고급 개발 가이드

17.0.0.1000

nexacro PLATFORM

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.

발행처 | (주)투비소프트
발행일 | 2018/07/24
주소 | (06083) 서울시 강남구 봉은사로 617 인탑스빌딩 2-5층
전화 | 02-2140-7700
홈페이지 | www.tobesoft.com

고객지원센터 | support.tobesoft.co.kr
제품기술문의 | 1588-7895 (오전 10시부터 오후 5시까지)

변경 이력

버전	변경일	내용
17.0.0.100	2017-10-13	17.0.0.100 공개로 전환
17.0.0.100.2	2017-11-01	추가 모듈에서 window 오브젝트 사용 시 제약 항목 추가
17.0.0.500	2018-02-22	프로젝트 내 파일 구조 일부 변경 내용을 반영했습니다. 프로젝트 및 파일 구조
17.0.0.1000	2018-07-23	사용자 컴포넌트 항목 재작성

차례

파트 I.

고급 활용

1.

프로토콜 어댑터

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

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

1.1 프로토콜 어댑터

프로토콜 어댑터를 사용하기 위해서는 모듈을 작성하고 넥사크로 스튜디오에 작성된 모듈을 등록하고 프로토콜, 서비스를 추가해야 합니다.

1.1.1 모듈 JSON, JS 파일

TypeDefinition에 등록할 모듈을 생성합니다. 모듈로 추가할 JSON파일과 JS파일을 작성합니다. 작성될 파일과 폴더는 아래와 같습니다.

[Base Library Path]	component	Xecure
	Xecure.json	Xecure.js

아래와 같은 형태로 JSON 파일을 작성합니다. 파일명은 "Xecure.json"으로 저장합니다. 'scripts' 항목에 지정된 자바스크립트 경로는 [Base Library Path > component] 폴더 아래 해당 폴더와 스크립트 파일이 있어야 합니다.

```
{
    "name": "Xecure",
    "version": "17.0.0.1",
    "description": "nexacro platform XecureAdp Protocol Library",
```

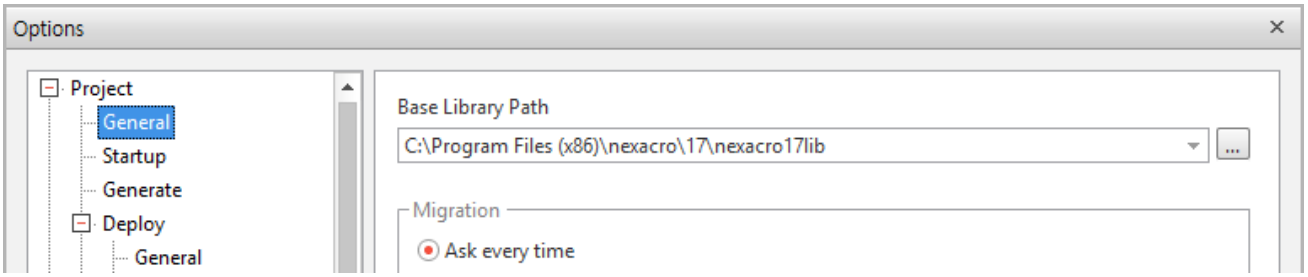
```
    "license": "",
    "scripts": [
        "Xecure/Xecure.js",
    ]
}
//@ sourceURL=Xecure.json
```

아래와 같은 형태로 JS 파일을 작성합니다. [Base Library Path > component] 폴더 아래 "Xecure"라는 폴더를 만들고 파일명은 "Xecure.js"로 저장합니다. 여기에서는 기본적인 동작 방식만을 설명합니다.

```
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");
    };
```

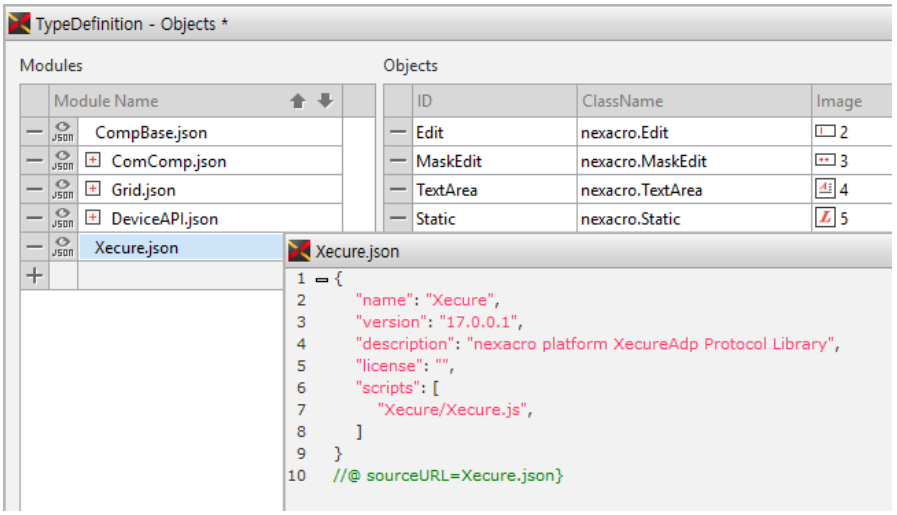
```
    _pXecureAdp.finalize = function ()
    {
     trace("_pXecureAdp.finalize");
    };

    delete _pXecureAdp;
}
```

등록할 모듈은 넥사크로 스튜디오에서 Base Library Path로 지정된 경로 아래 [component] 폴더 하위 경로에 파일이 있어야 합니다.
기본 설정은 넥사크로플랫폼 설치 경로 아래 nexacro17lib 폴더입니다.
Base Library Path는 넥사크로 스튜디오 메뉴 [Tools >Options > Project > General > Base Library Path]에서 다른 경로로 변경할 수 있습니다.



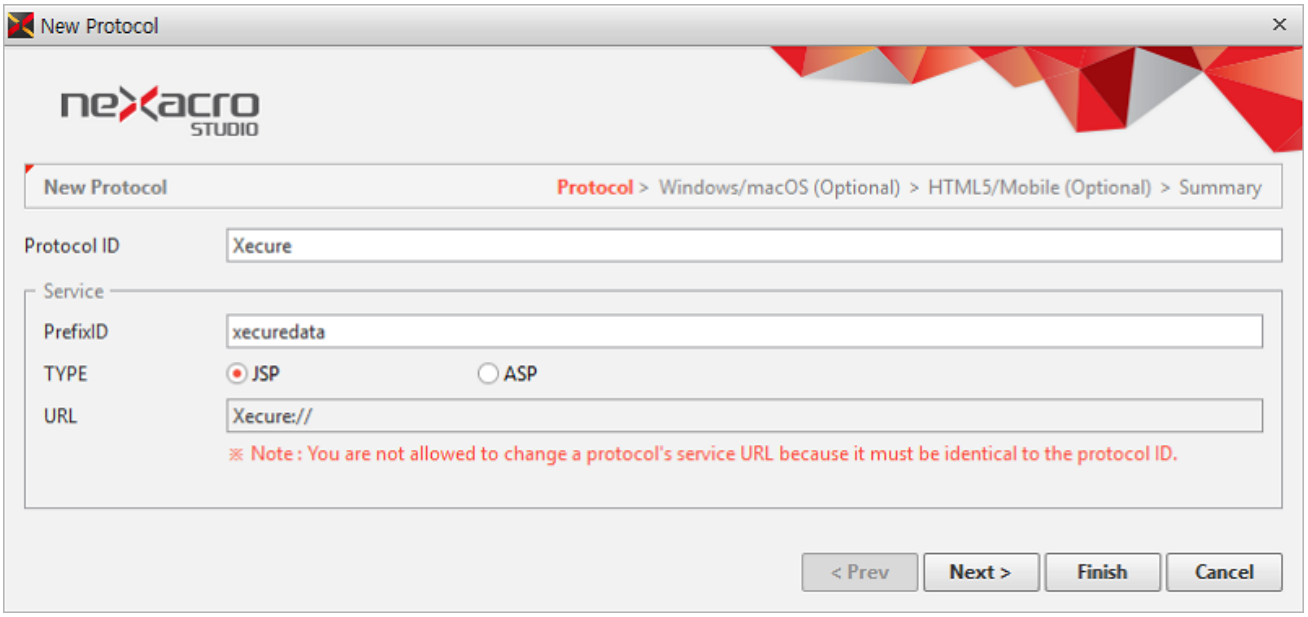
1.1.2 모듈 등록

넥사크로 스튜디오 [Project Explorer]에서 TypeDefinition 아래 Objects 항목을 더블 클릭합니다. [Modules] 목록 아래에 있는 "+" 버튼을 클릭하면 생성된 모듈 파일 중 JSON 파일을 선택할 수 있습니다. "Xecure.json" 파일을 선택하고 등록합니다.

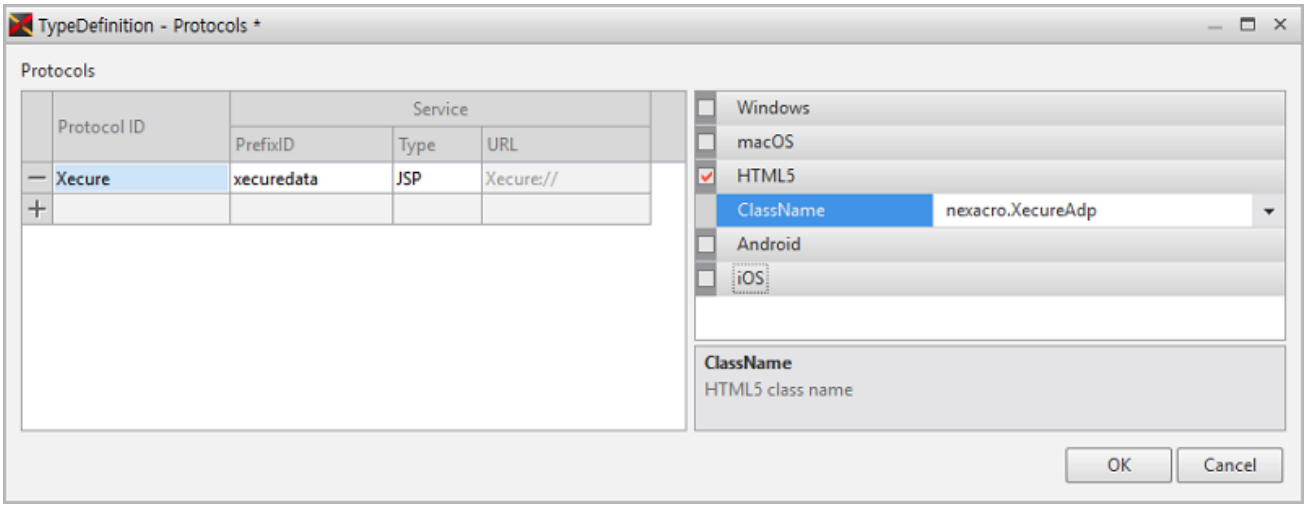


1.1.3 프로토콜, 서비스 등록

넥사크로 스튜디오 [Project Explorer]에서 TypeDefinition 아래 Protocols 항목을 더블 클릭합니다. "Protocol ID "와 서비스로 등록할 "PrefixID"를 입력합니다.



Finish 버튼을 클릭하고 HTML5 항목에 ClassName을 입력합니다.



> 프로토콜에 연결되어 등록한 서비스는 TypeDefinition 아래 Services 항목에서는 보이지 않습니다.

1.1.4 서비스 호출

등록한 서비스를 호출하게 되면 해당 프로토콜을 거쳐 통신이 진행됩니다.

```
nexacro.getApplication().transaction("MyService01", "xecuredata::localhost/test.jsp", "dsIn=
dsIn:A","dsOut=dsIn","a=b","fnCallback");
```

프로토콜로 등록된 모듈 내에 정의된 인터페이스를 해당 시점에 맞게 호출하는 방식으로 동작합니다. 모듈 내부의 동작방식은 아래와 같습니다.

1. 등록된 프로토콜이 처음 호출되는 시점에 ProtocolAdp 오브젝트를 생성합니다. 해당 오브젝트는 애플리케이션이 종료되기 전까지 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");
};
```

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

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

3. 내부에서 실제 통신이 호출되기 전에 encrypt 함수가 호출됩니다. 통신이 발생하기 전에 전송할 데이터를 암호화하거나 원하는 형태로 변환할 수 있습니다.

```
_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.

4 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;
};
```

1.2 HTTP 헤더 설정

데이터 형식에 따라 HTTP 헤더를 설정해야 하는 경우에는 아래와 같이 모듈 JS 파일을 작성하고 서비스를 호출할 수 있습니다.

1.2.1 서비스 호출

HTTP 헤더 설정을 위해 JS 파일에서 달라지는 부분만 설명합니다.

1 등록된 프로토콜이 처음 호출되는 시점에 ProtocolAdp 오브젝트를 생성합니다. _createPrototype 함수 호출 시 파라미터 값으로 생성할 클래스명을 지정합니다.

```
var _pXecureHttp = nexacro._createPrototype(nexacro.ProtocolAdp, nexacro.XecureHttp);
nexacro.XecureHttp.prototype = _pXecureHttp;
```

2 HTTP 헤더값을 설정합니다. 원하는 헤더값의 id, value 값을 지정할 수 있습니다.

```
_pXecureHttp._httpheaders = [];
_pXecureHttp._httpheaders.push({ id: "X-Requested-With", value: "XMLHttpRequest"});
_pXecureHttp._httpheaders.push({ id: "Accept", value: "application/json, text/json,
```

```
*/*"});
_pXecureHttp._httpheaders.push({ id: "Content-Type", value: "application/json"});
```

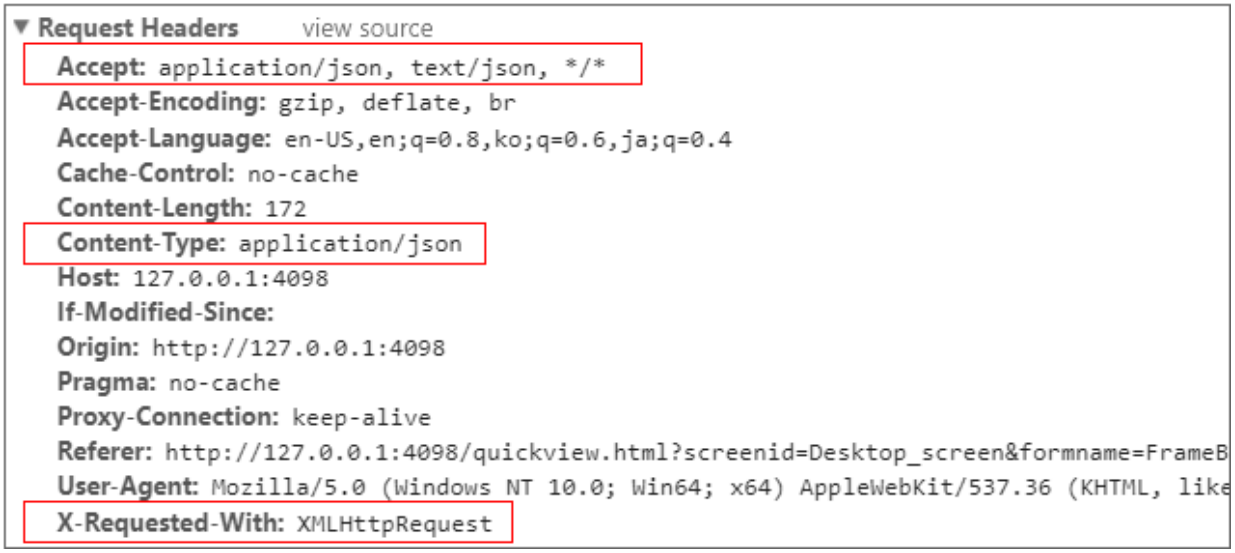
3 프로토콜 어댑터 버전 정보를 제공합니다. 이 함수가 없으면 "1.0"으로 인식합니다. getHTTPHeader 함수를 사용하기 위해서는 version 함수를 지정해야 하며 "1.1" 이상의 값을 가져야 합니다. version 함수가 없거나 값이 "1.1"보다 작으면 getHTTPHeader() 함수를 처리하지 않습니다.

```
_pXecureHttp.version = function ()
{
    return "1.1";
};
```

4 HTTP 통신 헤더 정보를 제공합니다. _httpheaders는 HTTP 헤더 정보를 담고있는 배열 오브젝트입니다.

```
_pXecureHttp.getHTTPHeader = function ()
{
    return this._httpheaders;
};
```

5 서비스 호출 후 HTTP 헤더를 확인해보면 설정한 값이 적용된 것을 확인할 수 있습니다.



1.3 PluginElement

데이터 처리 시 외부 플러그인을 연동해야 하는 경우 PluginElement를 사용할 수 있습니다. 외부 플러그인 적용은 해당 솔루션에서 제공하는 규칙에 따라 추가적인 코드가 필요할 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.hsecure.co.kr/solutions/solution_list.php?cate_cd=010103

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

1.3.1 서비스 호출

PluginElement를 사용할 때 프로토콜 어댑터를 사용하는 방식은 같고 프로토콜로 등록되는 JS 코드 일부만 변경합니다.

1. 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();
};
```

2. 통신 시 사용할 프로토콜은 getUsingProtocol 함수에서 반환합니다. 기본값으로 설정된 프로토콜은 http입니

다.

```
_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;
};
```

3 오브젝트가 삭제되는 시점에 finalize 함수가 호출됩니다. 애플리케이션이 내려가는 시점에 호출되며 생성된 플러그인을 제거해주는 코드를 처리해주어야 합니다.

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

2.

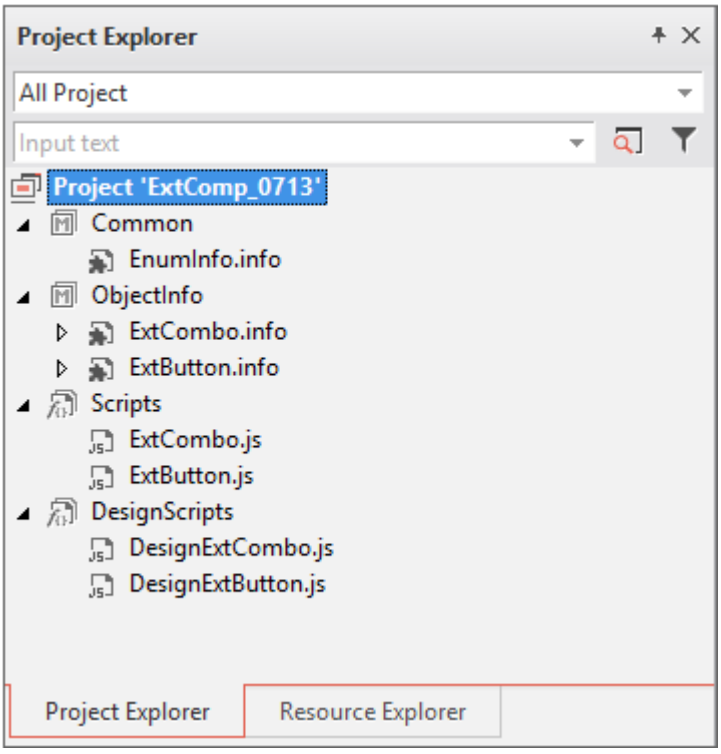
사용자 컴포넌트

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.

2.1 사용자 컴포넌트 구성

사용자 컴포넌트는 넥사크로 스튜디오의 모듈 프로젝트를 통해 생성할 수 있습니다. 하나의 모듈 프로젝트 안에는 1개 이상의 사용자 컴포넌트와 리소스를 포함할 수 있습니다.



모듈 프로젝트는 [Deploy] 메뉴를 통해 배포할 수 있는 파일로 생성됩니다. 배포된 모듈은 넥사크로 스튜디오에서 설치할 수 있으며 설치된 파일은 프로젝트 경로 아래 "_extlib_" 폴더 하위에서 확인할 수 있습니다. 설치되는 파일은 아래와 같습니다.

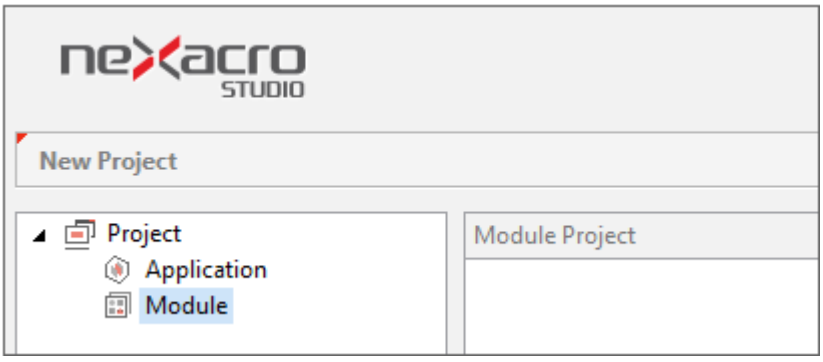
> 모듈 파일 (*.json)
> 오브젝트 정보 파일 (*.info)
> 컴포넌트 스크립트 파일, 디자인 스크립트 파일 (*.js)

2.2 사용자 컴포넌트 생성

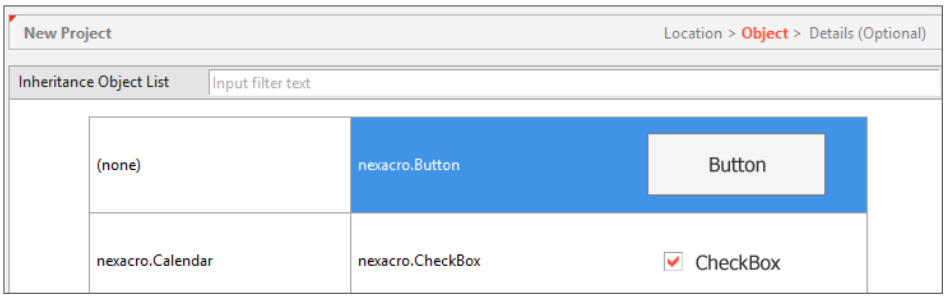
간단하게 사용자 컴포넌트를 생성하는 방법을 살펴봅니다. Button 컴포넌트를 상속받으며 속성, 메소드, 이벤트를 추가해봅니다.

2.2.1 모듈 프로젝트 생성하기

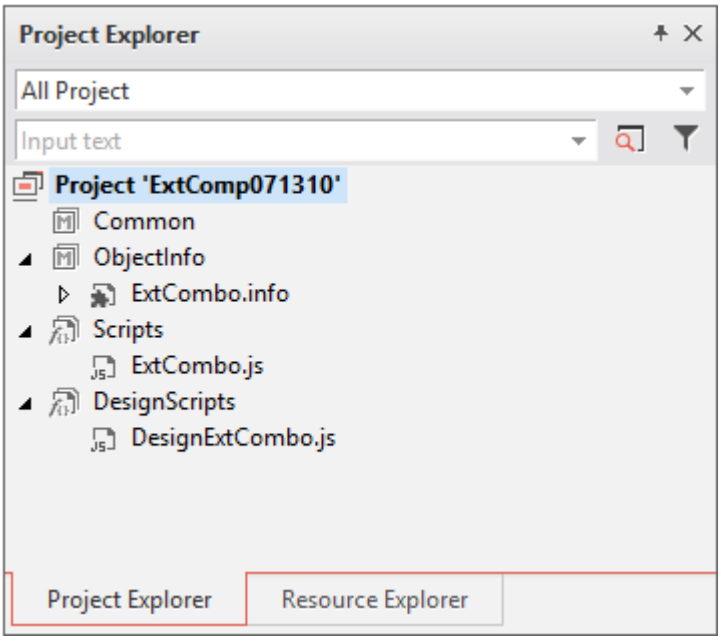
메뉴[File > New > Project]를 선택하고 왼쪽 프로젝트 항목 중 "Module"를 선택합니다.



프로젝트명과 경로를 입력하고 [Next] 버튼을 클릭하면 상속받을 오브젝트를 지정합니다. "none"을 선택하면 다른 오브젝트를 상속받지 않고 새롭게 사용자 컴포넌트를 생성할 수 있습니다. 예제에서는 Button 컴포넌트를 선택합니다.



Object ID를 입력하고 [Finish] 버튼을 클릭하면 모듈 프로젝트가 생성됩니다. 생성된 프로젝트는 아래와 같습니다.

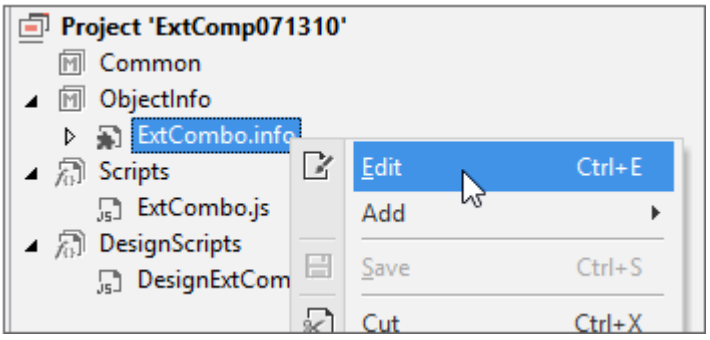


예제에서는 ExtCombo 라는 하나의 사용자 컴포넌트만 생성합니다. 하지만 모듈 프로젝트 안에는 1개 이상의 컴포넌트를 포함할 수 있습니다. 메뉴[File > New > Object]를 선택하면 바로 상속받을 오브젝트를 선택하는 화면으로 이동합니다.

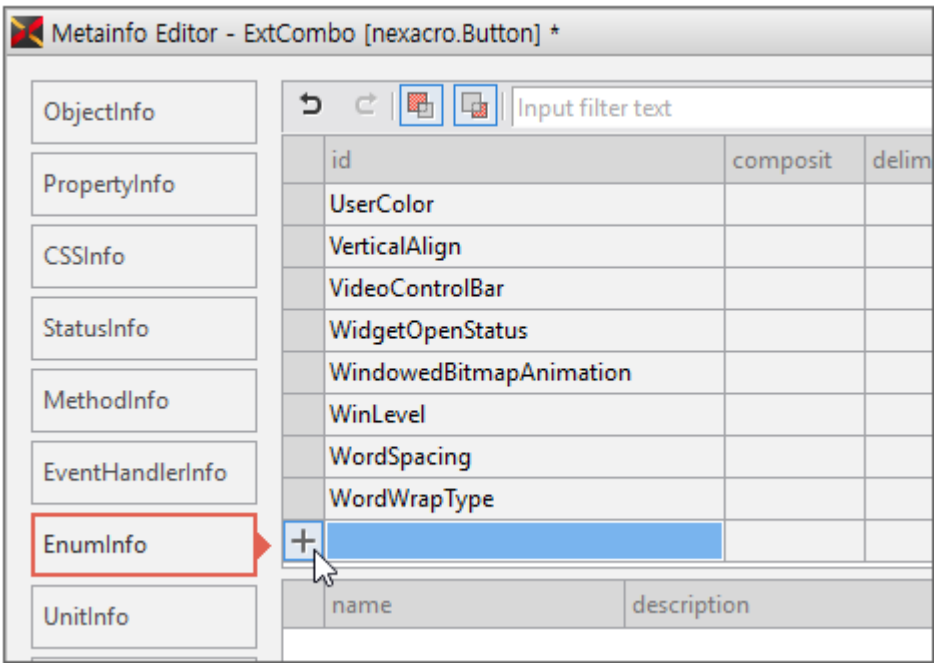
2.2.2 공통 정보 추가하기

생성된 프로젝트의 "Common" 항목 아래에는 아무런 파일도 생성되지 않았습니다. 컴포넌트와 관련된 정보 중에서 EnumInfo, UnitInfo, RefreshInfo는 공통으로 사용할 수 있기 때문에 "Common" 항목으로 분류합니다. 예제에서는 추가할 속성에서 사용할 Enum 정보를 생성해봅니다.

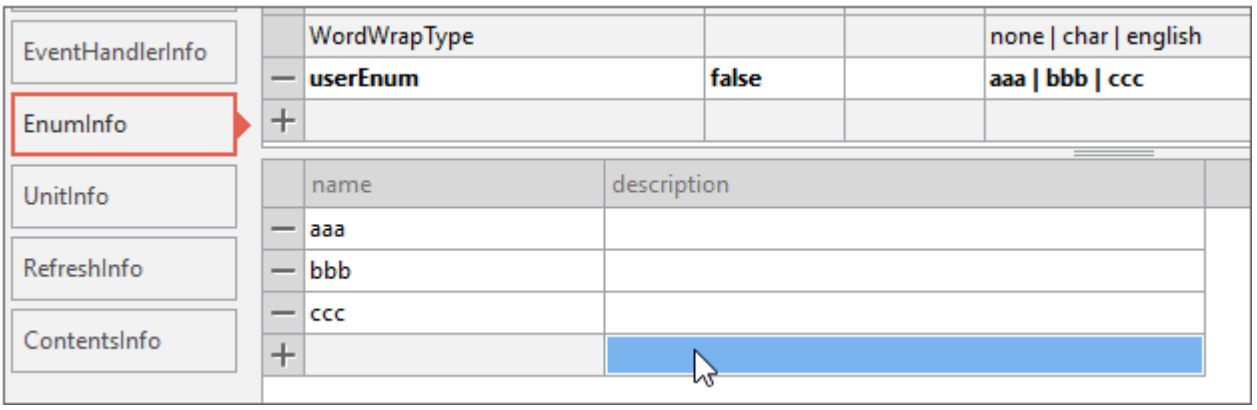
프로젝트 트리에서 "ObjectInfo > ExtCombo.info" 항목을 선택하고 [Edit] 메뉴를 선택합니다.



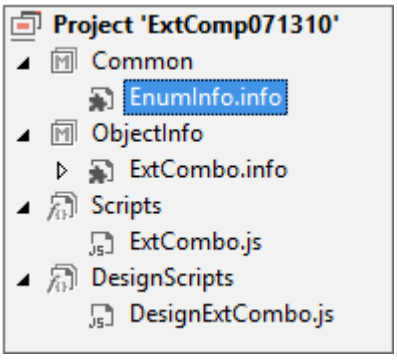
Enuminfo 항목을 선택하고 목록에서 [+] 버튼을 클릭합니다.



추가할 Enum 정보를 입력합니다. id는 userEnum으로 등록했습니다.

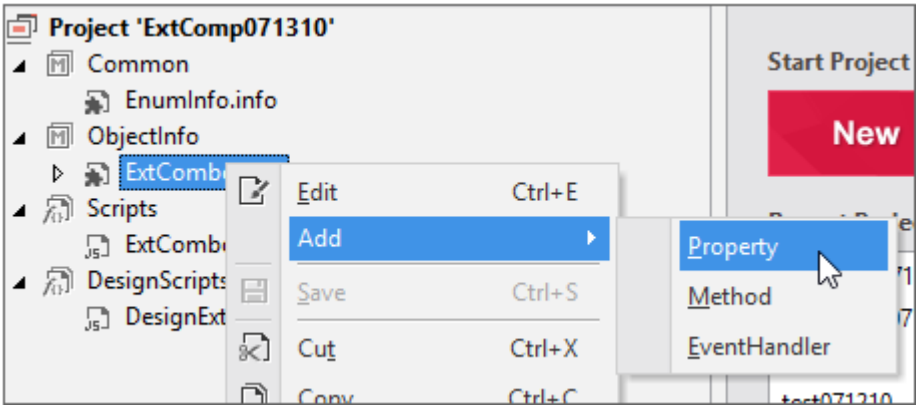


정보를 입력하고 [OK] 버튼을 클릭하면 프로젝트 트리의 "Common" 항목 아래에 EnumInfo.info 파일이 추가되는 것을 확인할 수 있습니다.

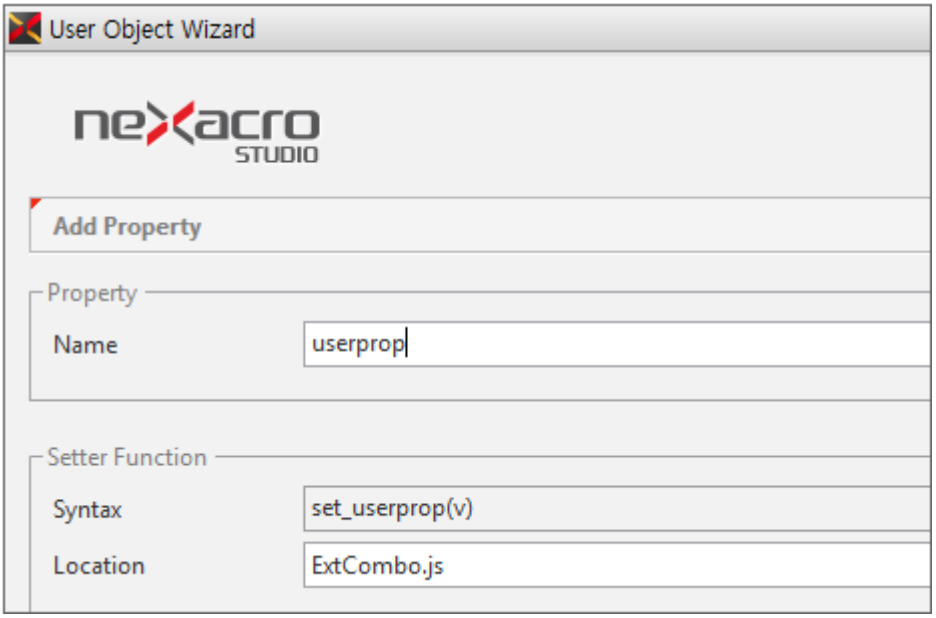


2.2.3 속성 추가하기

프로젝트 트리에서 "ObjectInfo > ExtCombo.info" 항목을 선택하고 [Edit] 메뉴를 선택하고 속성을 추가할 수도 있지만, [Edit] 메뉴 대신 [Add > Property] 항목을 선택해서 속성을 추가할 수도 있습니다.



추가할 속성명을 입력합니다. Syntax, Location 항목은 자동으로 설정됩니다.

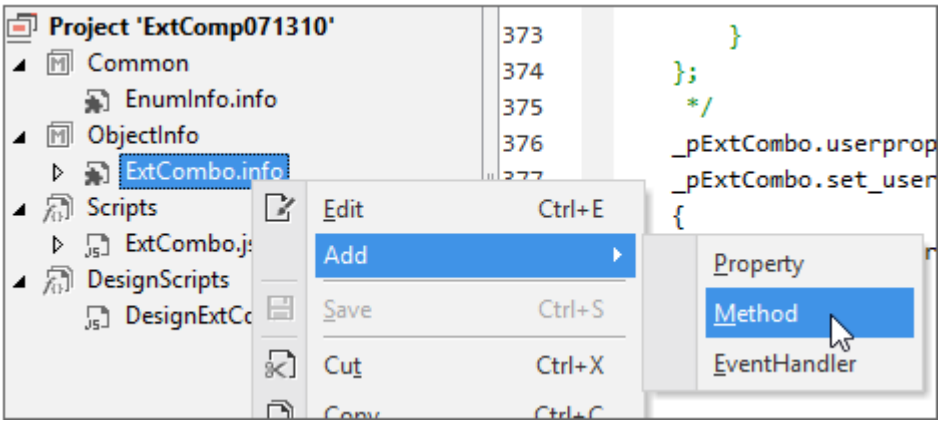


[Next] 버튼을 클릭하면 상세 설정을 지정할 수 있습니다. 예제에서는 group 항목값을 "Misc."로 지정하고 edittype 항목값을 "Enum"으로 지정합니다. enuminfo 항목에서 이전 단계에서 추가한 enum 정보인 "userEnum"을 선택합니다.

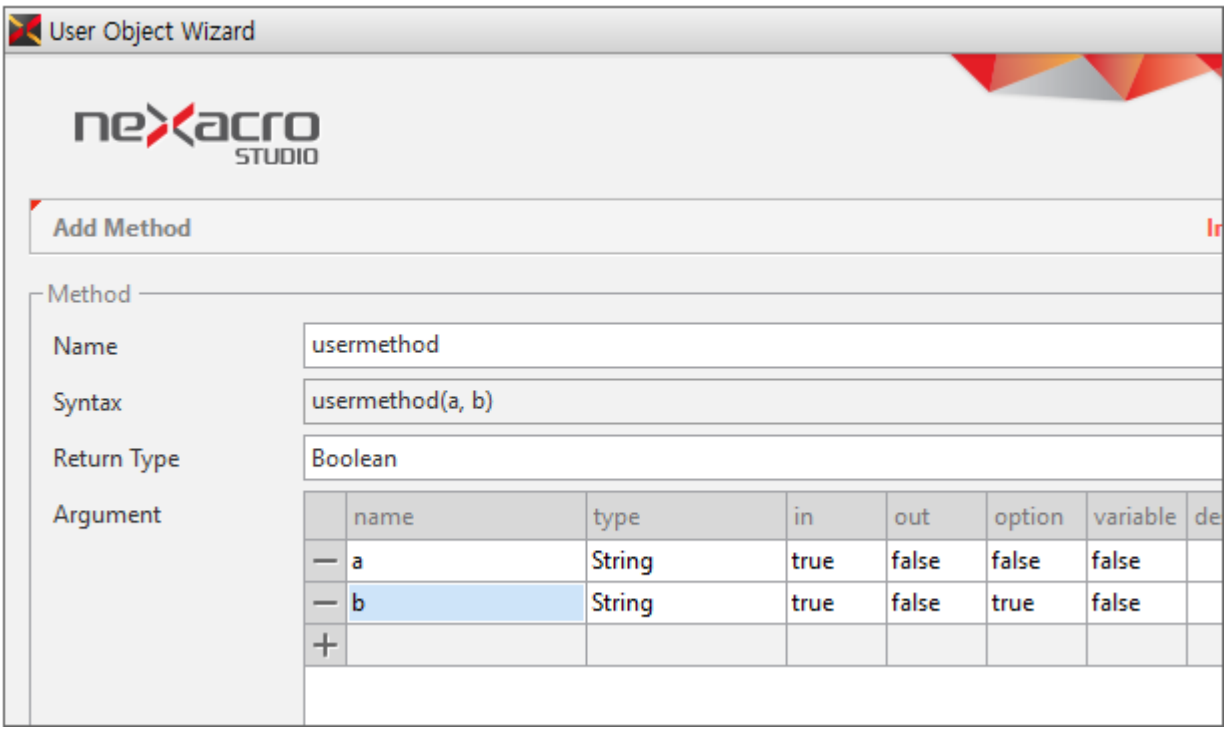
[Add > Property] 항목을 선택해서 속성을 추가한 경우에는 컴포넌트 스크립트 파일이 열리면서 set_ 함수가 추가됩니다. 함수 내 필요한 동작을 정의할 수 있습니다. 추가한 속성이 기본값을 가져야 하는 경우에는 기본값을 정의하는 코드를 추가합니다.

2.2.4 메소드 추가하기

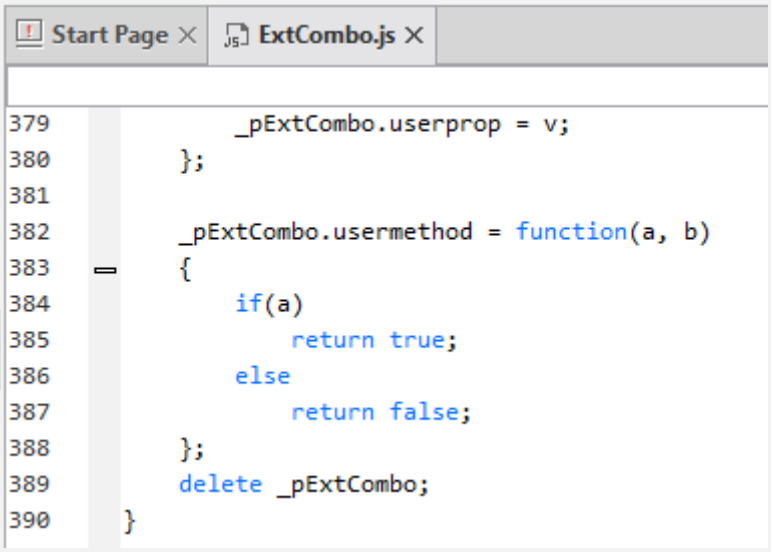
프로젝트 트리에서 "ObjectInfo > ExtCombo.info" 항목을 선택하고 [Edit] 메뉴를 선택하고 메소드를 추가할 수도 있지만, [Edit] 메뉴 대신 [Add > Method] 항목을 선택해서 메소드를 추가할 수도 있습니다.



추가할 메소드명을 입력하고 반환되는 값이 있는 경우 Return Type 항목을 선택하고 매개변수값의 형식을 지정합니다.

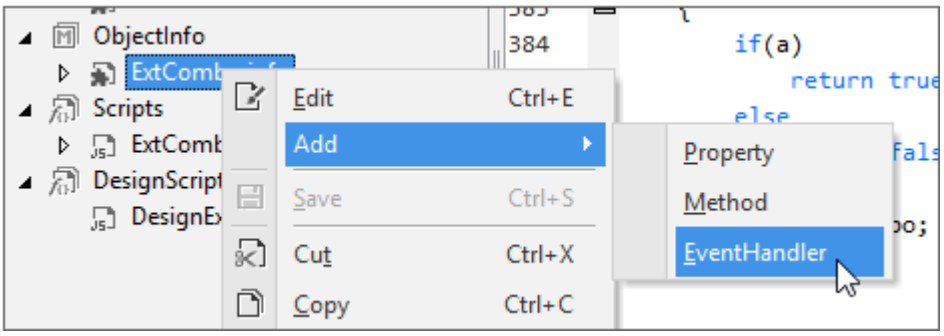


[Add > Method] 항목을 선택해서 메소드를 추가한 경우에는 컴포넌트 스크립트 파일이 열리면서 메소드 함수가 추가됩니다. 함수 내 필요한 동작을 정의할 수 있습니다.

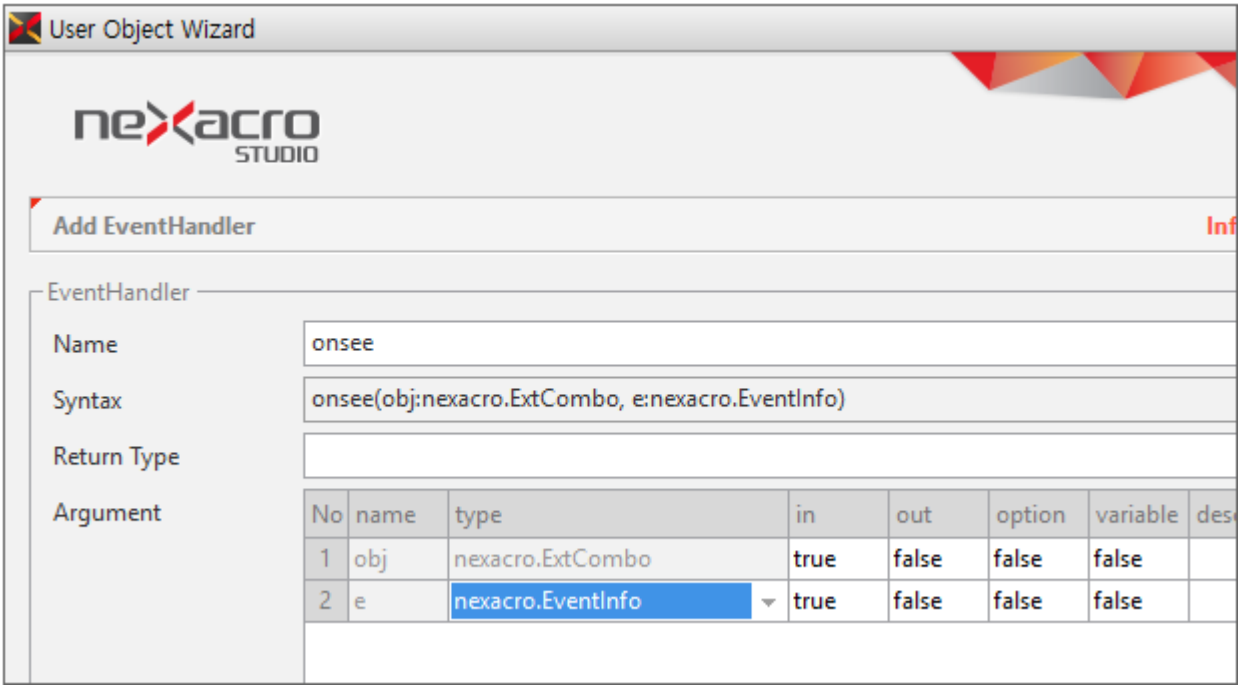


2.2.5 이벤트 추가하기

프로젝트 트리에서 "ObjectInfo > ExtCombo.info" 항목을 선택하고 [Edit] 메뉴를 선택하고 메소드를 추가할 수도 있지만, [Edit] 메뉴 대신 [Add > EventHandler] 항목을 선택해서 메소드를 추가할 수도 있습니다.

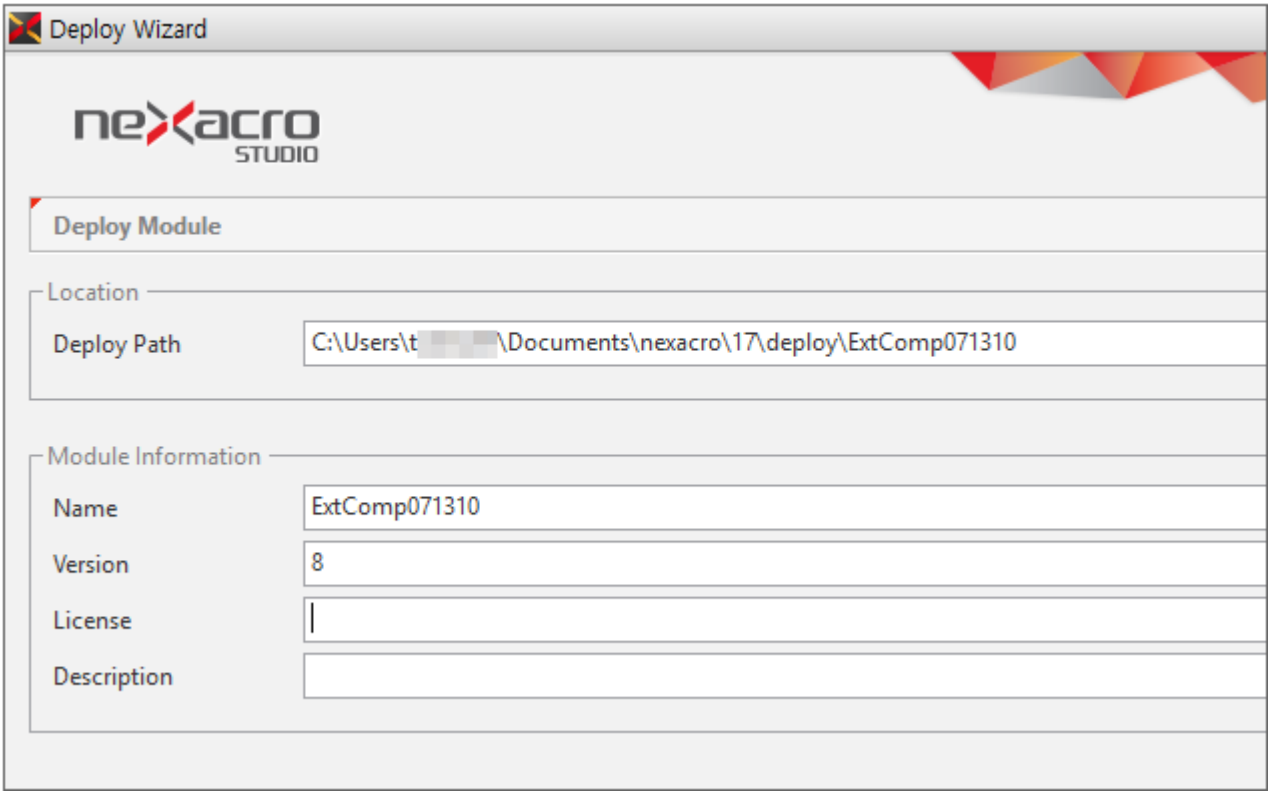


추가할 이벤트명을 입력하고 EventInfo 오브젝트를 선택합니다. 반환되는 값이 있는 경우 Return Type 항목을 선택하고 필요한 경우 EventInfo 오브젝트를 새로 생성한 후 생성된 오브젝트를 선택할 수 있습니다.



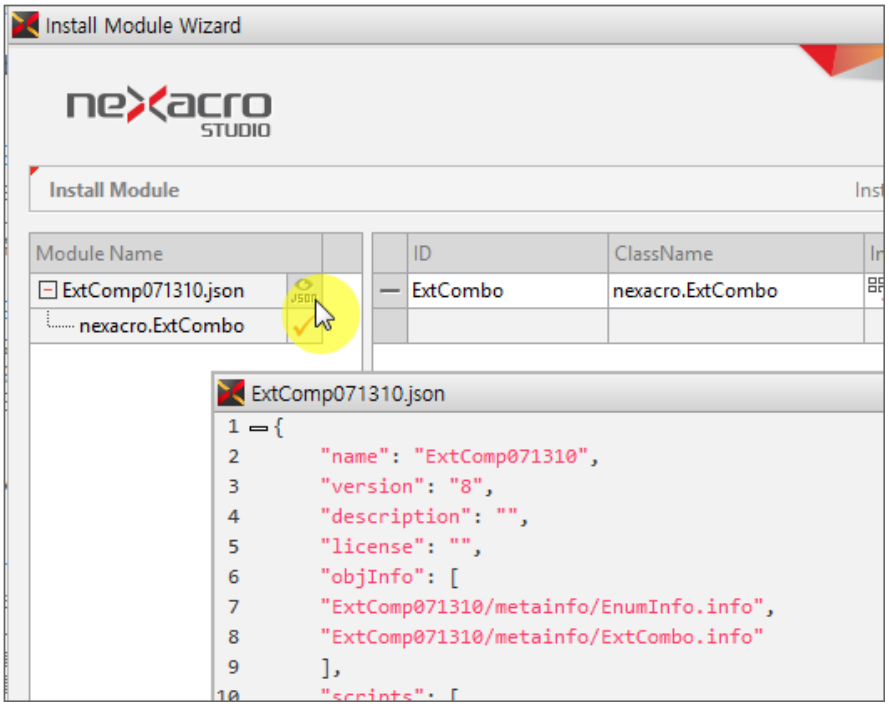
2.3 사용자 컴포넌트 배포 및 설치

컴포넌트는 모듈 형태로 배포합니다. 메뉴[Deploy > Module Package]에서 Deploy path를 지정하고 배포할 모듈명과 버전 정보를 입력하면 배포 파일을 생성합니다.

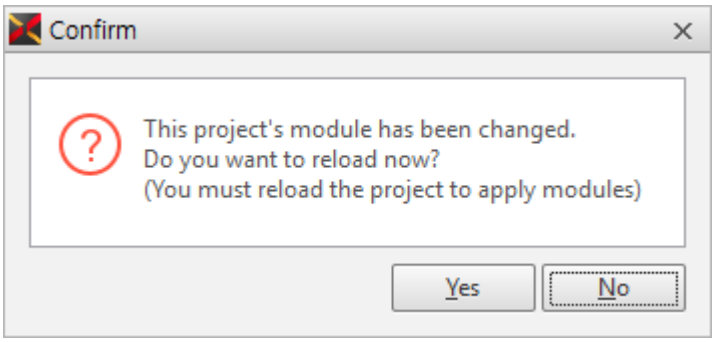


생성된 모듈 파일은 모듈명 뒤에 xmodule이라는 확장자가 붙습니다.

배포된 모듈 파일은 애플리케이션 프로젝트에서 메뉴[File > Import Module]에서 설치할 수 있습니다. 설치 단계에서 json 파일 옆에 있는 버튼을 클릭하면 버전 정보 등을 확인할 수 있습니다.

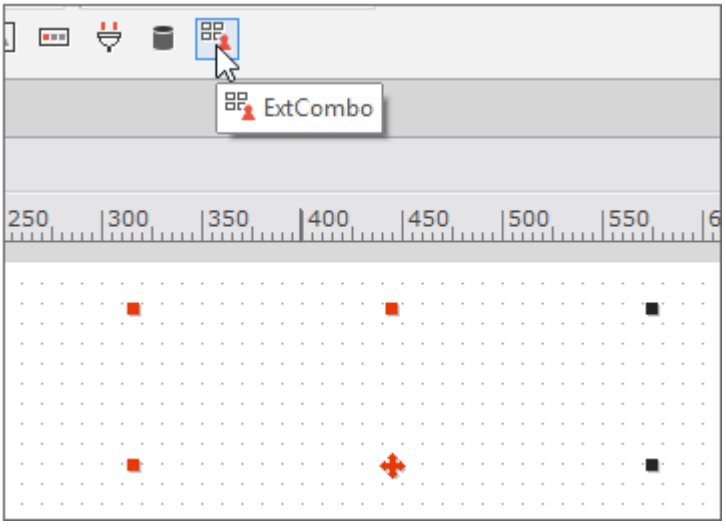


모듈 설치 후에는 프로젝트를 다시 로딩합니다.



2.4 사용자 컴포넌트 사용

추가한 사용자 컴포넌트는 기본 컴포넌트와 마찬가지로 오브젝트 툴바에 표시되며 선택 후 화면에 배치할 수 있습니다.

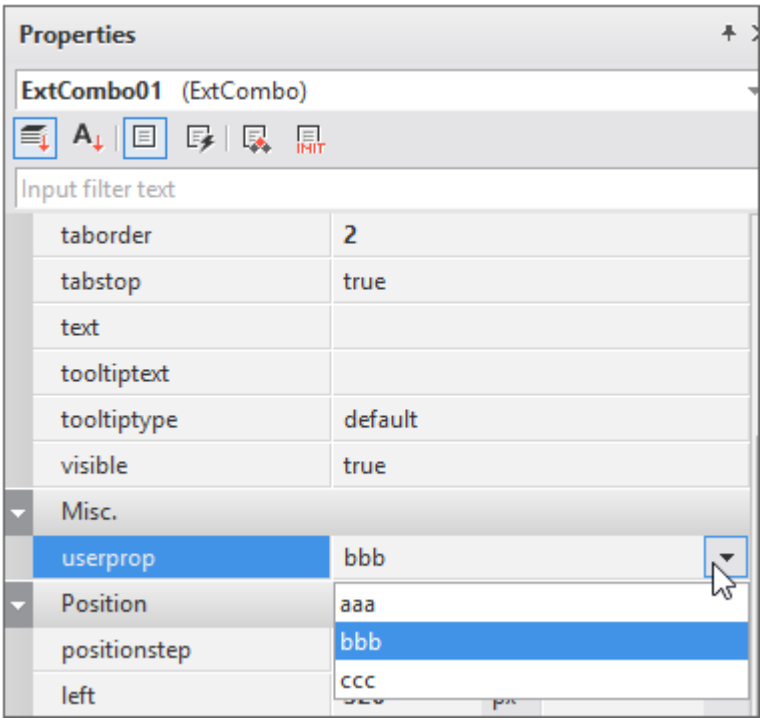


설정창에서 필요한 속성을 추가하거나 이벤트를 정의할 수 있으며 스크립트 창에서는 지정된 속성이나 메소드에 대해 자동완성 기능을 제공합니다.

```
1
2   this.Button00_onclick = function(obj:nexacro.Button,e:nexacro.ClickEventInfo)
3   {
4       trace(this.ExtCombo00.user
5       this.ExtCombo00.set_use
6       trace(this.ExtCombo00.u
7       trace(this.ExtCombo00.u
8   };
9
10  this.ExtCombo00_onsee = function(obj:nexacro.ExtCombo, e:nexacro.EventInfo)
11  {
12
13  };
14
```

usermethod
userprop
usermethod(a, b)

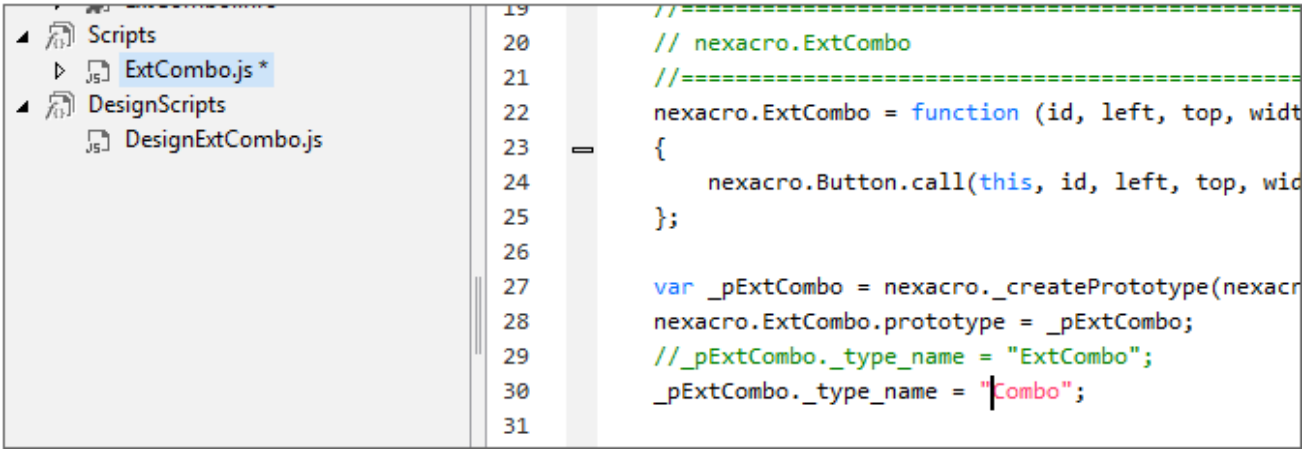
속성창에서 추가된 속성항목 선택 시 공통 정보에서 추가한 Enum 목록을 확인할 수 있습니다.



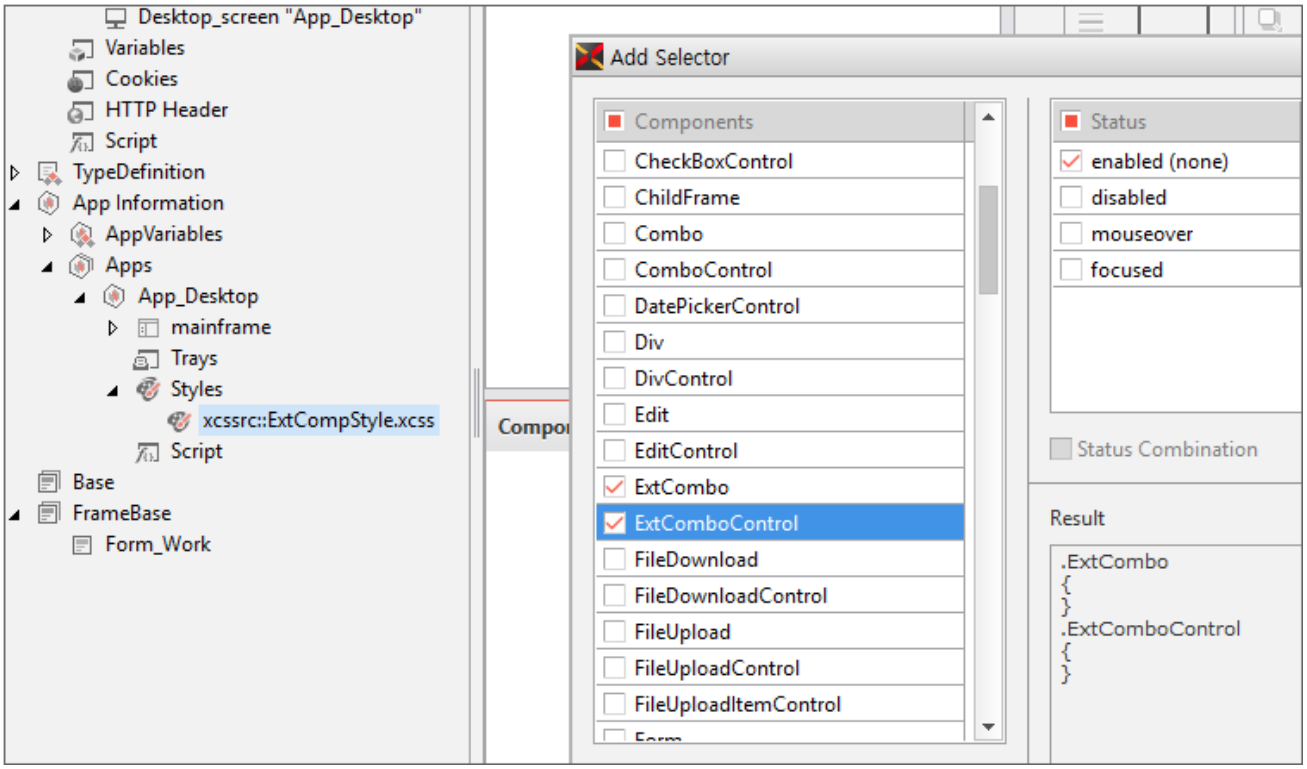
2.4.1 사용자 컴포넌트 스타일

하지만 화면에 배치했을 때 컴포넌트 UI가 화면에 표시되지 않습니다. 사용자 컴포넌트에 대한 스타일이 정의되지 않았기 때문에 화면에 컴포넌트 UI가 표시되지 않는 것입니다. 사용자 컴포넌트의 스타일은 아래와 같이 처리할 수 있습니다.

추가된 사용자 컴포넌트가 상속받은 오브젝트의 UI에 영향을 미치지 않는 경우에는 사용자 컴포넌트 스크립트에서 _type_name 값을 상속받은 오브젝트로 변경합니다. 아래의 경우 Combo 컴포넌트의 스타일을 그대로 사용합니다.



그렇지 않고 사용자 컴포넌트의 스타일이 상속받은 오브젝트와 달라지는 경우라면 사용자 컴포넌트에 맞는 스타일 속성을 정의해주어야 합니다. 테마 파일을 변경하거나 사용자 컴포넌트에 적용할 XCSS 파일을 생성해 사용할 수 있습니다.



파트 II.

동작 원리

3.

프로젝트 및 파일 구조

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

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

3.1 프로젝트 생성

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

일반적으로 프로젝트 생성 시 만들어지는 파일은 아래와 같습니다.

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

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

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

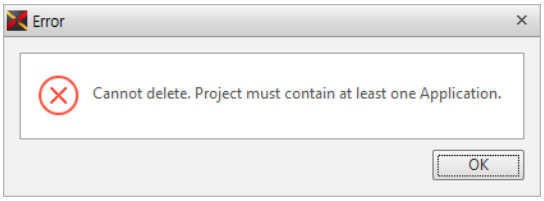
3.1.1 프로젝트 설정 파일 (XPRJ)

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

```
<?xml version="1.0" encoding="utf-8"?>
<Project type="application" version="2.0">
  <EnvironmentDefinition url="environment.xml"/>
  <TypeDefinition url="typedefinition.xml"/>
  <AppVariables url="appvariables.xml"/>
  <AppInfos>
        <AppInfo url="[file name].xadl"/>
        <AppInfo url="[file name].xadl"/>
  </AppInfos>
</Project>
```

3.1.2 애플리케이션 설정 파일 (XADL)

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



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

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

3.1.3 Type Definition

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

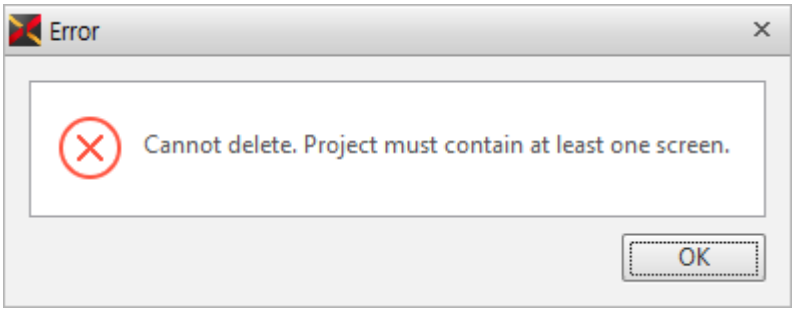
	항목	설명
Modules	모듈 정보	자바스크립트로 개발된 모듈 정보를 담고 있습니다. - 자체 개발된 모듈을 추가할 수 있습니다. - js 또는 json 파일 형태로 등록할 수 있습니다. 각 파일 간에 의존성을 가질 때에는 순서에 맞게 등록해야 합니다.
Components	오브젝트 정보	애플리케이션 실행에 필요한 오브젝트 정보를 담고 있습니다. - 지정된 오브젝트가 배포에 포함됩니다. - 각 오브젝트는 모듈을 기반으로 지정됩니다.
Services	서비스 그룹	프로젝트 규모에 따라 내부적으로 서비스 그룹을 정의합니다. - 각 서비스 그룹은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

```
<?xml version="1.0" encoding="utf-8"?>
<TypeDefinition version="2.3">
  <Modules>
    <Module url="CompBase.json"/>
    <Module url="ComComp.json"/>
    <Module url="Grid.json"/>
    <Module url="DeviceAPI.json"/>
  </Modules>
  <Components>
```

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

3.1.4 Environment

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




```
<?xml version="1.0" encoding="utf-8"?>
<ENV version="2.1">
  <Environment themeid="theme::default">
    <ScreenDefinition>
      <Screen id="[Screen id]" type="[type property value]"/>
```

```
    </ScreenDefinition>
    <Variables/>
    <Cookies/>
  </Environment>
</ENV>
```

3.1.5 AppVariables

App에서 공통으로 사용할 수 있는 Dataset, Variable 정보를 관리합니다.

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

3.2 리소스

3.2.1 initValue

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

```
<?xml version="1.0" encoding="utf-8"?>
<GlobalPropInitvalueDefinition>
    <nexacro.Button>
```

```
        <initvalueid id="[initvalueid ]" ...  />
    </nexacro.Button>
    <nexacro.Edit>
        <initvalueid id="[initvalueid ]" ... />
    </nexacro.Edit>
</GlobalPropInitvalueDefinition>
```

3.2.2 테마 (XTHEME)

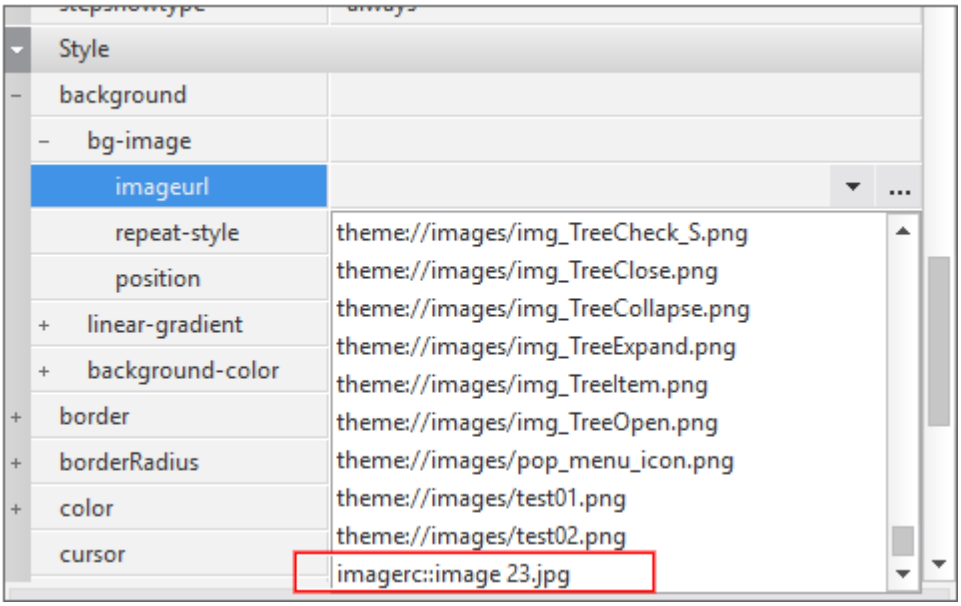
프로젝트 생성 후 넥사크로 스튜디오 [Resource Explorer] 창을 확인해보면 기본 설치 테마가 포함된 [Nexacro Theme] 항목과 새로운 테마를 추가할 수 있는 [Theme] 항목이 표시됩니다. [Theme] 항목에는 기존 테마를 복사해 넣거나 새 테마를 만들 수 있습니다.

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

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

3.2.3 ImageResource

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



3.2.4 UserFont

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

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

3.3 기타

3.3.1 Form (XFDL)

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

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

```
<?xml version="1.0" encoding="utf-8"?>
<FDL version="2.0">
    <Form>
```

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

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

3.3.2 스크립트 (XJS)

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

작성된 스크립트는 Form에서 다음과 같이 참조할 수 있습니다.

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

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

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

3.3.3 스타일 설정 파일 (XCSS)

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

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

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

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

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

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

4.

애플리케이션 구동 시나리오

넥사크로플랫폼 애플리케이션이 실행되는 과정은 아래와 같습니다.

4.1 bootstrap

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

<table>
<tr><th></th><th>HTML5</th><th>Runtime</th></tr>
<tr><td>❶</td><td>index.html</td><td>start.json
start_android.json
start_ios.json</td></tr>
<tr><td>❷</td><td colspan="2">get bootstrap info</td></tr>
<tr><td>❸</td><td rowspan="2"></td><td>update engine</td></tr>
<tr><td>❹</td><td>update resource (data)</td></tr>
<tr><td>❺</td><td colspan="2">load framework files</td></tr>
<tr><td>❻</td><td colspan="2">load component modules</td></tr>
<tr><td>❼</td><td colspan="2">load application</td></tr>
</table>

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

4.2 애플리케이션 로딩

애플리케이션이 실행되기 위해 애플리케이션과 연결된 폼을 로딩하는 과정은 아래와 같습니다.

넥사크로플랫폼 14	넥사크로플랫폼 17
ADL 로딩	Environment 로딩
ADL 실행	Screen 결정
TypeDefinition 구문 분석(parsing)	Screen 환경 정보 로딩
서비스 등록	결정된 Screen의 Application
클래스 목록 저장	Application 생성
서브파일 다운로드 및 실행	Environment 실행
Application 초기화	테마파일 로드
기본 클래스 등록	Initvalue 파일 로드
Application 속성 설정	Application 로드
Mainframe 생성 및 초기화	Application 속성 설정
MainFrame 내 Frame 생성	Mainframe 생성 및 초기화
Form 로딩	MainFrame 내 Frame 생성
Application, MainFrame, Frame 이벤트핸들러 등록	Form 로딩
이벤트 발생: Application.onloadtypedefinition	Application, MainFrame, Frame 이벤트핸들러 등록
GlobalVariables 생성	이벤트 발생: Application.onloadingappvariables
이벤트 발생: Application.onloadingglobalvariables	메소드 호출: MainFrame.createComponent
메소드 호출: MainFrame.createComponent	메소드 호출: MainFrame.on_created
메소드 호출: MainFrame.on_created	이벤트 발생: Application.onload
이벤트 발생: Application.onload	이벤트 발생: Form.onload
이벤트 발생: Form.onload	이벤트 발생: Application.onloadforms

로딩 순서에 따라 아직 생성되지 않는 항목은 사용할 수 없습니다.
예를 들어 onloadtypedefinition 이벤트 함수 내에서 GlobalVariables에 접근하게 되면 undefined로 표시됩니다.

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

4.3 폼 로딩

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

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

서브파일은 include 방식으로 연결되는 스크립트 파일 등을 의미합니다.

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

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

5.

넥사크로플랫폼 스크립트 언어

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

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

5.1 유효범위(Scope)

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

5.1.1 로컬 (local)

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

```
var value0 = 0;
```

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

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

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

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

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

5.1.2 this

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

Form 내에서 변수는 아래와 같이 선언합니다.

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

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

```
this.formvalue = 4;

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

```
}
```

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

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

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

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

eval 함수로 Form을 접근하는 경우에도 this를 사용해야 합니다.

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

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

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

5.1.3 Global

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

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

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

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

    value = 4;
    var localvar = 5;
}


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

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

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

includecode.xjs

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

includecode.xjs.js

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

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

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

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

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

아래와 같이 유효범위 없이 지정된 함수는 Global로 처리됩니다.

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

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

transaction()과 같은 애플리케이션 또는 폼 메소드는 해당하는 유효범위를 지정해주어야 합니다.

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

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

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

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

5.1.4 Expr

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

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

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

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

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

```
Grid: comp
Dataset: dataset
Cell: this
```

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

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

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

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

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

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

5.1.5 lookup

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

lookup 메소드는 아래와 같은 형식으로 사용할 수 있습니다.

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

실제 코드에서는 아래와 같이 사용됩니다.

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

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

// this에서 상위로 검색해서 fn_onclick에 해당하는 함수를 이벤트 핸들러에서 처리
btn00.addEventHandlerLookup( "onclick", "fn_onclick", this );
```

5.2 이벤트 핸들러

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

5.2.1 메소드

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

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

이벤트를 처리할 함수의 유효범위에 따라 적절한 메소드를 선택해 사용합니다.

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

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

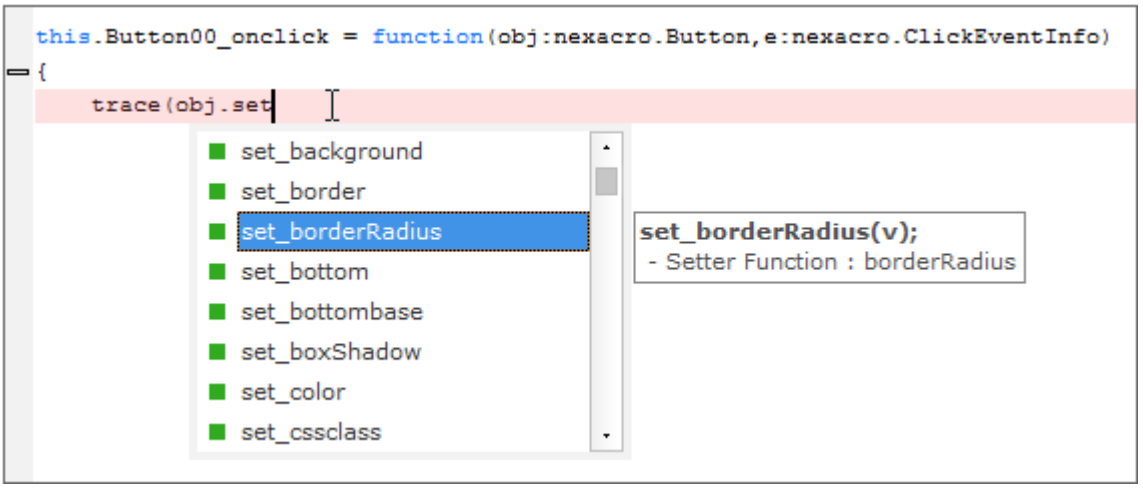
이벤트를 속성 창에서 생성하게 되면 아래와 같이 문자열로 설정됩니다.
<Button id="Button00" ... onclick="Button00_onclick"/>
해당 코드는 빌드 작업 후 변환된 자바스크립트 코드에서는 아래와 같이 this 지시자를 붙여서 생성됩니다.
this.Button00.addEventHandler("onclick", this.Button00_onclick, this);

5.2.2 오브젝트 타입

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

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

입력된 타입 값을 기준으로 넥사크로 스튜디오에서 인텔리센스 기능을 지원합니다.



해당 코드를 자바스크립트 코드로 변환할 때는 타입 값을 제거합니다.

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

오브젝트 타입을 지정하지 않아도 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.

5.3.1 set 메소드

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

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

아래와 같이 속성값에 직접 접근하는 방식은 더는 지원하지 않습니다.

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

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

```
<Button myprop="333">

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

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

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

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

5.3.2 동적인 속성

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

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

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

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

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

5.4 기타 변경 사항

5.4.1 nexacro 메소드

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

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

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

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

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

또한, 자바스크립트에서 예약어로 사용하는 일부 오브젝트의 명칭이 변경되었습니다.

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

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

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

속성이나 오브젝트도 예약어와 충돌되어 일부 변경되었습니다.

Component.class○→ Component.cssclass
ExcelExportObject.export() → ExcelExportObject.exportData()
VirtualFile.delete() → VirtualFile.remove()

5.4.2 동작 방식 변경

자바스크립트에서 지원하지 않거나 다르게 동작하는 일부 항목이 수정되었습니다.

<> 비교 연산자 지원하지 않음

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

switch 문 내 문자열 처리 방식 변경

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

정규표현식 /g 옵션 적용 방식 변경

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

5.4.3 오브젝트 명 생성 시 제약

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

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

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

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

5.4.4 변수명, 함수명 생성 시 제약

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

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

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

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

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

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

5.4.5 추가 모듈에서 window 오브젝트 사용 시 제약

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

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

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

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

6.

Frame Tree

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

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

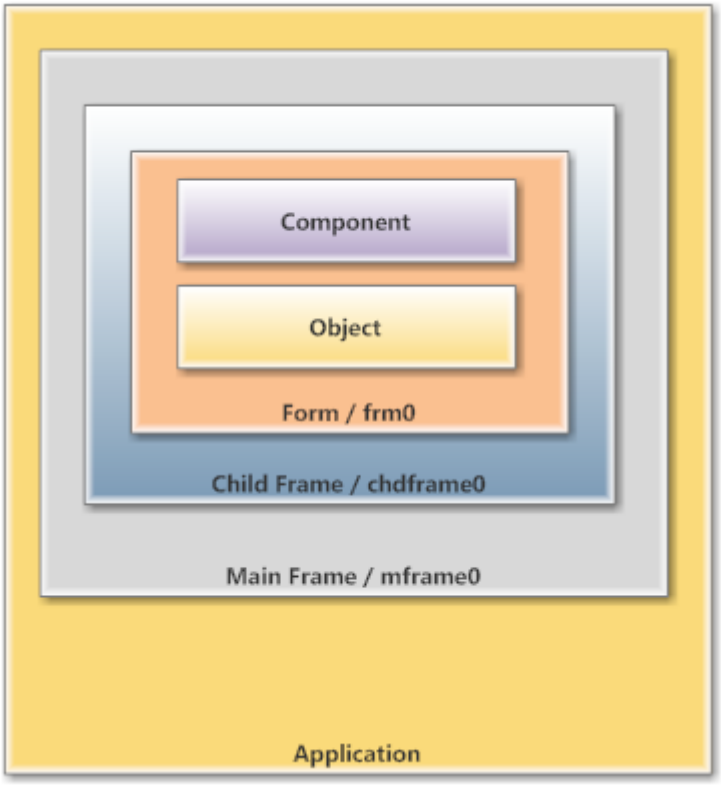
6.1 Application

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

// mainframe -> application
this.mainframe.parent

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

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

// childframe -> form
```

```
this.mainframe.frame.form

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

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

6.1.1 Script 예시 화면



6.1.2 application에서 form 오브젝트 접근

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

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

6.1.3 form에서 상위 오브젝트 접근

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

6.2 Form

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

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

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

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

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

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

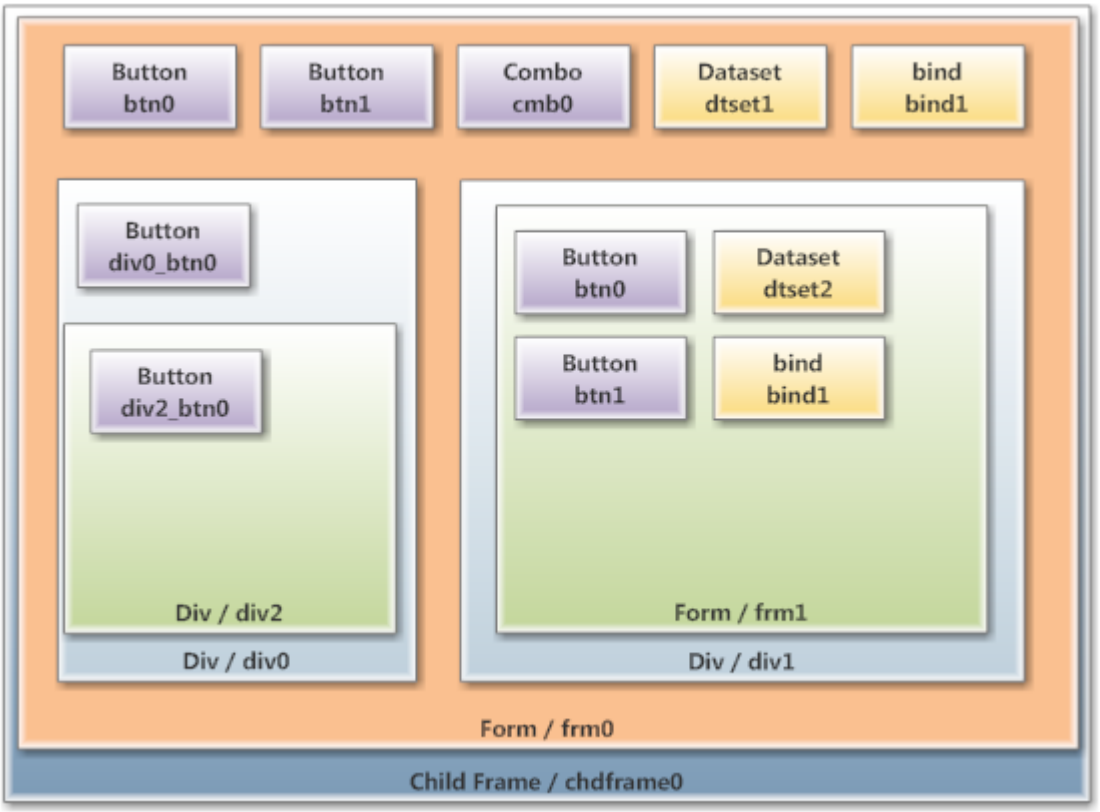
Div, PopupDiv, Tabpage와 같은 컨테이너 컴포넌트는 url 속성을 지정해 Form을 연결할 수 있습니다. 이런 경우 컨테이너 컴포넌트와 Form은 1:1 구조입니다. 스크립트 상에서 Form에 연결된 Object, Bind, Component는 아래와 같은 형식으로 접근할 수 있습니다.

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

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

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

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

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

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

6.2.1 Script 예시 화면



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

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

6.2.2 component / object / bind

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

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

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

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

6.2.3 container component

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

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

```
// div2내의 div2_btn0으로의 접근
this.div0.form.div2.form.div2_btn0.name == "div2_btn0"
this.all["div0"].form.all["div2"].form.all["div2_btn0"].name == "div2_btn0"
this.components["div0"].form.components["div2"].form.components["div2_btn0"].name == "div2_btn
0"
```

6.2.4 form을 연결한 container component

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

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

6.2.5 parent

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

파트 III.

참고

7.

애플리케이션 캐시

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

7.1 캐시의 종류

지원하는 캐시의 종류는 아래와 같습니다.

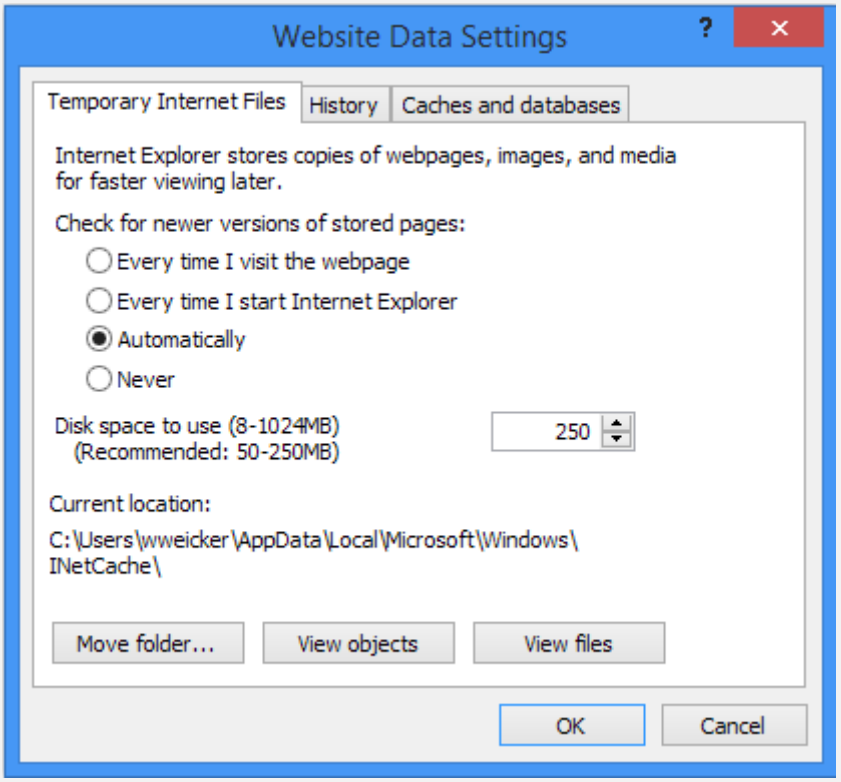
캐시 상태	설명
none	캐시 기능을 사용하지 않습니다.
dynamic	서버로부터 내려받은 파일이 갱신되었을 때만 수신합니다. 서버상의 파일이 갱신되지 않은 경우는 로컬 캐시(Local Cache) 파일을 재사용합니다.
session	엔진을 기동할 때, 단 한 번만 수신하고 해당 엔진이 종료될 때까지 로컬 캐시 파일만을 사용합니다. (엔진 기동 시 로컬 캐시 파일과 일치하면 재사용합니다.)
static	서버로부터 한 번이라도 내려받은 파일은 엔진 재실행과 상관없이 로컬 캐시 파일만 사용합니다. (단, Type Definition에 지정된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

cachelevel 속성값을 지정하지 않았을 경우 기본 캐시 속성은 Dynamic으로 적용됩니다.

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

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

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



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

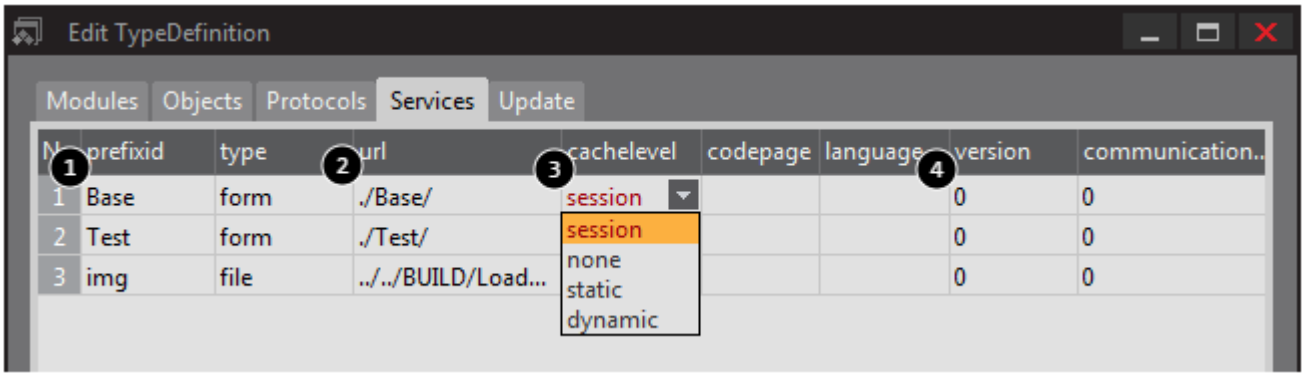
넥사크로 스튜디오에서 [Launch Project] 또는 [Quick View] 메뉴를 사용해 Runtime을 실행하는 경우에는 typedefinition에 설정된 Service-cachelevel 값을 무시하고 none으로 동작합니다.
(cachelevel 속성의 설정값이 변경되는 것은 아닙니다)

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

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

7.2 캐시 적용 방법

다음은 Type Definition 편집기에 대한 설명입니다.



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

Script

```
application.services["Base"].set_cachelevel("none" )
```

8.

Dataset XML Format

넥사크로플랫폼은 서버모듈로 X-API를 제공합니다. 이 API를 사용하여 생성된 Dataset format을 넥사크로플랫폼 런타임이 해석해 애플리케이션을 화면에 보여줍니다.

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

8.1 Dataset XML layout

XML Declaration					
Root					
	Cache				
	Parameters				
		Parameter (반복)			
	Dataset (반복)				
		ColumnInfo			
			ConstColumn (반복)		
			Column (반복)		
		Rows			
			Row (반복)		
				Col (반복)	
				OrgRow	
					Col (반복)

8.1.1 XML 선언

XML임을 명시하는 XML 선언문을 다음과 같이 정의합니다.

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

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

8.1.2 XML 예

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

8.2 Dataset 요소

Dataset의 각 요소(element)들을 설명합니다.

8.2.1 Root

개요

Dataset을 나타내는 최상위 요소

자식 요소

Parameters, Dataset

반복 여부

반드시 1개가 있어야 합니다.

속성

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

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

8.2.2 Parameters

개요

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

자식 요소

Parameter

반복 여부

없거나 1개가 있을 수 있습니다.

8.2.3 Parameters > Parameter

개요

파라미터의 값을 명시합니다.

자식 요소

없다.

반복 여부

없거나 Parameter 요소는 파라미터 개수만큼 반복될 수 있습니다.

속성

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

속성 이름	설명
id	파라미터 이름
type	파라미터값의 타입

비고

넥사크로플랫폼에서는 다음과 같은 필수 파라미터를 사용하도록 정의하고 있습니다. 넥사크로플랫폼과의 연동을 위해서는 파라미터를 설정하여야 합니다.

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

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

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

type에 올 수 있는 데이터 타입은 다음과 같습니다.

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

8.2.4 Dataset

개요

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

자식 요소

ColumnInfo, Rows

반복 여부

없거나 xml이 전달하는 Dataset의 개수만큼 반복될 수 있습니다.

속성

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

속성 이름	설명
id	Dataset 이름

8.2.5 Dataset > ColumnInfo

개요

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

자식 요소

ConstColumn, Column

반복 여부

없거나 1개가 있을 수 있습니다.

8.2.6 Dataset > ColumnInfo > ConstColumn

개요

항상 고정된 값을 갖는 컬럼의 스키마를 표현합니다.

자식 요소

없다.

반복 여부

없거나 고정값 컬럼의 개수만큼 있을 수 있습니다.

속성

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

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

비고

type에 올 수 있는 데이터 타입은 다음과 같습니다.

- STRING: 문자열. 컬럼의 크기와 관계없이 길이 제한이 없습니다.(단 시스템에서 한 번에 할당 가능한 크기인 2GB 정도로 제한됩니다)
- INT: 정수(-2^{31} ~ $2^{31}-1$)
- FLOAT, DECIMAL: ($\pm 2.2X10^{-308}$ ~$\pm 1.7X10^{308}$)
- BIGDECIMAL: 실수($\pm 10^{-1056}$ ~ $\pm 10^{1056}$)
- DATE: YYYYMMDD(-8192/01/01 ~ 8191/12/31)
- DATETIME : YYYYMMDDHHmmssuuu (msec포함가능)
- TIME : 6자리 HHmmssuuu(msec포함가능)
- BLOB: 이진 데이터

8.2.7 Dataset > ColumnInfo > Column

개요

파라미터의 값을 명시합니다.

자식 요소

없다.

반복 여부

없거나 비 고정값 컬럼의 개수만큼 있을 수 있습니다.

속성

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

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

비고

type에 설정되는 값은 ColumnInfo에서의 type과 같습니다.

Prop은 다음과 같은 값을 갖습니다.

- count : 컬럼의 row 개수를 summary 표시합니다.
- sum: 컬럼의 row값의 합을 summary에 표시합니다.
- max: 컬럼의 row들 중 Max값을 summary에 표시합니다.
- min: 컬럼의 row들 중 Min값을 summary에 표시합니다.
- avg: 컬럼의 row값의 평균을 summary에 표시합니다.
- Text: summary에 sumtext 속성값을 표시합니다.

8.2.8 Dataset > Rows

개요

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

자식 요소

Row

반복 여부

없거나 1개 있을 수 있습니다.

8.2.9 Dataset > Rows > Row

개요

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

자식 요소

Col, Org_Row

반복 여부

없거나 Dataset의 Row 개수만큼 있을 수 있습니다.

속성

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

<Rows>
     <Row type="update">
          <Col id="currentCode">10001</Col>
          <Col id="currentprice">13400</Col>
          <OrgRow>
               <Col id="currentCode">10001</Col>
               <Col id="currentprice">13700</Col>
          </OrgRow>
     </Row>
</Rows>

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

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

8.2.10 Dataset > Rows > Row > Col

개요

Dataset의 각 컬럼값을 표현합니다.

자식 요소

없습니다.

반복 여부

없거나 컬럼의 개수만큼 있을 수 있습니다.

속성

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

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

비고

빈 문자열인 경우

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

Null인 경우 Tag가 없는 것으로 구분합니다.

8.2.11 Dataset > Rows > Row > OrgRow

개요

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

자식 요소

Col

반복 여부

없거나 1개 있을 수 있습니다.

비고

부모 요소인 Row와 유사하나 속성이 없습니다.

8.2.12 Dataset > Rows > Row > OrgRow > Col

개요

변경되기 전의 Column 값을 표현합니다.

자식 요소

없습니다.

반복 여부

없거나 컬럼의 개수만큼 있을 수 있습니다.

속성

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

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