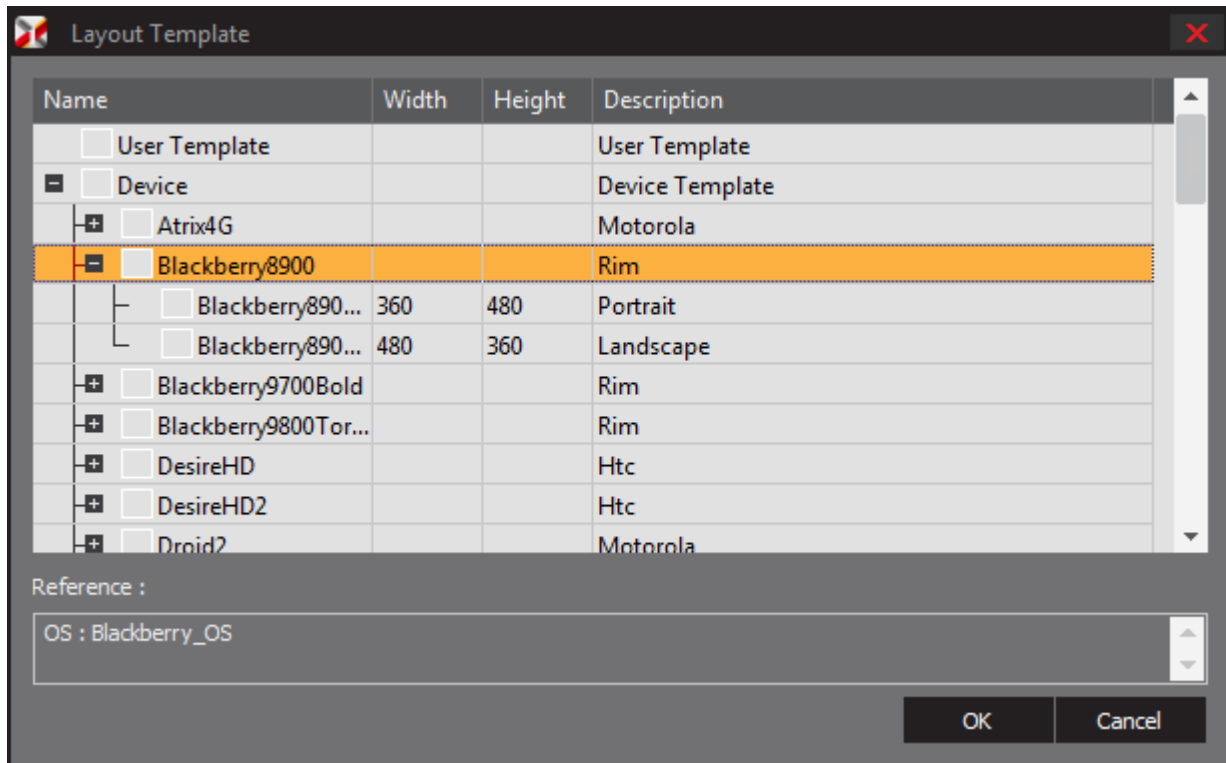
생성될 품에 필요한 레이아웃 정보를 입력하는 단계입니다. 스크린 종류별, 가로, 세로의 크기에 맞춘 레이아웃 정보를 설정합니다.



The screenshot shows the 'New Form Wizard - Step3' window. The 'Layout' tab is selected in the top navigation bar. The left sidebar shows 'FDL' with 'Dimension' and 'Layouts' options. The main area displays 'Add Layout' and a table of existing layouts. The table has columns: No, Name, Screenid, Width, Height, and Des. Below the table, there are buttons for 'Previous', 'Next', 'Finish', and 'Cancel'.

No	Name	Screenid	Width	Height	Des
1	GalaxyTab_10_1_P	GalaxyTab_10_1	800	1280	
2	GalaxyTab_10_1_L	GalaxyTab_10_1	1280	800	
3	iPad2_P	iPad2	768	1024	
4	iPad2_L	iPad2	1024	768	
5	iPhone4_P	iPhone4	640	960	
6	iPhone4_L	iPhone4	960	640	

템플릿 버튼을 누르면 아래 그림과 같은 Layout Template 대화상자가 나타나고 원하는 디바이스나 Graphic Array 정보를 선택한 후 OK 버튼을 누르면 선택한 레이아웃 정보가 New Form Wizard에 적용됩니다.



생성된 품과 관련된 상세한 옵션은 넥사크로 스튜디오 가이드를 참고하세요.

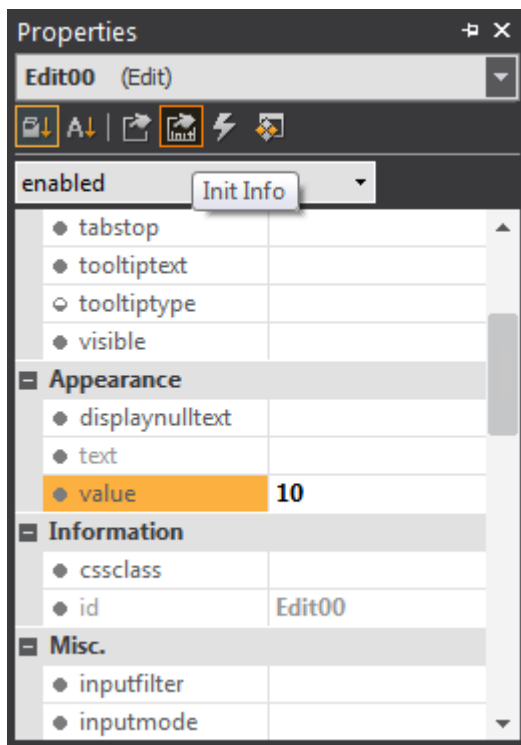


## 9.3 InitValue

MLM을 적용한 애플리케이션이 특정한 값을 가진 상태에서 시작해야 하는 경우가 있습니다. 예를 들어 각 레이아웃마다 다른 디자인 속성을 가지는 경우 레이아웃마다 다른 속성값을 지정해야 합니다.

하지만 사용자가 입력하거나 변경할 수 있는 속성은 레이아웃이 변한다고 해서 입력되거나 수정된 값이 처음 설정한 값으로 변경해서는 안 됩니다. 이런 항목에 속성값을 직접 지정하지 않고 InitValue를 선언해주면 처음 애플리케이션을 호출할 때만 해당 값을 사용하고 레이아웃 변경과 상관없이 수정된 값을 유지할 수 있습니다.

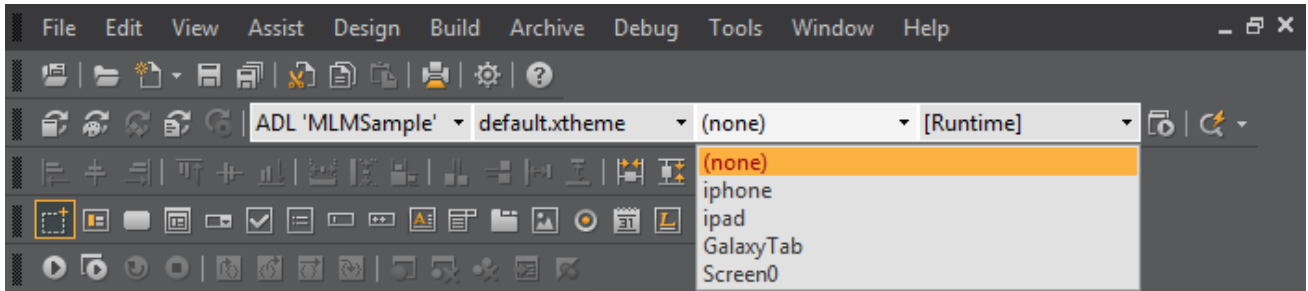
- 넥사크로 스튜디오 속성 창에 Init Info라는 항목이 추가됐습니다. 필요할 때 아래 그림과 같이 InitValue를 적용할 수 있습니다. 수정된 값은 xfdl 파일에 <InitValue> 태그로 추가됩니다.



- InitValue로 지정된 값은 품이 로딩되기 전에 적용되며 속성값은 품이 로딩되면서 적용됩니다. InitValue와 속성값을 같이 지정하는 경우에는 속성값이 InitValue를 덮어쓰게 됩니다.
- 레이아웃에 따라 InitValue를 따로 지정할 수는 없습니다.
- 컴포넌트의 속성값이 임의로 수정되지 않은 상태(Default Value)일 때는 레이아웃 변경이 속성값에 영향을 미치지 않습니다.

## 9.4 실행

- Quick View는 현재 활성화된 레이아웃 탭을 기준으로 보여주게 되어 있습니다.
- Launch Project는 기본 툴바에서 선택한 스크린 정보를 기준으로 보여주게 되어 있습니다.
- 스크린 정보를 입력하지 않거나 ADL에 없는 스크린일 경우 활성화된 ADL의 Theme를 적용해 보여주게 되어 있습니다.



## 9.5 XML 추가/변경 사항

### XPRJ

‘Layout Template’ 기능으로 생성되는 정보는 XPRJ 파일에 저장하게 되어 있습니다.

```
<?xml version="1.0" encoding="utf-8"?>
<Project active_adl="TEST" version="1.1">
  <TypeDefinition url="default_typedef.xml"/>
  <GlobalVariables url="globalvars.xml"/>
  <ADL id="TEST" url="TEST.xadl"/>
  <LayoutTemplate>
    <ScreenGroup name="ScreenGroup00" description="">
      <Layout name="Template00" width="1024" height="768" description=""/>
    </ScreenGroup>
  </LayoutTemplate>
</Project>
```

### XADL

프로젝트 생성 시 또는 ScreenInfo 항목을 직접 편집해 생성되는 정보는 XADL 파일에 저장됩니다.

```
<?xml version="1.0" encoding="utf-8"?>
<ADL version="1.2">
  <TypeDefinition url="default_typedef.xml"/>
  <GlobalVariables url="globalvars.xml"/>
  <Application id="TEST2">
    <Layout>
      ...
    </Layout>
    <ScreenInfo>
      <Screen name="Screen0" theme="default.xtheme"
        systemtype="iPhone" os="iOS" device="default"/>
    </ScreenInfo>
  </Application>
</ADL>
```

## XFDL

폼의 레이아웃 정보 및 컴포넌트의 속성값들은 XFDL 파일에 저장됩니다.

```
<?xml version="1.0" encoding="utf-8"?>
<FDL version="1.5">
  <TypeDefinition url="..\default_typedef.xml"/>
  <Form id="LayoutForm">
    <Layouts>
      <Layout width="1024" height="768"/>
      <Layout name="iPad_P" screenid="iPad" width="768" height="1024"/>
      <Layout name="iPad_L" screenid="iPad" width="1024" height="768"/>
    </Layouts>
  </Form>
</FDL>
```

‘Default’ 탭에서 Form의 속성이 변경된 경우는 <Form> 태그에 저장됩니다.

```
<?xml version="1.0" encoding="utf-8"?>
<FDL version="1.5">
  <TypeDefinition url="..\default_typedef.xml"/>
  <Form id="LayoutForm" style="background:red;">
    <Layouts>
      <Layout width="1024" height="768"/>
      <Layout name="iPad_P" screenid="iPad" width="768" height="1024"/>
      <Layout name="iPad_L" screenid="iPad" width="1024" height="768"/>
    </Layouts>
  </Form>
</FDL>
```

추가된 레이아웃 탭에서 폼의 속성이 변경된 경우 해당하는 <Layout> 태그에 저장됩니다.

```
<?xml version="1.0" encoding="utf-8"?>
<FDL version="1.5">
  <TypeDefinition url="..\default_typedef.xml"/>
  <Form id="LayoutForm" style="background:red;">
    <Layouts>
      <Layout width="1024" height="768"/>
      <Layout name="iPad_P" screenid="iPad" width="768" height="1024">
```

```

        style="background:yellow;"/>
        <Layout name="iPad_L" screenid="iPad" width="1024" height="768"/>
    </Layouts>
</Form>
</FDL>

```

추가된 레이아웃 탭에서 컴포넌트의 속성이 변경된 경우 해당하는 <Layout> 태그의 하위에 변경된 속성값만 저장됩니다.

```

<?xml version="1.0" encoding="utf-8"?>
<FDL version="1.5">
    <TypeDefinition url="..\default_typedef.xml"/>
    <Form id="LayoutForm">
        <Layouts>
            <Layout width="1024" height="768">
                <Button id="Button00" text="Button00"
                    left="20" top="20" width="120" height="50"/>
            </Layout>
            <Layout name="iPad_P" screenid="iPad" width="768" height="1024">
                <Button id="Button00" style="color:blue;"/>
            </Layout>
            <Layout name="iPad_L" screenid="iPad" width="1024" height="768"/>
        </Layouts>
    </Form>
</FDL>

```

InitValue가 추가되면 <InitValue> 태그로 저장됩니다. 폼에 대한 InitValue를 지정하는 경우에는 InitValue 태그 자체 속성으로 처리되고 각 컴포넌트의 InitValue는 <InitValue> 태그 안에 변경된 속성값이 처리됩니다.

```

<?xml version="1.0" encoding="utf-8"?>
<FDL version="1.5">
    <TypeDefinition url="..\default_typedef.xml"/>
    <Form id="LayoutForm">
        <Layouts>
        </Layouts>
        <InitValue>
            <Button id="Button00" style="color:blue;"/>
        </InitValue>
    </Form>
</FDL>

```

서브 레이아웃은 폼을 구성하는 XML과 기본 구조는 같습니다. 대상 컴포넌트 하위에 default 레이아웃이 있고 그 하위에 서브 레이아웃이 생성됩니다. 각 서브 레이아웃에서 수정된 속성은 <Layout> 태그의 하위에 변경된 속성값만 저장됩니다.

```
<?xml version="1.0" encoding="utf-8"?>
<FDL version="1.5">
  <TypeDefinition url="..\default_typedef.xml"/>
  <Form id="LayoutForm">
    <Layouts>
      <Layout width="1024" height="768">
        <Div id="Div00" taborder="0" left="20" top="20" width="300" height="300">
          <Layouts>
            <Layout width="300" height="300">
              <Button id="Button00" left="17" top="67" width="120" height="50"/>
            </Layout>
            <Layout name="SubLayout00" width="200" height="200">
              <Button id="Button00" style="color:blue;"/>
            </Layout>
          </Layouts>
        </Div>
      </Layout>
      <Layout name="iPad_P" screenid="iPad" width="768" height="1024"/>
      <Layout name="iPad_L" screenid="iPad" width="1024" height="768"/>
    </Layouts>
  </Form>
</FDL>
```

## 파트 III.

---

### 확장

# 10.

## 프로토콜 어댑터

넥사크로플랫폼에서 제공하는 데이터 처리는 데이터 자체를 변환해 처리하지는 않습니다. 기본 프로토콜인 HTTP를 사용해 요청과 응답을 처리합니다. 하지만 사용자 환경에 따라 데이터를 암호화해서 통신하거나 원하는 형식으로 변환해 처리하는 과정을 필요로 할 수 있습니다.

프로토콜 어댑터는 다른 데이터 구조를 필요로 하는 관계를 서로 연결해 원하는 형태로 동작하게 만드는 도구입니다. 간단한 설정만으로 원하는 기능을 쉽게 만들 수 있습니다.

### 10.1 프로토콜 어댑터

프로토콜, 모듈, 서비스 항목은 프로젝트 폴더에 있는 default\_typedef.xml 파일을 직접 등록하거나 프로젝트 탐색창에서 TypeDefinition 항목을 두 번 클릭해 열리는 [Edit TypeDefinition]창에서 해당 항목을 추가할 수 있습니다.

#### 모듈(Module) 등록

프로젝트 폴더에 있는 default\_typedef.xml 파일을 직접 수정하거나 [Edit TypeDefinition > Modules]창에서 해당 항목을 추가합니다.

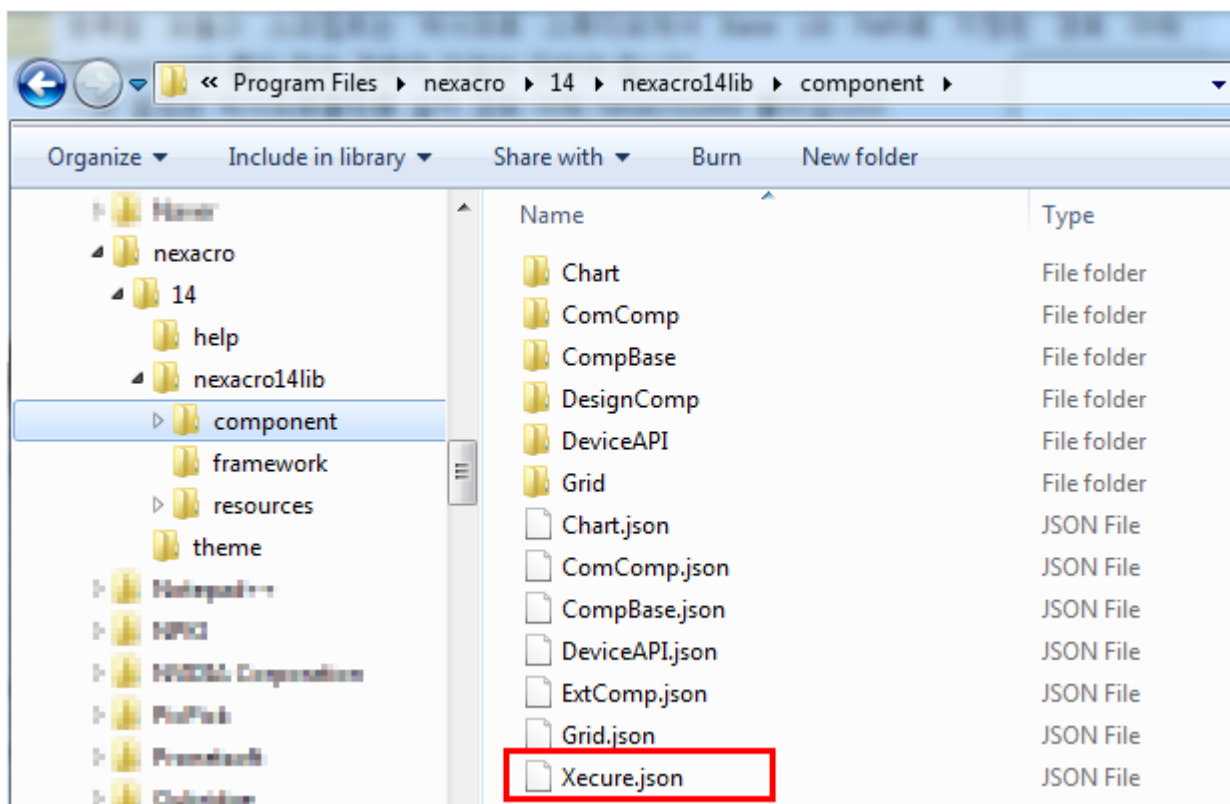
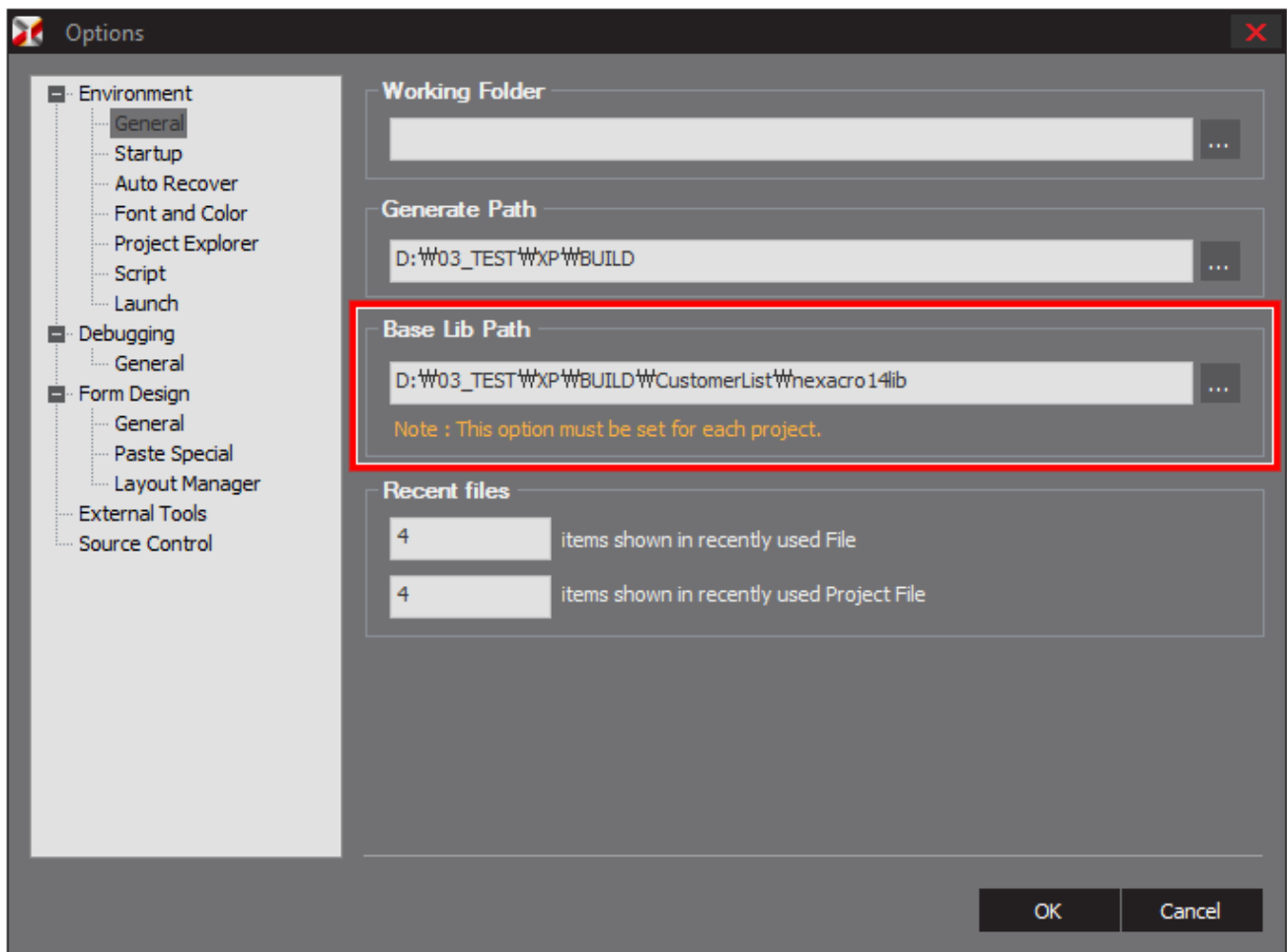


등록할 모듈과 스크립트는 넥사크로 스튜디오에서 Base Lib Path로 지정된 경로 아래 [component] 폴더 하위 경로에 파일이 있어야 합니다.

기본 설정은 넥사크로플랫폼 설치 경로 아래 nexacro14lib 폴더입니다.

Base Lib Path는 [Tools > Options > Environment > Base Lib Path]에서 다른 경로로 변경할 수 있습니다.



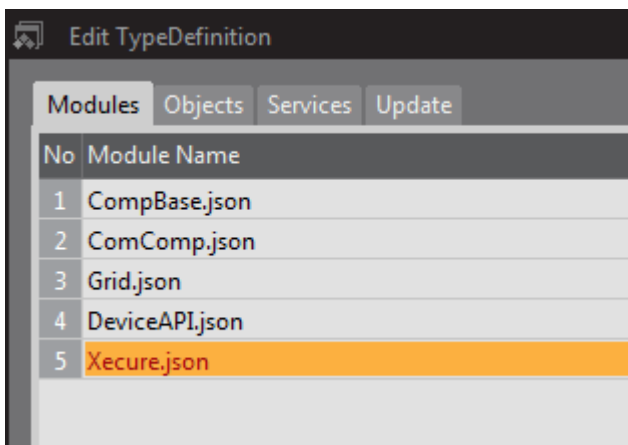


모듈은 아래와 같은 형태로 JSON 파일을 작성합니다. 'scripts' 항목에 지정된 자바스크립트 경로는 [Base Lib Path > component] 폴더 아래 해당 폴더와 스크립트 파일이 있어야 합니다.

```
{
  "name": "Xecure",
  "version": "14.0.0.0",
  "description": "nexacro platform XecureAdp Protocol Library",
  "license": "",
  "scripts": [
    "Xecure/Xecure.js",
  ]
}

//@ sourceMappingURL=Xecure.json
```

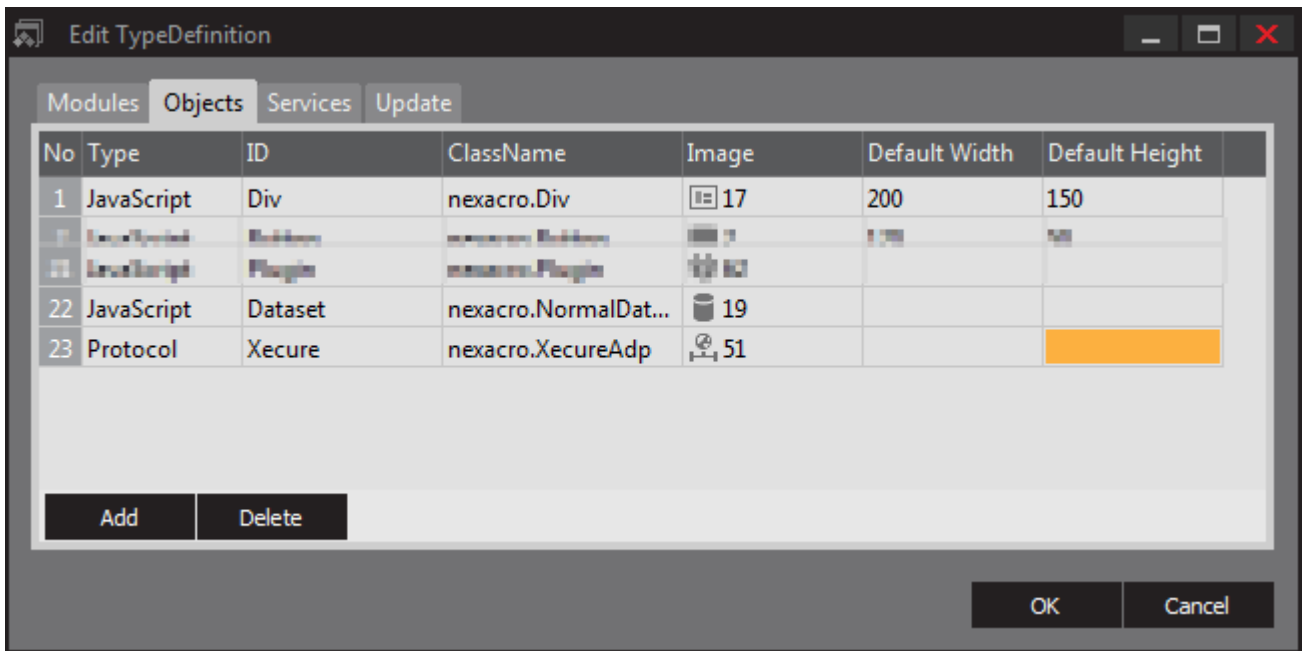
```
<Module url="Xecure.json"/>
```



## 프로토콜(Protocol) 등록

프로젝트 폴더에 있는 default\_typedef.xml 파일을 직접 수정하거나 [Edit TypeDefinition > Objects]창에서 해당 항목을 추가합니다.

```
<Component type="Protocol" id="Xecure" classname="nexacro.XecureAdp"/>
```



프로토콜로 등록되는 클래스에 정의된 인터페이스를 해당 시점에 맞게 호출하는 방식으로 동작합니다. 프레임워크 내부에서 동작하는 방식은 아래와 같습니다.

1. nexacro.ProtocolAdp를 상속받은 클래스로 작성합니다.

등록된 프로토콜이 처음 호출되는 시점에 ProtocolAdptor 오브젝트를 생성합니다. 해당 오브젝트는 애플리케이션이 종료되기전까지 통신이 호출되는 시점에 사용됩니다.

```
var _pXecureAdp = nexacro._createPrototype(nexacro.ProtocolAdp);
nexacro.XecureAdp.prototype = _pXecureAdp;
```

2. 오브젝트가 생성되는 시점에 initialize 함수가 호출됩니다.

```
_pXecureAdp.initialize = function ()
{
    trace("_pXecureAdp.initialize");
};
```

3. 오브젝트가 삭제되는 시점에 finalize 함수가 호출됩니다.

애플리케이션이 내려가는 시점에 호출됩니다.

```
_pXecureAdp.finalize = function ()
{
    trace("_pXecureAdp.finalize");
};
```

4. 내부에서 실제 통신이 호출되기 전에 encrypt 함수가 호출됩니다.  
통신이 발생하기 전에 전송할 데이터를 암호화하거나 원하는 형태로 변환합니다.

```
_pXecureAdp.encrypt= function(url, data)
{
    trace("encrypt url=" + url + ";data=" + data);
    return data;
};
```



encrypt 함수는 요청 방식(Request Method)이 POST 방식일 때만 호출됩니다.  
transaction이나 load 메소드 사용 시 GlobalVariables와 관련된 별도 설정이 없다면 POST 방식으로 동작합니다.

5. 내부에서 실제 통신이 끝난 후에 decrypt 함수가 호출됩니다.  
수신한 데이터를 복호화하거나 원하는 형태로 변환합니다.

```
_pXecureAdp.decrypt = function (url, data)
{
    trace("decrypt url=" + url + ";data=" + data);
    return data;
};
```

프로토콜로 등록되는 클래스의 전체 코드는 아래와 같습니다. 사용 환경에 따라 실제 코드는 수정해 사용해야 합니다.

```
if (!nexacro.XecureAdp)
{
    nexacro.XecureAdp = function ()
    {
    };

    var _pXecureAdp = nexacro._createPrototype(nexacro.ProtocolAdp);
    nexacro.XecureAdp.prototype = _pXecureAdp;

    _pXecureAdp._type = "nexacroXecureAdp";
    _pXecureAdp._type_name = "XecureAdp";

    _pXecureAdp.encrypt= function(url, data)
    {
```

```

        trace("encrypt url=" + url + ";data=" + data);
        return data;
    };

    _pXecureAdp.decrypt = function (url, data)
    {
        trace("decrypt url=" + url + ";data=" + data);
        return data;
    };

    _pXecureAdp.initialize = function ()
    {

        trace("_pXecureAdp.initialize");
    };

    _pXecureAdp.finalize = function ()
    {
        trace("_pXecureAdp.finalize");
    };

    delete _pXecureAdp;
}

```

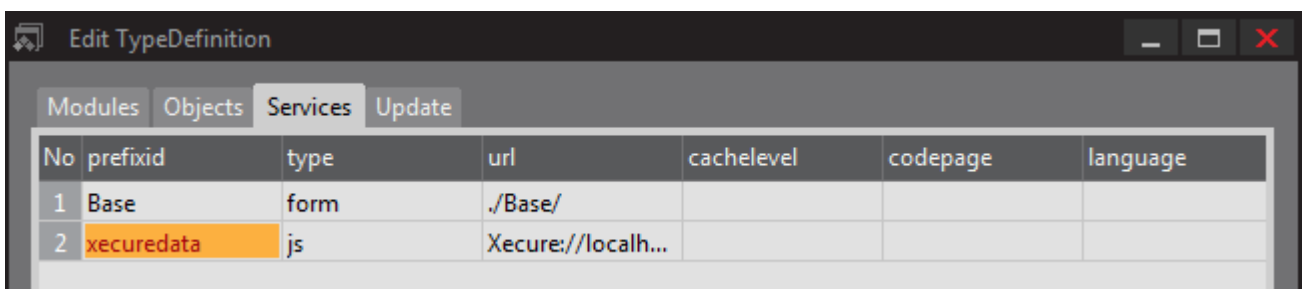
## 서비스 등록

프로젝트 폴더에 있는 default\_typedef.xml 파일을 직접 수정하거나 [Edit TypeDefinition > Services]창에서 해당 항목을 추가합니다.

```

<Service prefixid="xecuredata" type="js" url="Xecure://localhost/" version="0"
communicationversion="0"/>

```



서비스 그룹으로 등록된 폼을 프로토콜 어댑터를 통해 호출하려면 서비스 그룹 등록 시 url을 아래와 같이 등록하고 사용할 수 있습니다.

```
<Service prefixid="Base" type="form" url="Xecure://localhost/Base" version="0"
communicationversion="0"/>
```

```
this.Div00.set_url('Base:test.xfdl');
```

## 데이터 호출

등록한 서비스를 호출하게 되면 해당 프로토콜을 거쳐 통신이 진행됩니다.

```
application.transaction("MyService01", "xecuredata::test.js",
"dsIn=dsIn:A","dsOut=dsIn","a=b","fnCallback");
```

데이터 처리에 대한 전체적인 흐름을 정리하면 아래와 같습니다.

1. 해당 프로토콜로 데이터 호출  
Application.transaction(), Dataset.load(), Div.set\_url();
2. 프로토콜 초기화  
initialize
3. 전송 데이터 변환  
encrypt
4. 서버 호출  
변환된 데이터를 지정된 서버로 전송합니다.
5. 서버 응답
6. 수신 데이터 변환  
decrypt
7. 콜백 함수 또는 데이터 처리  
호출 유형에 따라 데이터를 처리합니다.

## 10.2 PluginElement

데이터 처리 시 암호화와 같은 추가적인 보안이 필요한 경우에는 스크립트로 처리하지 않고 외부 플러그인을 설정해야 합니다. 이런 경우에는 PluginElement를 사용해 외부 플러그인을 적용할 수 있습니다.

외부 플러그인 적용은 해당 솔루션에서 제공하는 규칙에 따라 추가적인 코드를 필요로 할 수 있습니다. 아래 설명은 일반적으로 PluginElement를 사용하는 가이드를 제공합니다.



아래 설명은 웹 구간 암호화 처리를 위해 제큐어웹(XecureWeb)을 적용하는 경우에 대한 설명입니다. 참고로 작성된 예제로 일부 내용이 누락됐을 수 있습니다. 제품에 대한 자세한 내용은 아래 링크(한국어)를 참고하세요.  
[http://www.softforum.co.kr/02product/pro\\_sub01\\_02.asp](http://www.softforum.co.kr/02product/pro_sub01_02.asp)



런타임 버전을 사용하는 경우 외부 플러그인은 애플리케이션 최초 실행 시 설치 웹페이지를 불러와 설치하는 형태를 권장합니다. 설치 시 문제가 발생했을 때 웹 브라우저를 통해 설치해야 플러그인 문제인지 넥사크로플랫폼 문제인지 확인할 수 있습니다.

## 프로토콜(Protocol) 클래스 코드

PluginElement를 사용할 때 프로토콜 어댑터를 사용하는 방식은 같고 프로토콜로 등록되는 클래스 코드만 일부 변경합니다.

- 오브젝트가 생성되는 시점에 initialize 함수가 호출됩니다.  
 내부에서 사용하는 플러그인은 nexacro.PluginElement를 사용할 수 있습니다.  
 Parent\_element 없이 생성해 프레임워크 내부 hidden 영역에 추가해 화면에 나타나지는 않습니다.

```
_pXecureAdp.initialize = function ()
{
    trace("_pXecureAdp.initialize");
    var parent_elem = null;
    this.xecure = new nexacro.PluginElement();
    this.xecure.setElementClassId("{7E9FDB80-5316-11D4-B02C-00C04F0CD404}");
    this.xecure.create();
};
```

- 오브젝트가 삭제되는 시점에 finalize 함수가 호출됩니다.  
애플리케이션이 내려가는 시점에 호출됩니다.

```
_pXecureAdp.finalize = function ()
{
    this.xecure.destroy();
    delete this.xecure;
};
```

- 내부에서 실제 통신이 호출되기 전에 encrypt 함수가 호출됩니다.  
통신이 발생하기 전에 전송할 데이터를 암호화합니다.

```
_pXecureAdp.encrypt= function(url, data)
{
    return data;
};
```

- 내부에서 실제 통신이 끝난 후에 decrypt 함수가 호출됩니다.  
수신한 데이터를 복호화합니다.

```
_pXecureAdp.decrypt = function (url, data)
{
    return data;
};
```

- 통신 시 사용할 프로토콜은 getUsingProtocol 함수에서 반환합니다.  
기본값으로 설정된 프로토콜은 http입니다.

```
_pXecureAdp.getUsingProtocol = function (url)
{
    return "http";
};
```

- 통신 시 추가로 쿠키에 들어갈 정보는 getCommunicationHeaders 함수에서 반환합니다.

```
_pXecureAdp.getCommunicationHeaders = function (url)
{
    var headers = [];
    headers.push({id:"test", value:"test1"});
    return headers;
};
```



```
};
```

# 11.

---

## 확장 모듈

외부 모듈(Extension DLL)을 사용하기 위해서는 지정된 절차에 따라 확장 모듈(DLL)을 만들어야 합니다. 새로 모듈을 만들거나 기존 모듈을 사용하기 위해 필요한 내용을 설명합니다.

### 11.1 개발 환경

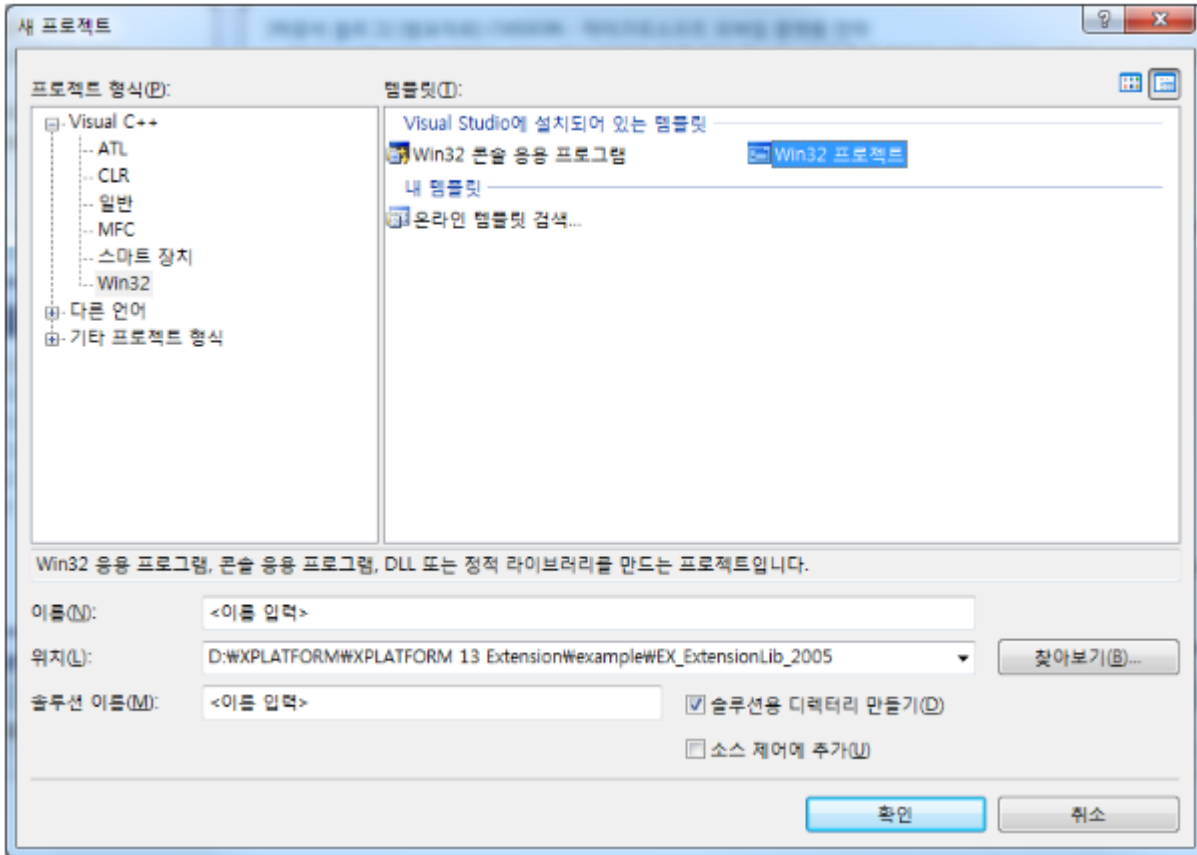
넥사크로플랫폼에서 사용할 수 있는 외부 모듈을 만들기 위해서는 아래와 같은 도구와 라이브러리가 필요합니다.

1. 마이크로소프트 비주얼 스튜디오 2005  
비주얼 스튜디오 버전이 2005보다 높은 경우에는 플랫폼 도구 집합을 v80으로 빌드합니다.
2. V8 자바스크립트 엔진  
<https://code.google.com/p/v8/>

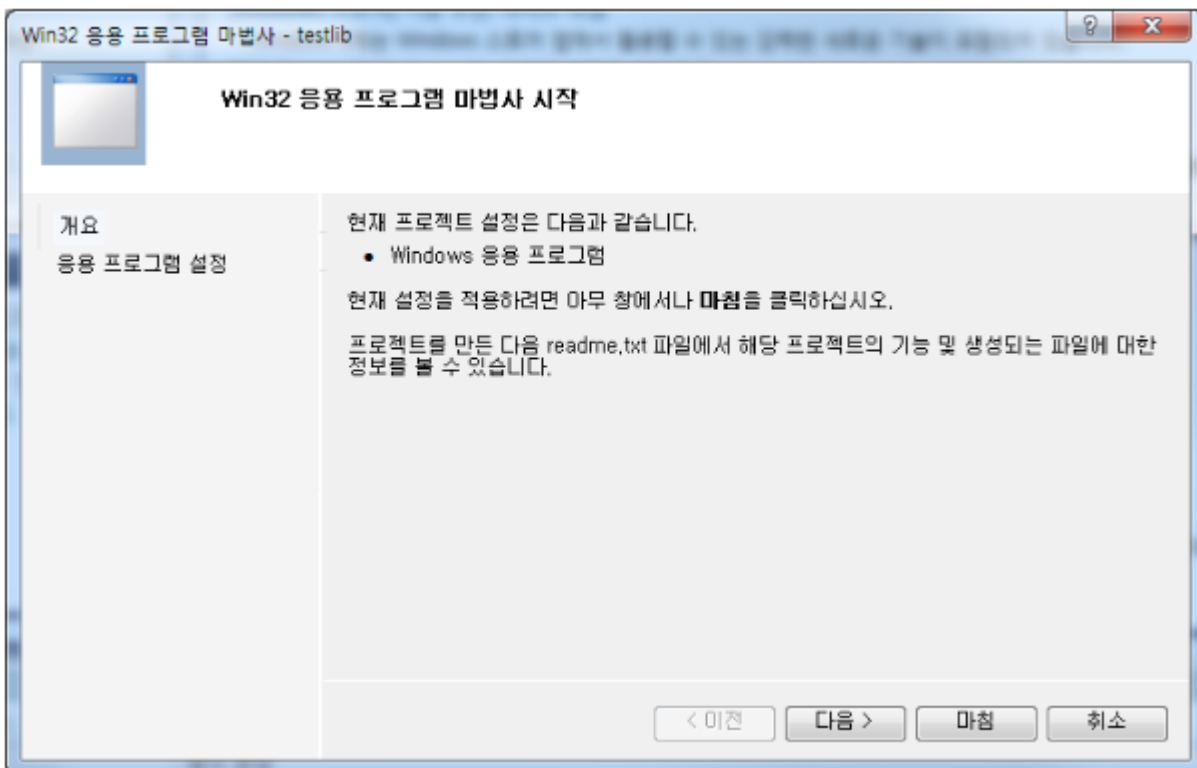
### 11.2 프로젝트 생성과 설정

#### 프로젝트 생성

1. [새 프로젝트]를 실행 후 [템플릿 > Visual C++ > Win32] 항목에서 [Win32 프로젝트]를 선택합니다. 이름 항목에 프로젝트 이름을 넣고 [확인]을 클릭합니다.



2. 설정 확인 후 [다음] 버튼을 클릭합니다.



3. [응용 프로그램 설정]에서 응용 프로그램 종류에서 DLL을 선택하고 [마침] 버튼을 클릭합니다.

## 프로젝트 설정

1. [추가 포함 디렉터리]에 V8 Javascript Engine의 v8.h 경로를 추가합니다.
2. [추가 종속성]에 'v8.lib'를 추가합니다.
3. [추가 라이브러리 디렉터리]에 [추가 종속성]에 있는 v8 Lib의 경로를 추가합니다.

## 11.3 확장 모듈 제작

넥사크로플랫폼 엔진에서는 확장 모듈의 ExtensionInitialize 정보를 읽고 설정합니다. ExtensionInitialize에서는 DLL에 필요한 초기화 작업과 attach 오브젝트와 같은 function 포인터 값을 설정합니다.

### Entry Funcs

v8::Object에 v8::Function 을 설정하는 함수를 지정합니다.

```
typedef bool (*pFnAttachObject)( v8::Handle<v8::Object>*);
```

### Initialize

초기화 단계에서는 아래와 같은 작업을 처리합니다.

- 넥사크로플랫폼에서 nexacro.\_addExtensionModule 함수로 확장 모듈을 로 처음 로드할 때 호출됩니다.
- nexacro.\_clearExtensionModule 함수가 호출되기 전까지 호출되지 않습니다.
- 확장모듈에서 초기화에 필요한 작업들을 진행합니다.

```
extern "C" __declspec(dllexport) bool ExtensionInitialize()
```

### EntryPoint

EntryPoint 단계에서는 아래와 같은 작업을 처리합니다.

- EXTENSIONENTRYFUNCS를 설정합니다.
- 넥사크로플랫폼에서 nexacro.\_addExtensionModule 함수로 매번 호출되며 인자로 넘어온 자바스크립트 오브젝트에 v8::Function 등을 설정합니다.

```
extern "C" __declspec(dllexport) bool ExtensionEntryPoint(EXTENSIONENTRYFUNCS*)
```

## Shutdown

- 넥사크로플랫폼에서 nexacro.\_clearExtensionModule을 통해 호출됩니다.
- 확장 모듈을 더 이상 사용하지 않을 때 필요한 작업을 진행합니다.

```
extern "C" __declspec(dllexport) void ExtensionShutdown()
```

## 11.4 넥사크로플랫폼 API

확장 모듈 개발과 관련된 넥사크로플랫폼 API는 아래와 같습니다.

### nexacro.\_addExtensionModule

확장 모듈 로딩 시 사용하는 API 입니다.

```
nexacro._addExtensionModule = function(object, modulepath)
```

Argument	설명
object	확장 모듈의 속성, 함수 등을 저장하는 오브젝트
modulepath	확장 모듈의 전체 경로를 지정합니다. 넥사크로플랫폼에서 지원하는 Path Alias를 사용할 수 있습니다.

```
var ext = {};
nexacro._addExtensionModule(ext, "c:/extension.dll");
nexacro._addExtensionModule(ext, "%COMPONENT%\extension.dll");
```

## nexacro.\_clearExtensionModule

확장 모듈을 더 이상 사용하지 않고 로딩된 모듈을 내려야 할 때 사용하는 API 입니다.

```
nexacro._clearExtensionModule = function(modulepath)
```

Argument	설명
modulepath	내려야 하는 확장 모듈의 전체 경로를 지정합니다. 넥사크로플랫폼에서 지원하는 Path Alias를 사용할 수 있습니다. 값을 주지 않는다면 로딩된 확장 모듈을 모두 내립니다.



nexacro.\_clearExtensionModule 함수로 제거된 확장 모듈의 오브젝트를 계속 사용할 경우 메모리 누수(memory leak)가 발생할 수 있으므로 로딩되지 않은 모듈의 오브젝트를 사용하면 안됩니다.

## 12.

# 사용자 컴포넌트

넥사크로플랫폼에서는 기본적으로 사용자들이 필요로 하는 컴포넌트를 제공하고 있습니다. 하지만 사용자 환경에 따라 특화된 기능을 필요로 하는 경우가 있습니다. 이럴 때는 투비소프트 파트너 개발사 또는 개발자가 작성한 사용자 컴포넌트를 적용하거나 직접 필요한 기능을 추가한 사용자 컴포넌트를 작성할 수 있습니다.

이번 장에서는 사용자 컴포넌트가 제공되는 형식과 이를 사용하는 방법에 대한 가이드를 제공합니다.



사용자 컴포넌트를 작성하기 위한 가이드는 파트너 개발사에 일부 제공되고 있으며 앞으로 전체 사용자에게 제공될 예정입니다.

이 문서에서는 사용자 컴포넌트를 작성하는 구체적인 방법에 대해서는 다루지 않으며 이미 개발된 사용자 컴포넌트를 적용하는 방법만 다룹니다.

## 12.1 사용자 컴포넌트 구성

사용자 컴포넌트는 다음과 같은 형식으로 구성됩니다. 사용자 컴포넌트는 [10. 프로토콜 어댑터](#)에서 모듈과 프로토콜을 등록하는 방법과 비슷하지만 약간의 차이가 있습니다.

사용자 컴포넌트는 구성에 따라 제공되는 파일의 개수가 달라질 수 있지만, 기본적으로 제공되는 파일은 3가지입니다.

1. 모듈 파일 (\*.json)
2. 오브젝트 정보 파일 (\*.info)
3. 컴포넌트 스크립트 파일 (\*.js)

### 모듈(Module) 등록

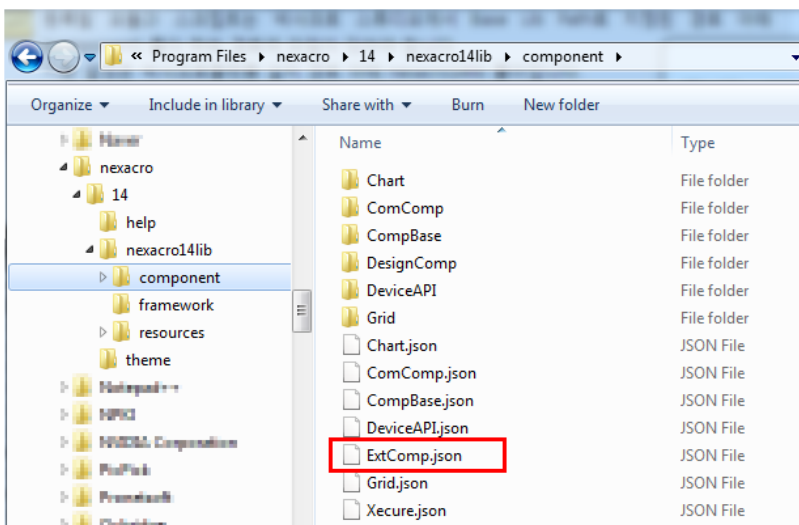
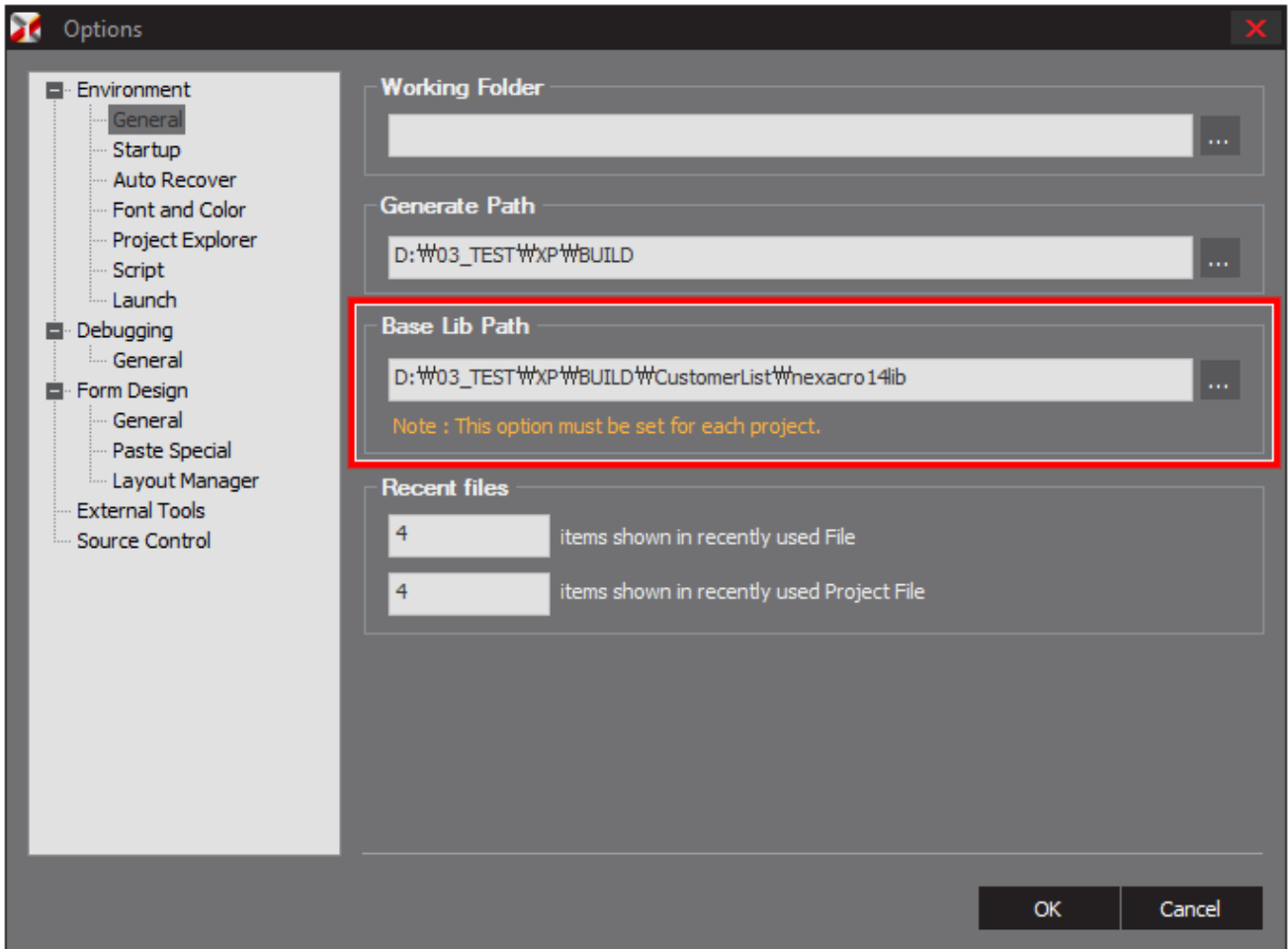
프로젝트 폴더에 있는 default\_typedef.xml 파일을 직접 수정하거나 [Edit TypeDefinition > Modules]창에서 해당 항목을 추가합니다.



등록할 모듈과 스크립트는 넥사크로 스튜디오에서 Base Lib Path로 지정된 경로 아래 [component] 폴더 하위 경로에 파일이 있어야 합니다.

기본 설정은 넥사크로플랫폼 설치 경로 아래 nexacro14lib 폴더입니다.

Base Lib Path는 [Tools > Options > Environment > Base Lib Path]에서 다른 경로로 변경할 수 있습니다.





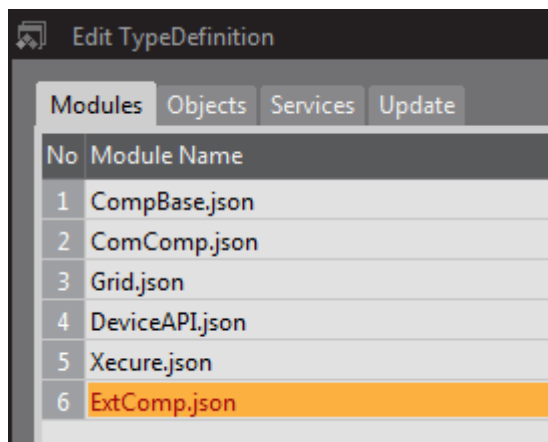
모듈은 아래와 같은 형태로 JSON 파일을 작성합니다. 'scripts', 'objInfo' 항목에 지정된 자바스크립트 경로는 [Base Lib Path > component] 폴더 아래 해당 폴더와 스크립트 파일이 있어야 합니다.

```
{
  "name": "ComComp",
  "version": "14.0.0.0",
  "description": "nexacro platform Common Component Library",
  "license": "",
  "scripts": [
    "ExtComp/ExtCombo.js"
  ],
  "objInfo": [
    "ExtComp/ExtCombo.info",
    "ExtComp/ExtEnum.info"
  ]
}
```

//@ sourceMappingURL=ExtComp.json

10. [프로토콜 어댑터](#)와 다른 점은 사용자 컴포넌트에서 제공하는 속성, 메소드, 이벤트, Enum 관련 정보를 'objInfo' 항목으로 설정하고 있습니다.

```
<Module url="ExtComp.json"/>
```



## 오브젝트 정보

nexacro.ExtCombo 컴포넌트는 모듈 설정 시 'ExtCombo.info', 'ExtEnum.info' 2개의 ObjInfo 파일을 설정했습니다. 'ExtCombo.info' 파일은 컴포넌트에 필요한 정보를 담고 있으며 'ExtEnum.info' 파일은 Enum 속성에 대한 정보를 담고 있습니다. Enum 속성 정보는 용도에 따라 컴포넌트 info 파일에 포함하거나 별도 파일로 작성할 수 있

습니다.



넥사크로 스튜디오에서는 ObjInfo에 지정된 파일 내용을 기준으로 컴포넌트를 추가하고 속성창에 정보가 나타나며 실제 실행되는 애플리케이션의 동작은 scripts에 설정된 자바스크립트 파일에 정의된 내용에 따라 처리됩니다.

ExtCombo.info 파일에서 추가되거나 재정의된 항목은 아래와 같습니다. 자세한 내용은 전체 코드를 참조하세요.

- 새로 추가된 속성  
userprop  
itemopacity
- 새로 추가된 Pseudo  
pushed  
focused
- 상속받은 속성을 재정의  
tooltiptext
- 추가된 메소드  
testFunc
- 추가된 이벤트  
ontest



제공된 코드는 이해를 돕기 위해 제공되며 특정 기능을 수행하기 위한 용도는 아닙니다.

```
<?xml version='1.0' encoding='utf-8'?>
<MetaInfo version="1.0">
  <Object id="nexacro.ExtCombo">

    <!-- define extend component based nexacro.Combo -->
    <ObjectInfo
      inheritance="nexacro.Combo"
      typename="nexacro.ExtCombo"
      csstypename="ExtCombo,ExtComboABC[userprop='abc']"
      csscontrolname="ExtComboControl,ExtComboControlABC[userprop='abc']"
      group="UIComponent"
      csspseudo="true"
      container="false"
      composite="true"
      tabstop="true"
      cssstyle="true"
```

```

contents="true"
formats="false"
contentseditor="auto"
defaultwidth="300"
defaultheight="200"
requirement="Runtime,HTML5"
description=""/>

```

```
<PseudoInfo>
```

```
<Pseudo
```

```

    name="pushed"
    control="true"
    deprecated="false"
    unused="false"

```

```
/>
```

```
<Pseudo
```

```

    name="focused"
    control="true"
    deprecated="false"
    unused="true"

```

```
/>
```

```
</PseudoInfo>
```

```
<ControlInfo>
```

```
</ControlInfo>
```

```
<PropertyInfo>
```

```
<!-- define new property -->
```

```
<Property
```

```

    name="userprop"
    group="Misc"
    type="Enum"
    defaultvalue="abc"
    readonly="false"
    initonly="false"
    hidden="false"
    control="false"
    style="false"
    expr="false"
    deprecated="false"

```

```

        unused="false"
        objectinfo=""
        enuminfo="enum_ext_test"
        unitinfo=""
        requirement="Runtime,HTML5"
        description="this is desc"
    />
<!-- re-define super's property -->
<Property
    name="tooltiptext"
    group="Misc"
    type="String"
    defaultvalue="1111111"
    readonly="false"
    initonly="false"
    hidden="false"
    control="false"
    style="false"
    expr="false"
    deprecated="false"
    unused="false"
    objectinfo=""
    enuminfo=""
    unitinfo=""
    requirement="Runtime,HTML5"
    description="this is desc"
/>
<!-- define new style property -->
<Property
    name="itemopacity"
    group="Style"
    type="Opacity"
    defaultvalue=""
    readonly="false"
    initonly="false"
    hidden="false"
    control="false"
    style="true"
    expr="false"
    deprecated="false"
    unused="false"

```

```

        objectinfo="nexacro.Style_opacity"
        enuminfo=""
        unitinfo=""
        requirement="Runtime,HTML5"
        description="this is desc"
    />
</PropertyInfo>

<MethodInfo>
    <Method
        name="testFunc"
        group=""
        async="false"
        deprecated="false"
        unused="false"
        requirement="Runtime,HTML5"
        description="this is test method"
    >
        <Syntax
            text = "testFunc(a [, b])"
        >
        <Return/>
        <Arguments>
            <Argument
                name="a"
                type="String"
                in="true"
                out="false"
                option="false"
                variable="false"
                description="any string"
            />
            <Argument
                name="b"
                type="String"
                in="true"
                out="false"
                option="true"
                variable="false"
                description="any string"
            />
        </Arguments>
    </Method>
</MethodInfo>

```

```

        />
    </Arguments>
</Syntax>
</Method>
</MethodInfo>

<EventHandlerInfo>
    <!-- define event -->
    <EventHandler
        name="ontest"
        group="Event"
        deprecated="false"
        unused="false"
        requirement="Runtime,HTML5"
        description="this is test event"
    >
        <Syntax
            text="ontest(obj:Object, e:nexacro.EventInfo)"
        >
            <Return/>
            <Arguments>
                <Argument
                    name="obj"
                    type="Object"
                    in="true"
                    out="false"
                    option="false"
                    variable="false"
                    description="Event Source Object"
                />
                <Argument
                    name="e"
                    type="nexacro.EventInfo"
                    in="true"
                    out="false"
                    option="false"
                    variable="false"
                    description="Event Information Object"
                />
            </Arguments>

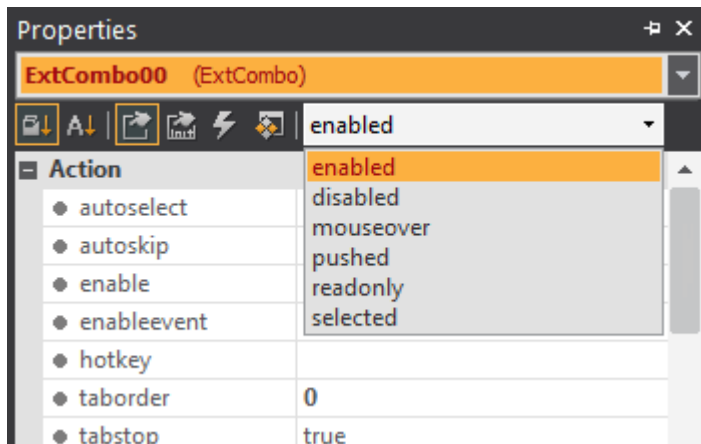
```

```

        </Syntax>
    </EventHandler>
</EventHandlerInfo>
</Object>
</MetaInfo>

```

사용자 컴포넌트를 화면에 배치하면 속성창에 추가된 Pseudo 항목이나 속성, 이벤트를 확인할 수 있습니다.



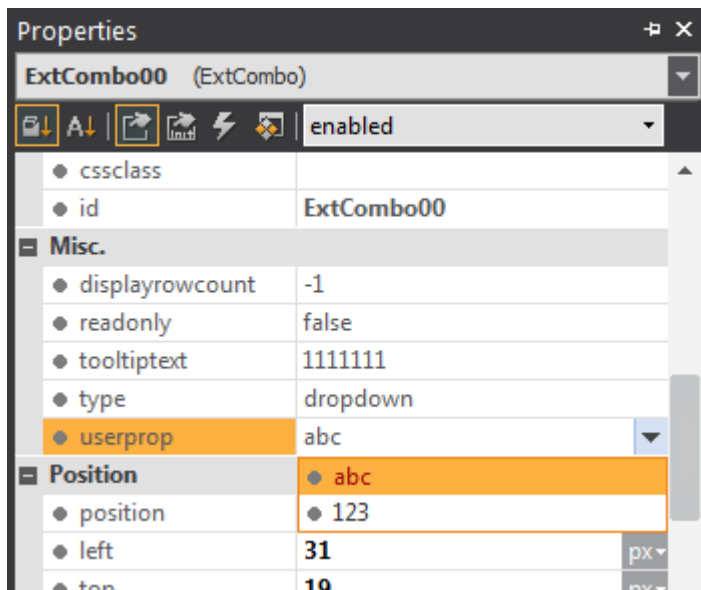
예제 코드에서 userprop 속성을 설정하는 항목 중 enuminfo는 직접 값을 입력하지 않고 'enum\_ext\_test'라는 Enum 항목을 지정했습니다. 해당 항목에 대한 내용은 'ExtEnum.info' 파일에서 찾을 수 있습니다.

```

<?xml version='1.0' encoding='utf-8'?>
<MetaInfo version="1.0">
    <!-- define enum information -->
    <EnumInfo
        id="enum_ext_test"
        composite="false"
        delimiter=""
        description="abc, 123"
    >
        <Enum
            name="abc"
            description="abc"
        />
        <Enum
            name="123"
            description="123"
        />
    </EnumInfo>
</MetaInfo>

```

Enum 항목을 지정했을 때 속성창에서 해당 속성의 입력영역은 선택목록 형태로 나타납니다.



## 컴포넌트 스크립트

애플리케이션 실행 시 컴포넌트의 각 속성과 메소드, 이벤트를 어떻게 처리할지에 대한 내용은 자바스크립트로 작성된 파일에 추가되어야 합니다.

ExtCombo.js 파일 전체 코드는 아래와 같습니다.



제공된 코드는 이해를 돕기 위해 제공되며 특정 기능을 수행하기 위한 용도는 아닙니다.

```
if (!nexacro.ExtCombo)
{
    // =====
    // nexacro.ExtCombo
    // =====
    nexacro.ExtCombo = function(id, position, left, top, width, height, right, bottom, parent)
    {
        nexacro.Combo.call(this, id, position, left, top, width, height, right, bottom,
parent);

        this.userprop = "abc";
        this.itemopacity = "";
        this.onclick = null;
    };
};
```



```

var _pExtCombo = nexacro._createPrototype(nexacro.Combo);
nexacro.ExtCombo.prototype = _pExtCombo;
_pExtCombo._type = "nexacroExtCombo";
_pExtCombo._type_name = "ExtCombo";

_pExtCombo.set_userprpo = function (v)
{
    this.userprop = v;
}

_pExtCombo.set_itemopacity = function (v)
{
    this.itemopacity = v;
}
_pExtCombo.testFunc = function (a,b)
{
    return a+b;
}

delete _pExtCombo;
}

```

## 12.2 사용자 컴포넌트 적용

### 컴포넌트 등록

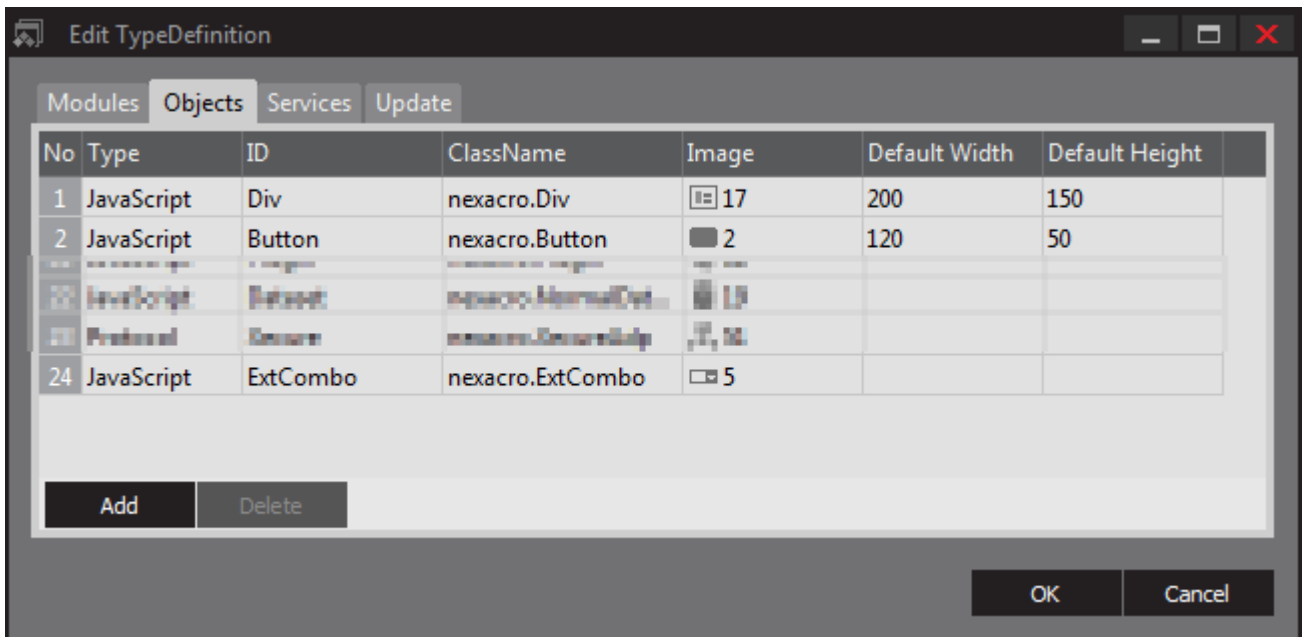
프로젝트 폴더에 있는 default\_typedef.xml 파일을 직접 수정하거나 [Edit TypeDefinition >Objects]창에서 해당 항목을 추가합니다.

id는 사용자가 임의로 지정할 수 있지만 classname 값은 사용자 컴포넌트 생성 시 지정된 클래스 이름을 지정해야 합니다. 클래스 이름은 사용자 컴포넌트와 함께 제공되는 가이드 문서를 참고하거나 info 파일에 있는 Object id 항목을 참조합니다.

모듈 등록 시 사용한 ExtCombo.info 파일에서는 Object id 값을 'nexacro.ExtCombo'로 설정했기 때문에 컴포넌트 등록 시 classname 값을 'nexacro.ExtCombo'로 설정합니다.

```
<?xml version='1.0' encoding='utf-8'?>
<MetaInfo version="1.0">
  <Object id="nexacro.ExtCombo">

    <!-- define extend component based nexacro.Combo -->
    <ObjectInfo
      inheritance="nexacro.Combo"
      typename="nexacro.ExtCombo"
```



설정된 컴포넌트의 기본 정보는 default\_typedef.xml 파일에 저장됩니다.

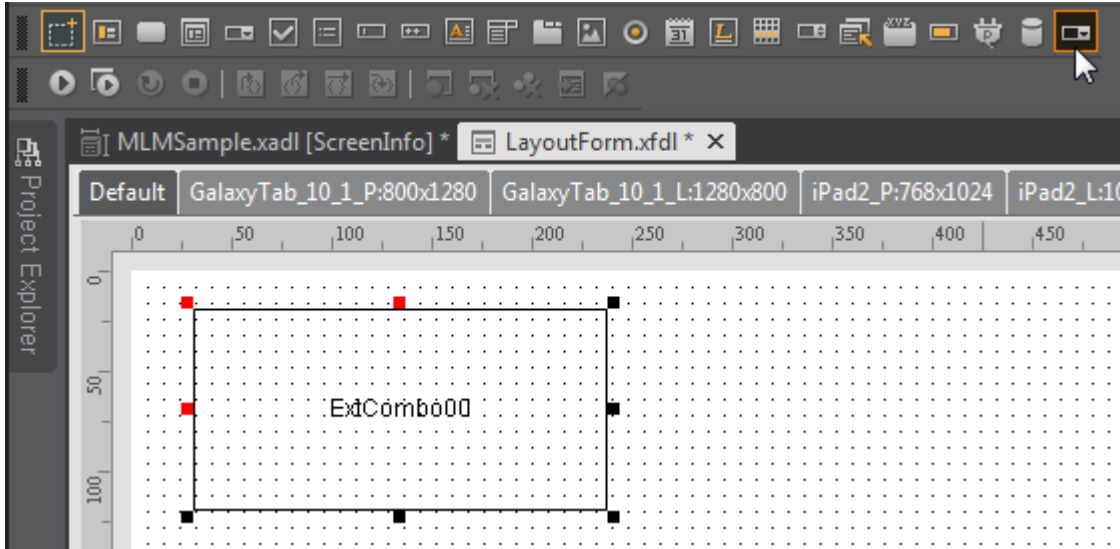
```
<Component type="JavaScript" id="ExtCombo" classname="nexacro.ExtCombo"/>
```

추가로 컴포넌트 배치 시 기본 설정되는 높이, 너비, 아이콘에 대한 정보는 프로젝트 설정 파일(\*.xprj)에 저장됩니다.

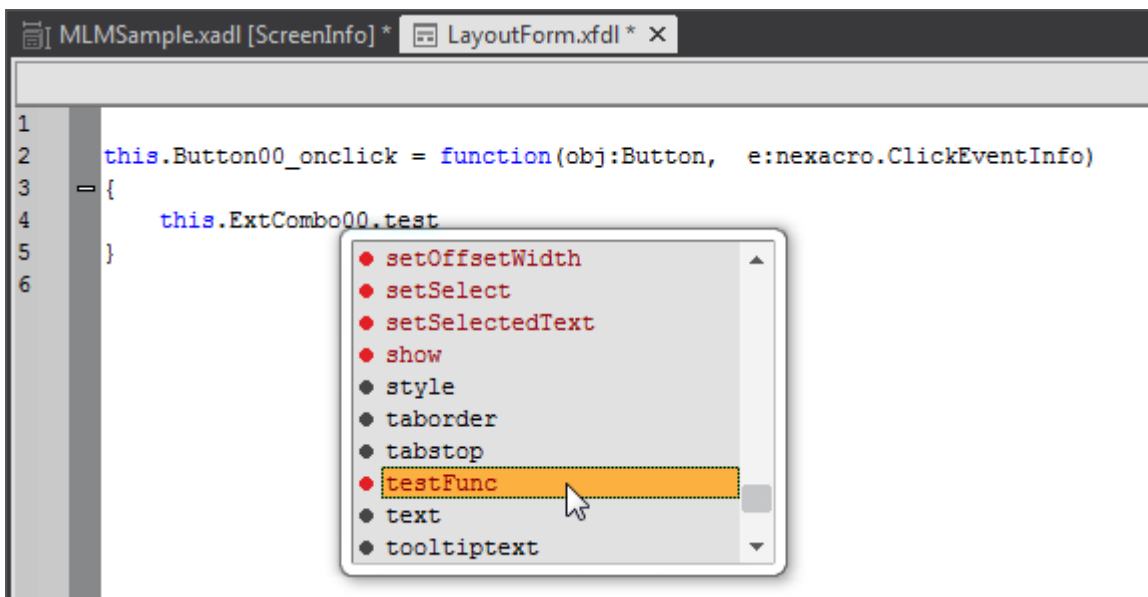
```
<?xml version="1.0" encoding="utf-8"?>
<Project active_adl="dd" version="1.2">
  <TypeDefinition url="default_typedef.xml"/>
  <GlobalVariables url="globalvars.xml"/>
  <ADL id="dd" url="dd.xadl"/>
  <ComponentInfoGroup>
    <ComponentInfo name="ExtCombo" defaultwidth="100" defaultheight="20" image="5"/>
  </ComponentInfoGroup>
```

## 컴포넌트 사용

기본 컴포넌트를 사용하는 것과 같이 툴바에 있는 컴포넌트 아이콘을 선택하고 폼에 적절한 크기로 배치합니다.



설정창에서 필요한 속성을 추가하거나 이벤트를 정의할 수 있으며 스크립트 창에서는 지정된 속성이나 메소드에 대해 자동완성 기능을 제공합니다.



# 찾아보기

---

## A

ADL(Application Definition Language), [17](#)

## G

Global Variable, [22](#)

## I

ID 선택자(ID Selector), [55](#)  
InitValue, [96](#)

## S

ScreenInfo, [91](#)

## T

Type Definition, [20](#)

## X

X-API, [27](#)  
XML Format, [28](#)

## ㄱ

그룹핑(Grouping), [55](#)

## L

눈금자(Dot Grid), [82](#)

## 人

스타일 편집 창, [62](#)

## O

유사클래스(Pseudo-classes), [55](#)  
유효범위(Scope), [31](#)  
이벤트 핸들러, [37](#)

## ㅈ

자식선택자(Child Selector), [55](#)  
좌표계, [78](#)

## ㅋ

클래스선택자(Class Selector), [54](#)

## ㅌ

타입선택자(Type Selector), [54](#)  
트래커, [81](#)

## ㅍ

포지션 에디터, [85](#)