NEXCORE Alopex UI 2.3

Alopex UI 개발가이드

Version 1.1.2

SK 주식회사 C&C

제•개정 이력

제•개정 번호	제•개정 이력	제•개정 일자
1.0.0	최초 작성	2015-04-01
1.0.1	컨텐츠 현행화(Alopex Runtime 내용 삭제)	2016-01-21
1.1.0	표준 문서 양식으로 재작성	2016-12-02
1.1.1	홈페이지 메뉴 구조 변경에 따른 링크 수정	2017-08-30
1.1.2	홈페이지 메뉴 구조 변경에 따른 링크 수정	2018-07-05

목 차

Alopex UI 개발가이드

1. 문서 개요

본 문서는 프로젝트에서 Alopex UI 프레임워크 (이하 Alopex) 를 활용하여 어플리케이션을 개발하는 절차 및 방법을 기술합니다.

1.1. 목적

본 프로젝트의 개발 절차 및 방법을 가이드 함으로써, 개발의 생산성과 효율성을 향상하고, 개발된 결과의 통일성을 유지하여 향후 유지보수의 편의성, 높은 가독성을 제공하기 위함입니다

1.2. 문서의 범위

본 문서에서는 다음의 사항을 기술합니다

- **프레임워크 기반 기본 아키텍처**
- **개발 환경**
- **개발 절차**
- **API 레퍼런스**
- **프로그래밍 가이드**

본 문서는 기본 웹 기술(HTML, Javascript, CSS) 관련 내용은 자세히 다루지 않습니다. (개발자가 웹 기술에 대한 기본적인 지식과 내용은 이미 갖추었다고 가정)

1.3. 대상

본 문서는 Alopex 프레임워크를 활용하여 어플리케이션을 구현하는 개발자를 대상으로 합니다

2. 아키텍처

본 장에서는 프레임워크 기반의 애플리케이션 아키텍처 및 제약사항들을 설명합니다

2.1. 아키텍처

2.1.1. UI 컴포넌트

Alopex 는 HTML 요소에 기 정의된 class 이름을 지정할 경우, 자동으로 UI 컴포넌트로 만들어줍니다. Alopex 그리드나 달력과 같은 복잡한 마크업을 가진 컴포넌트의 경우에는 API 호출을 통해 컴포넌트 생성을 하게 됩니다. 사용자가 원하는 컴포넌트를 만들어 Alopex 의 구조에 확장하여 사용할 수 있습니다.

유형	Class 기반 컴포넌트	Javascript 기반 컴포넌트	사용자 컴포넌트
코드 예제	<button class="Button"></button>	$('#id').showDatePicker();	
컴포넌트 종류	버튼, 체크박스, 텍스트인풋, 탭 등	날짜선택기, 다이얼로그 등	사용자 정의
설명	HTML 요소에 컴포넌트 class(대문자 시작)를 명시하면 Alopex 가 문서 로딩 시점에 이를	Javascript API 를 통해서 사용자가 명시적으로 컴포넌트를 생성합니다. HTML 마크업은 없거나,	컴포넌트를 상속하거나 새로운 컴포넌트를 만들어서 사용자가 원하는 컴포넌트를 만듭니다.

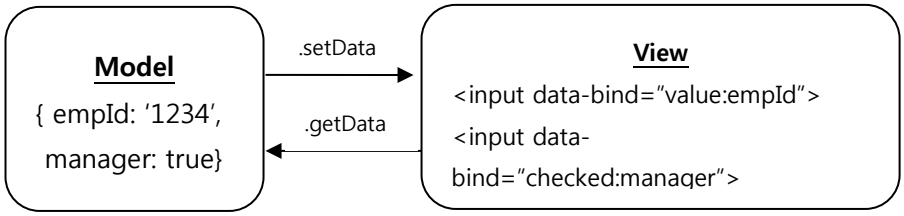
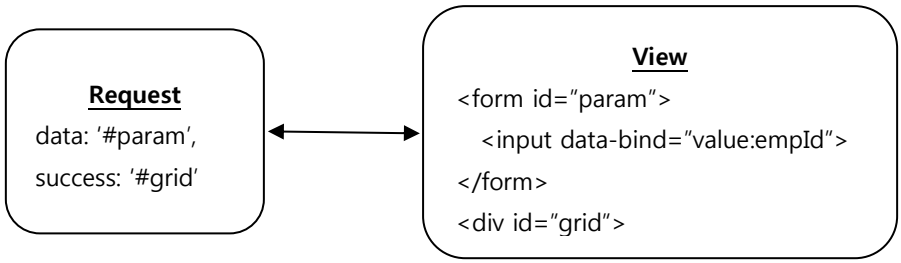
	컴포넌트로 만들어줍니다. HTML 마크업은 자동 생성되지 않고, 모두 작성해줍니다.	div 와 같은 기본적인 하나의 컨테이너 요소로만 구성되며, 상세한 마크업은 동적으로 생성됩니다.	오픈소스 컴포넌트를 이와 같은 형태로 만들어 wrapping 할 수 있습니다.

- Alopex 가 제공하는 컴포넌트 종류는 http://ui.alopex.io/를 참고하세요.

2.1.2. 데이터바인딩

HTML 요소에 data-bind="type: key" 속성을 지정하여 서버-화면 간 데이터를 매핑합니다.

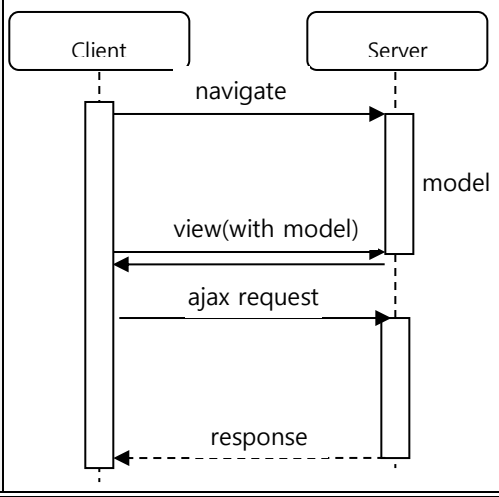
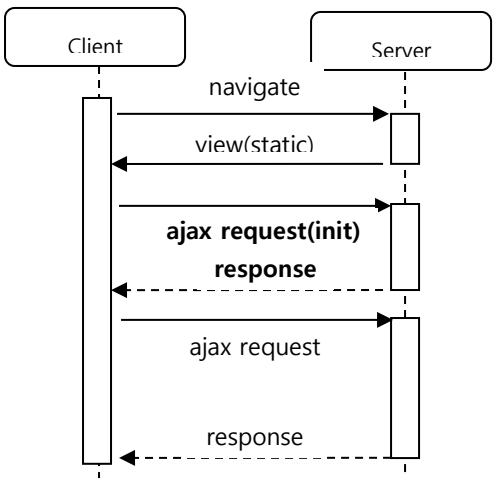
.setData 와 .getData API 를 통해 JSON 과 HTML 간의 바인딩이 수행되며, 통신함수에서 자동 바인딩을 통해 API 호출 없이 바로 서버와 UI 를 연결할 수 있습니다

유형	개념도
수동 데이터바인딩	
자동 데이터바인딩	

- value, checked, selected 등의 지시자(control)는 사용자가 확장하여 사용 가능합니다.

2.1.3. 화면 이동 및 통신

Alopex 는 서버사이드 MVC 를 하지 않고, 클라이언트에서 100% Ajax 방식으로 통신하는 형태를 표준으로 합니다. 화면 이동 시 form submit 을 통해 파라미터를 서버에 전달하지 않고, 파라미터는 화면 이동 시 클라이언트에서 다음 화면으로 전달되며, 다음 화면에서 초기화 시 통신을 수행하게 됩니다.

	서버 MVC + Ajax 방식	Alopex Full-Ajax 방식
개념도		

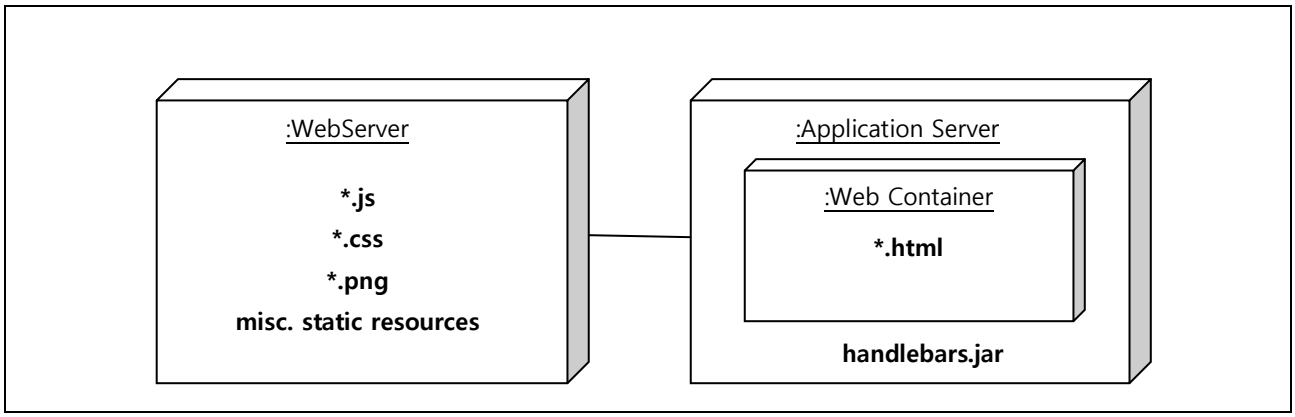
- 서버 MVC 와는 달리, 화면의 초기화 시 Ajax 로 데이터를 불러오도록 하는 것이 핵심

2.1.4. Handlebars 템플릿팅

클라이언트 랜더링 시 깜빡임 현상(FOUC: Flash On Unstyled Content)을 최소화하고, 보안 측면을 강화하기 위해 Alopex 는 일부 영역을 서버사이드 랜더링에 위임합니다.

Handlebars(http://handlebarsjs.com/)의 템플릿 문법을 사용합니다. 자세한 사항은 동적 페이지 구성을 참고하세요.

- ✓ 페이지 공통영역 처리(GNB, LNB, 공통 include 리소스 등)
- ✓ 코드/메시지의 key/value 치환 처리
- ✓ 국제화(i18n) 레이블/메시지 치환
- ✓ 권한에 따른 Show/Hide 처리
- ✓ ContextPath 등 동적인 Path 처리



- *.html 은 애플리케이션 서버 내에서 템플릿 파일 역할을 수행합니다.

2.2. 구성

2.2.1. HTML 파일

HTML 파일은 화면 표현을 위한 마크업 기능 외에 추가적으로 아래 역할을 수행합니다.

- ✓ 데이터 바인딩 매핑: data-bind 속성을 통해 HTML 요소에 바인딩 될 key 를 지정합니다.
- ✓ 유효성 검증 룰: data-validation-rule 속성을 통해 HTML 요소의 유효성 검증 룰을 지정합니다.
- ✓ 템플릿: handlebars 템플릿 표현을 통해 템플릿팅 엔진이 치환할 영역을 명시합니다.

```
<form>
<input class=”Textinput” data-bind=”value:empId”>
  <input class=”Textinput” data-validation-rule=”required:true”>
</form>
```

2.2.2. Javascript 파일

각 화면별 자바스크립트 파일 또는 모듈용 자바스크립트 파일은 별도 파일들로 생성합니다.

2.2.2.1. 화면 Javascript 파일

- ✓ $a.page 의 함수에 function 을 파라미터로 넣어서 초기화 함수를 작성합니다.
- ✓ 내부 함수(사용자 함수)는 function 으로 생성합니다.

```
<page1.js>7

$a.page(function(){

        this.init = function(id, param){
                //초기화 루틴 수행
                func1();
        };

        function func1(){
                //내부 함수
        }

});
```

2.2.2.2. 모듈용 Javascript 파일

- ✓ $a.page 의 함수에 function 을 파라미터로 넣어서 초기화 함수를 작성합니다.
- ✓ 파일명과 모듈(변수)명은 항상 일치시킵니다(module1.js – var module1)
- ✓ 외부에서 사용할 함수는 this 에 등록하고, 내부 함수는 function 으로 생성합니다.
- ✓ $a.page 를 리턴 받아 글로벌 변수에 등록하고 이를 페이지 등에서 활용합니다.

```
<module1.js>

var module1 = $a.page(function(){

        this.init = function(){
                //초기화 루틴 수행
        };
        this.func2 = function(){
                //외부 함수
        };
        function func1(){
                //내부 함수
        }

});
<page1.js>

$a.page(function(){
        this.init = function(){
                //module1의 외부 함수 호출
                module1.func2();
        };
});
```

2.2.3. UI 컴포넌트

- ✓ HTML 요소에 기 정의된 class 를 지정합니다.
- ✓ Javascript UI 컴포넌트는 페이지 Javascript 에서 API 호출을 통해 생성합니다.
- ✓ 사용자 정의 컴포넌트는 공통 Javascript 파일에서 미리 등록한 다음 사용합니다.

```
<page1.html>

<input id="txt0" class="Textinput">
<div class="MyWidget"></div>
```

```
<page1.js>

$a.page(function(){

        this.init = function(){
                $('#txt0').setEnabled(false);
        };

});
```

```
<common.js>
$a.setup({
defaultComponentClass: {
  mywidget: 'MyWidget',
  …
}
});

$a.widget.mywidget=$a.inherit($a.widget.object, {
widgetName:'mywidget',
setters: …
getters: …
…
});
```

2.2.4. Ajax 및 데이터 객체

- ✓ 모든 통신은 $a.request API 를 통해 수행합니다.
- ✓ Alopex 에서는 별도 데이터 객체 스키마를 사용하지 않으며, Javascript 의 일반적인 Literal Object 를 사용합니다.

```
page1.js

$a.request('foo/bar', {
        data:{baz: 'qux'},
        success: function(res){
                var obj1 = res.key1;
                var obj2 = res.key2;
                …
        };
});
```

2.3. 개발 규칙 및 유의사항

- **아래의 내용을 포함하여 상세 개발표준 수립 후 프로젝트에서 작성**

- ✓ HTML 작성 시 유의사항 - JSP 가 아님
- ✓ 페이지 스크립트 생성 규칙
- ✓ 단위: 탭페이지 내 탭의 단위, 팝업 페이지의 단위
- ✓ 함수 명명 규칙

2.4. 서버 환경 배포 시 유의사항

배포 정책 수립 후 프로젝트에서 작성

3. 개발 환경

3.1. UI 프로젝트 디렉토리 구조

UI 프로젝트 디렉토리 구조는 아래와 같은 구조로 가집니다.

```
AlopexUI_package
├──css──── app
│      └──── lib
│             └── alopex
│                 └── alopex-ui-xxx.css (alopex-ui-default.css가 기본 디자인)
│                 └── images
│                 │   └── *.jpg, *.png, ...
│                 └── src
│                     └── alopex-ext.css
│
├── html
└── script──── app
           └──── lib
                 └── alopex
                 │   └── alopex-ui.min.js
                 │   └── src
                 │       └── alopex-ext.js, alopex-ext-setup.js
                 │
                 └── jquery
                     └──jquery-x.x.x.js (1.10.x / 1.11.x 등 사용)
```

폴더명	설명
script	app, lib 등의 sub-directory 를 갖습니다. 'app' 은 업무 로직 javascript 소스들이 위치합니다. 'lib'은 alopex, jquery 등의 javascript 라이브러리 소스들이 위치합니다. 벤더별로 그룹화합니다.
css	app, lib 등의 sub-directory 를 갖습니다. 'app' 은 업무 화면의 스타일시트 소스들이 위치합니다. 'lib'은 alopex, jquery 등의 스타일시트 소스들이 위치합니다. 벤더별로 그룹화합니다.
html	업무화면 html 파일 디렉터리. 업무 화면이 위치합니다.

4. 개발절차

본 장에서는 Alopex UI 프레임워크 (이하 Alopex)를 이용해 개발 프로젝트 진행 시, 고려해야 할 사항과 개발 프로세스를 가이드합니다. 아래 내용에서 개발도구는 Eclipse 를 사용하는 기준으로 작성되었습니다.

4.1. 라이프사이클

순번	개발 TASK	주요작업내용	수행도구	선행 TASK
1	HTML 생성	HTML 페이지 생성	Eclipse	-
2	CSS 생성	스타일 클래스 생성	Eclipse	1
3	페이지 스크립트 생성	Javascript 페이지 생성	Eclipse	1
4	페이지 스크립트 함수 생성	Javascript 함수 생성(업무 로직 작성)	Eclipse	1,3
5	배포 및 테스트	설정한 테스트케이스를 가지고 메서드기능 검증	웹 브라우저	1,2,3,4
6	형상관리 반영	형상관리 레파지토리 Check-In 또는 Check-Out	형상관리 PlugIn	5

4.2. HTML 생성 절차

아래 절차에 따라 html 파일을 생성합니다.

- ✓ Eclipse 탐색기 뷰에서 html 파일을 생성하려는 프로젝트의 '업무 폴더'를 마우스 우클릭하여 New > File 메뉴를 선택
- ✓ 팝업 창에서 HTML 파일명을 입력하고 Finish 선택
- ✓ 생성된 html 파일에 내용을 작성하시면 됩니다.

4.3. CSS 생성 절차

아래 절차에 따라 퍼블리싱, 업무 공통 css 파일을 생성합니다.(화면별 css 파일은 생성하지 않습니다.)

- ✓ Eclipse 탐색기 뷰에서 css 파일을 생성하려는 프로젝트의 '업무 폴더'를 마우스 우클릭하여 New > File 메뉴를 선택
- ✓ 팝업 창에서 css 파일명을 입력하고 Finish 선택
- ✓ 생성된 css 파일에 퍼블리싱, 업무 공통 css 를 작성하시면 됩니다.

4.4. 페이지 스크립트 생성 절차

아래 절차에 따라 javascript 파일을 생성합니다.

- ✓ Eclipse 탐색기 뷰에서 javascript 파일을 생성하려는 프로젝트의 '업무 폴더'를 마우스 우클릭하여 New > File 메뉴를 선택
- ✓ 팝업 창에서 Javascript 파일명을 입력하고 Finish 선택
- ✓ 페이지 스크립트 작성 절차와 방법은 프로그래밍 가이드의 페이지를 참조하세요.

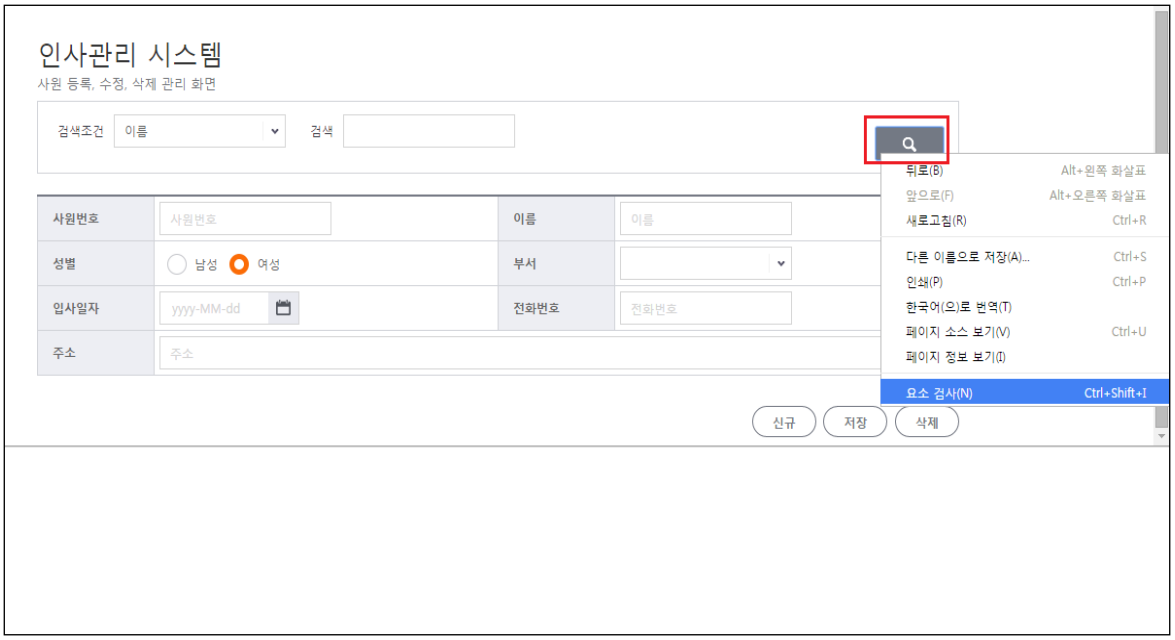
4.5. 배포 및 테스트

애플리케이션 개발시 테스트를 위하여 크롬 브라우저를 사용합니다. 크롬 브라우저의 디버깅 도구인 요소 검사를 이용하면 테스트를 보다 쉽고 빠르게 할 수 있습니다.

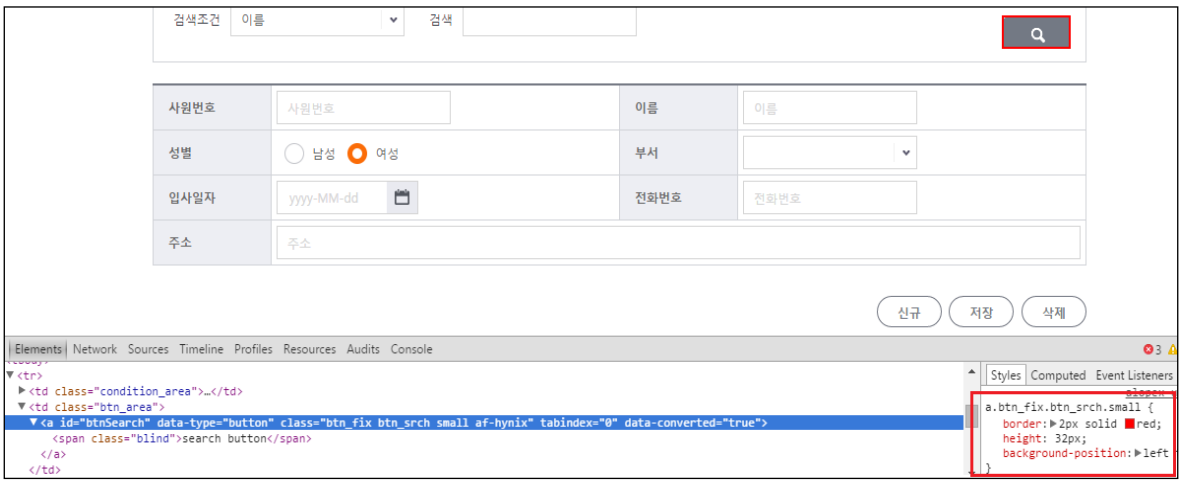
4.5.1. 크롬 요소검사

크롬 브라우저의 요소 검사 기능을 사용하면 html 태그, css 속성을 실시간으로 편집하고 미리보기 할 수 있습니다.(실제 source code 에 반영되지 않음)

- **크롬 브라우저에서 업무화면을 띄우고 요소검사를 하고 싶은 부분에서 마우스 우클릭하여 요소 검사 메뉴를 선택**



- **크롬 브라우저 하단(또는 팝업)에 Elements 탭에 해당 요소의 마크업이 나오고, 그 우측에 css 속성이 나옵니다. css 속성에 원하는 스타일을 정의하면 실시간으로 반영됩니다.(아래 화면에서는 button 에 border 속성을 red color 로 주어서 button 의 테두리가 붉은색으로 바뀐 것을 볼 수 있습니다.)**



4.5.2. 크롬 콘솔

크롬 브라우저의 Console 탭에서는 화면 테스트 도중 발생하는 javascript 에러를 확인할 수 잇습니다.

4.5.3. 크롬 디버깅

크롬 브라우저의 Sources 탭에서는 javascript 코드를 라인단위로 디버깅하며 테스트 할 수 있습니다.

4.6. 형상관리 반영

테스트까지 완료된 소스는 형상관리 시스템을 통해 반입(Check-In)합니다.

5. API 레퍼런스

5.1. 홈페이지 소개

Alopex UI 홈페이지(http://ui.alopex.io)에서 개발가이드, API 를 확인할 수 있으며 버젼별 업데이트 내역과 최신 라이브러리를 다운로드를 할 수 있습니다.

5.2. 가이드 활용 방법 간단 설명

Alopex UI API 사용법은 Development 메뉴에서 확인할 수 있습니다. 아래와 같이 컴포넌트별 API 가이드 페이지로 이동합니다.

- ✓ 홈페이지 상단의 Development > Component 또는 Javascript 선택
- ✓ Component 또는 Javascript 페이지의 좌측 메뉴에서 해당 컴포넌트 및 Javascript 선택

5.2.1. Basic

컴포넌트를 생성하기 위한 마크업을 확인할 수 있습니다.

```
//버튼 생성
<button class="Button">Button</button>

//toggle 버튼 생성
<button class=" Button Toggle">Button</button>
```

5.2.2. Attributes

마크업으로 컴포넌트의 속성을 정적으로 변경할 수 있습니다.

```
<button class="Button" data-disabled=”true”>Button</button>
```

5.2.3. Functions

함수를 호출해 컴포넌트의 속성을 동적으로 변경할 수 있습니다.

```
$('#btn1').setEnabled(false);
<button id=”btn1” class="Button">Button</button>
```

5.2.4. Extra Examples

추가적인 예제코드를 확인할 수 있습니다.

5.2.5. Setup

$a.setup() API 를 통한 컴포넌트의 전역 옵션 설정을 확인할 수 있습니다.

6. 프로그래밍 가이드

6.1. 퍼블리싱

Alopex 를 활용하여 퍼블리싱을 할 때 절차 및 방법을 기술합니다. 아울러 개발 시 유의할 사항과 가이드라인 등을 다룹니다.

- **홈페이지 상단의 Document > 퍼블리싱 가이드 참고**

6.2. Javascript 코딩 컨벤션

Javascript 코드 작성 시 지켜야 할 표준에 대한 가이드입니다. 개발된 결과의 통일성을 유지하여 향후 유지보수의 편의성, 높은 가독성을 제공하기 위해 준수해야 합니다.

6.2.1. Javascript 작성 규칙

- **var 는 생략하지 않습니다.**

```
//bad
a = 3;

//good
var a = 3;
```

- **여러 변수 선언은 var 를 한 번만 사용합니다.**

```
//bad
var a = 1;
var b = 2;
var c = 3;

//good
var a = 1,
    b = 2,
    c = 3;
```

- **세미콜론(;)은 생략 가능할 수 있으나, 반드시 적어 줍니다.**
- **eval() 함수는 사용하지 않습니다. 대신, JSON.parse 를 사용합니다.(보안적인 측면)**
- **multiline 스트링은 + 로 붙여 줍니다.**

```
//bad
var str = 'aabbcc\
      ddeeff';

//good
var str = 'aabbcc' +
    'ddeeff';
```

- **Array 와 Object 의 초기화는 생성자 대신 리터럴을 사용합니다.**

```
//bad
var arr = new Array('a','b');
var arr2 = new Array();
var obj = new Object();
obj.a = 1;
var obj2 = new Object();

//good
var arr = ['a', 'b'];
var arr2 = [];
var obj = {a: 1};
var obj2 = {};
```

- **배열 루프는 for-in 대신 일반 for 문으로 돌립니다. for-in 은 Object loop 에만 사용합니다.**

```
var arr = [...];

//bad
for (var n in arr){
 ...
}

//good
for (var i = 0; i < arr.length; i++){
...
}
```

6.2.2. jQuery 사용 규칙

- **Alopex 사용 시 $(document).ready 대신 $a.page 을 사용합니다. $(document).ready 는 유틸리티성 js 코드 내에서 꼭 필요한 경우에만 사용합니다.**

```
//bad
$(document).ready(function(){
...
});

//good
$a.page(function(){
    this.init = function(){
    ...
    };
});
```

- **HTML attribute 에 event 나 addEventListener 대신 jQuery bind 를 사용합니다.(크로스 브라우징)**

```
//bad
<button onclick=" ..."></button>
btn.addEventListener("click", function(){
...
});

//good
$('#btn').click(function(){
...
});
```

- **DOM API 대신 jQuery API 를 사용합니다.(크로스 브라우징)**

```
//bad
document.getElementById("myBtn");
document.querySelector(".example");

//good
$('#myBtn')
$('.example')
```

- **아래와 같은 naming 규칙을 따릅니다.**

```
functionNamesLikeThis
variableNamesLikeThis
CONSTANT_VALUES_LIKE_THIS
```

- **jQuery Object 를 받는 변수는 구분을 위하여 $를 앞에 붙입니다.**

```
var $div = $('#myDiv');
```

- **따옴표는 반드시 single quotation(')을 사용합니다. jQuery selector 나 HTML attribute 와 혼용할 경우에만 double quotation 을 씁디다.**

```
//bad
var msg = "This is some HTML";

//good
var msg = 'This is some HTML';
var html = '<div style="color:red;"></div>';
var $anchor = $('a[href="localhost"]')
```

- **IF 문 하나에도 brace 를 사용합니다.**

```
//bad
if (value) a = 1;

//good
if (value){
    a = 1;
}
```

- **함수 주석은 jsDoc 형태로 작성합니다. 나머지는 line 주석을 사용합니다.**

```
/**
 * Operates on an instance of MyClass and returns something.
 * @param {project.MyClass} obj Instance of MyClass which leads to a long
 *     comment that needs to be wrapped to two lines.
 * @return {boolean} Whether something occurred.
 */
function PR_someMethod(obj) {
  // ...
}
```

- **자료형의 확인은 typeof 를 사용합니다.**

```
if(typeof variable === 'string'){}
```

- **Switch 문 대신 객체 리터럴을 사용합니다.**

```
//bad
switch( foo ) {
  case 'alpha':
    alpha();
    break;
  case 'beta':
    beta();
    break;
  default:
    //...
    break;
}

//good
var switchObj = {
  alpha: function() {
    // ....
  },
  beta: function() {
    // ....
  },
  _default: function() {
    // ....
  }
};
switchObj[foo]();
```

- **Boolean 의 적절한 사용. 아래와 같은 값은 Boolean 이 false 입니다.**

```
null
undefined
'' the empty string
0 the number
```

- **아래는 Boolean 이 true 이므로 주의합니다.**

```
'0' the string
[] the empty array
{} the empty object
```

- **그러므로 아래 코드처럼, 간결한 분기문을 작성합니다.**

```
//bad
while (x != null) {
if (y != null && y != '') {

//good
while (x) {
if (y) {
```

- **비교문에 === 사용(등호 세 개를 사용합니다. 값과 자료형 모두 비교)**

```
//bad
if (a == 3 && b != 2){

//good
if (a === 3 && b !== 2){
```

- **Object Property 는 점(Dot.)을 사용합니다. 동적인 이름만 []를 사용합니다.**

```
var obj = {a: 1, b: 2};
var key = 'a';

//bad
var c = obj['a'];

//good
var c = obj.a;
var d = obj[key];
```

6.2.3. 코딩 스타일 규칙

들여쓰기는 space 대신에 Tab 을 사용합니다. 개발도구에서 반드시 공백 문자가 화면에 표시되도록 설정해 놓습니다

6.3. 공통

아래는 프로젝트에서 업무 개발의 편의성과 생산성 향상을 위해 공통으로 처리해야 할 코드 작성 가이드입니다.

6.3.1. 통신 셋업

통신 요청에 글로벌로 사용되는 옵션값을 미리 설정하여, 화면에서 호출되는 $a.request 함수의 코드를 최소화할 수 있습니다.

6.3.1.1. 글로벌 옵션

url, async, timeout, method 옵션과 같이 모든 $a.request 함수에서 동일하게 사용되는 옵션을 기본값으로 설정할 수 있습니다.

```
$a.request.setup({
url : function(id, option) {
      return 'http://ui.alopex.io/' + id;
   },
   async: 'true',
   method: 'post',
   timeout: 30000
});
```

6.3.1.2. 글로벌 콜백함수

- **before(전처리)**

통신 요청을 하기 전 헤더를 설정하거나 프로그레스를 띄우는 처리를 합니다.

```
var progress = null //글로벌 변수

$a.request.setup({
   before: function(id, option) {
            //헤더 설정. this는 통신 오브젝트
      this.requestHeaders["Content-Type"] ="application/json; charset=UTF-8";

            //프로그레스 띄우기
            progress =  $('block 처리할 영역').progress();
   }
});
```

- **after(후처리)**

after 함수는 서버로부터 응답을 받은 이후 요청의 처리 성공/실패를 판단하는 함수입니다. request 오브젝트의 isSuccess 속성값을 true/false 로 저장해 success/fail 함수를 호출합니다.

```
$a.request.setup({
   after: function(res) {
      if( success condition ) {
         this.isSuccess = true;
       }else{
         this.isSuccess = false;
       }
   }
})
```

- **success(요청성공)**

after 함수에서 요청성공으로 판단한 경우 호출됩니다. 프로그레스를 닫는 등 요청성공 시 업무 공통적으로 처리할 부분을 작성합니다.

```
$a.request.setup({
    success: function(res) {
                //프로그레스 닫기
                progress.remove();
       }
})
```

- **fail(요청실패)**

after 함수에서 요청실패로 판단한 경우 호출됩니다. 프로그레스 닫기, 실패 메시지 팝업 등 요청실패 시 업무 공통적으로 처리할 부분을 작성합니다.

```
$a.request.setup({
   fail: function(res) {
                //프로그레스 닫기
                progress.remove();

                //실패 메시지 처리
       }
})
```

- **error(통신에러)**

서버와의 통신 오류 시 호출됩니다. 프로그레스 닫기, 에러 메시지 팝업 등 통신에러 시 업무 공통적으로 처리할 부분을 작성합니다.

```
$a.request.setup({
   error: function(res) {
                //프로그레스 닫기
                progress.remove();

                //실패 메시지 처리
       }
})
```

6.3.1.3. 통신 인터페이스

서버와의 통신과 Alopex 데이터 바인드 기능을 사용하기 위해서 정해진 데이터 규격이 필요합니다. 이 설정을 통신 인터페이스 설정이라고 합니다.

- **interface**

서버와의 통신을 위한 데이터 규격입니다. 서버환경에 따라 변경될 수 있습니다.

```
$a.request.setup({
   interface : {
      message: {},
      fieldData : {},
      listData : {}
   }
})
```

- **object**

object 데이터를 자동 바인드하기 위한 설정입니다. 위 interface 에서 fieldData 가 object 데이터라면 아래와 같이 바인드 할 데이터를 리턴합니다.

```
$a.request.setup({
   object : function(elem, data) {
      //elem: 데이터가 바인드 될 HTML 엘리먼트
      //data: 서버로부터 받은 응답 오브젝트(이 오브젝트를 이용하여 데이터 참조 위치를 결정)
      return data.fieldData;
   }
});
```

- **grid (for Alopex Grid)**

Alopex Grid 데이터를 자동 바인드하기 위한 설정입니다. 위 interface 에서 listData 가 grid 데이터라면 아래와 같이 바인드 할 데이터를 리턴합니다.

```
$a.request.setup({
   grid : function(elem, data) {
                var key;
                //통신 요청 시 데이터 바인드 키를 elem.key로 설정함.
                if (elem.key) {
                        //(data-bind="grid:gridData" 에서 `gridData`)
                        key = elem.key;
      } else {
        //grid element 데이터 바인드키를 key값으로 가지고 있어야함.
        var bindkey = $(elem).attr('data-bind') ?
           ($.trim($(elem).attr('data-bind').replace(/\s*grid\s*:/gi, ''))) : undefined;
        key = bindkey || elem.id;
      }
      if (!data.listData[key]) {
        data.listData[key] = {};
      }
      return {
        list: data.listData[key].list,    //현재 선택된 페이지의 그리드 데이터
        currentPage: data.listData[key].currentPage, //현재 선택된 페이지
        perPage: data.listData[key].perPage,    //페이지 당 표시되는 최대 데이터 수
        currentLength: data.listData[key].currentLength, //현재 선택된 페이지 데이터 수
        totalLength : data.listData[key].totalLength        //전체 데이터 수
      };
        }
});
```

6.3.1.4. NEXCORE J2EE 통신 인터페이스

기본 통신 인터페이스 외에 새로운 플랫폼의 인터페이스를 정의 할 수 있습니다. 아래는 NEXCORE J2EE 통신 인터페이스를 설정하는 방법입니다.

- **플랫폼 지정**

platform 에 'NEXCORE.J2EE'를 지정합니다.

```
$a.request.setup({
    platform: 'NEXCORE.J2EE',
        url : function(id, option) {
      return 'http://ui.alopex.io/' + id;
    },
    async: 'true',
    method: 'post',
    timeout: 30000
});
```

아래와 같은 통신 인터페이스는 빌트인으로 제공합니다. 따라서 사용자가 인터페이스를 별도로 정의하지 않으면 아래의 설정에 따라 통신합니다.

```
$a.request.setup('NEXCORE.J2EE', {
    interface: {
      dataSet: {
        message: {},
        fields: {},
        recordSets: {}
      },
      transaction: {},
      attributes: {}
    },
    object: function(elem, data) {
      return data.dataSet.fields;
    },
    grid: function(elem, data) {
      var key;
      if ($.alopex.util.isValid(elem.key)) {
        key = elem.key;
      } else {
        var bindkey = $(elem).attr('data-bind')?
            ($.trim($(elem).attr('data-bind').replace(/\s*grid\s*:/gi, ''))) : undefined;
        key = bindkey || elem.id;
      }
      if (!$.alopex.util.isValid(data.dataSet.recordSets[key])) {
        data.dataSet.recordSets[key] = {};
      }
      return {
        list: data.dataSet.recordSets[key].nc_list,
        currentPage: data.dataSet.recordSets[key].nc_pageNo,
        perPage: data.dataSet.recordSets[key].nc_recordCountPerPage,
        currentLength: data.dataSet.recordSets[key].nc_recordCount,
        totalLength : data.dataSet.recordSets[key].nc_totalRecordCount
      };
    },
    before: function(id, option) {
      // 헤더 추가.
      this.requestHeaders["Content-Type"] ="application/json; charset=UTF-8";
      this.data.transaction.id = id;
    }
});
```

- **통신 인터페이스 변경**

빌트인으로 제공하는 인터페이스를 변경하려면 공통 셋업 자바스크립트에서 아래와 같이 사용자 인터페이스를 정의합니다.

```
$a.request.setup('NEXCORE.J2EE', {
   //사용자 인터페이스 정의
        interface: {…},
        object: function(elem, data) {…},
        grid: function(elem, data) {…},
        before: function(id, option) {…},
        after: function(res) {…},
        success: function(res) {…},
        fail: function(res) {…},
       error: function(res) {…}
});
```

NEXCORE J2EE 통신 인터페이스에 관한 자세한 가이드는 홈페이지의 NEXCORE J2EE 통신 인터페이스의 플랫폼 사용예제를 참고하세요.

6.3.1.5. NEXCORE .NET 통신 인터페이스

아래는 NEXCORE .NET 통신 인터페이스를 설정하는 방법입니다.

- **플랫폼 지정**

platform 에 'NEXCORE.NET 를 지정합니다.

```
$a.request.setup({
   platform: 'NEXCORE.NET'
});
```

NEXCORE.NET 역시 아래와 같은 통신 인터페이스를 빌트인으로 제공합니다.

따라서 사용자가 인터페이스를 별도로 정의하지 않으면 아래의 설정에 따라 통신합니다.

```
$a.request.setup('NEXCORE.NET', { // platform
      interface : {
         request : {
            ServiceType : {}, // 서비스 타입
            ServiceName : {}, // 서비스 명
            ServiceData : {
               DataSet : {
                  DataSetName : {},
                  Tables : [] // grid 데이터
               },
               Hashtable: {} // form 데이터
            }
         }
      },
      object : function(elem, data) {
         return data.request.ServiceData.Hashtable;
      },
      grid : function(elem, data) {
         var key;
         if ($.alopex.util.isValid(elem.key)) {
            key = elem.key;
         } else {
            key = getGridKey(elem);
         }
         var Tables = data.request.ServiceData.DataSet.Tables;
         var gridData = null;
         $.each(Tables, function(i, v){
            if(v.TableName === key) gridData = v;
         });
```

```
        if(gridData === null){
            gridData = { "TableName" : key };
            Tables.push(gridData);
        }
        return {
            list : gridData.Rows,
            currentPage : gridData.PageNumber,
            perPage : gridData.RowspPage,
            currentLength : gridData.RowsLength,
            totalLength : gridData.TotalRowsLength,
            setList: function(list) {
                gridData.Rows = list;
            },
            setCurrentPage: function(currentPage) {
                gridData.PageNumber = currentPage;
            },
            setperPage: function(perPage) {
                gridData.RowspPage = perPage;
            },
            setcurrentLength: function(currentLength) {
                gridData.RowsLength = currentLength;
            },
            settotalLength: function(totalLength) {
                gridData.TotalRowsLength = totalLength;
            }
        };
    }
});
```

6.3.2. 컴포넌트 셋업

Alopex 컴포넌트의 글로벌 옵션을 설정할 수 있도록 $a.setup 함수를 제공합니다.

```
$a.setup( componentName, options )
```

- **parameters**
  - ✓ componentName {string} : Alopex UI 컴포넌트 이름
  - ✓ options {object} : 해당 컴포넌트의 옵션

아래는 $a.setup 함수를 이용하여 datepicker 와 dialog 의 공통 옵션을 셋업 하는 예제입니다.

```
// datepicker 컴포넌트 셋업
$a.setup('datepicker', {
    selectyear: true,
    selecmonth: true,
    showothermonth: true,
    showbottom: true,
    locale: 'en'
});

// tree 컴포넌트 셋업
$a.setup('tree', {
        idKey : 'code',
        textKey :'title',
    });

// dialog 컴포넌트 셋업
$a.setup('dialog', {
    modal: true,
    movable: true
});
```

컴포넌트 셋업의 우선순위는 개별 페이지에서 설정한 셋업이 공통 페이지에서 셋업보다 우선합니다.

6.3.3. 페이지 이동 셋업

페이지 이동에 사용되는 $a.navigate 함수 URL 파라미터의 처리를 글로벌하게 셋업하여 개별 페이지에서 화면이동 API 사용시, URL 을 아이디 형태로 전달 가능합니다.

```
$a.navigate.setup({
   url: function(url, param) {
      reutrn '/html/' + url + '.html';
   }
});

// 개별 페이지에서 API 호출
$a.navigate('customer/list');
```

6.3.4. 팝업 셋업

화면이동에 사용되는 $a.navigate 함수 URL 파라미터의 처리를 글로벌하게 셋업하여 개별 페이지에서 화면이동 API 사용시, URL 을 아이디 형태로 전달 가능합니다.

```
$a.popup.setup({
   url: function(url, param) {
      reutrn '/html/' + url + '.html';
   }
});

// 개별 페이지에서 API 호출
$a.popup({
        url: 'popup.html',
        callback: function(){...}
});
```

6.3.5. 메시지 처리

페이지에서 공통적으로 사용하는 메시지는 모듈용 Javascript 파일에서 처리합니다.

- **기본 Alert 사용**

Javascript 기본 alert 를 사용한 메시지 처리 방법입니다.

```
<common.js>

var common = $a.page(function(){

        this.init = function(){
                //초기화 루틴 수행
        };

        this.alert = function(message){
                alert(message);
        };

});
```

```
<setup.js>

$a.request.setup({
   fail: function(res) {
            //실패 메시지 처리
            common.alert('서버에서 처리 중 오류가 발생했습니다.');
      }
})
```

- **팝업 사용**

Alopex 팝업 컴퍼넌트를 사용한 방법입니다. 팝업에 대한 자세한 내용은 팝업 가이드를 참고하세요.

```
<popup.html>

var common = $a.page(function(){

      this.init = function(id, param){
            var message = param.message;
            $('#textarea_errorMessage').val(message);
      };

});

<textarea class="Textarea" id="textarea_errorMessage"></textarea>
```

```
<setup.js>

$a.request.setup({
   fail: function(res) {
            //실패 메시지 처리
            $a.popup({
                  url : 'popup.html,
                  data : {
                        message : '서버에서 처리 중 오류가 발생했습니다.',
                  },
                  …
            });
      }
})
```

6.3.6. 컴포넌트 확장

Alopex 에서 기본으로 제공하는 컴포넌트를 확장해 새롭게 컴포넌트를 생성할 수 있습니다.

6.3.6.1. 컴포넌트 상속

컴포넌트 확장은 상속을 통해 구현합니다. 상속은 $a.inherit 함수를 사용합니다. 이 함수는 Alopex 에서 사용되는 내부 함수로 상속 관계를 정의할 때 사용합니다.

```
$a.inherit(parent, child)
```

- **parameter**
  - ✓ parent : 상속 받을 부모 객체 혹은 함수를 선언합니다.
  - ✓ child : 상속 대상

- **return**

  상속 받은 자바스크립트 object.

6.3.6.2. 컴포넌트 확장

Alopex 에서 제공하는 컴포넌트를 상속받아 기능을 확장할 수 있습니다. 아래 코드는 $a.inherit 함수를 통해 textinput 컴포넌트를 상속받고 연도를 입력하는 텍스트 element 를 생성하는 기능을 확장한 예제입니다.

```
$a.widget.year = $a.inherit($a.widget.textinput, {
   widgetName: 'year',
   setters: ['year'],

   init : function(el, options) {
      var $el = $(el);
      $el.attr('type', 'text').attr('class', 'yearInput')
         .attr('maxlength', '4')
         .attr('data-validate-rule', '{year:true}')
         .attr('data-keyfilter-rule', 'digits')
         .attr('placeholder', 'yyyy');
   }
});
```

6.3.6.3. 사용자 정의 컴포넌트

사용자 정의 컴포넌트를 만들어 사용할 때는 아래 코드와 같이 컴포넌트를 상속합니다.

```
$a.widget.newcomponent = $a.inherit($a.widget.object, {
   widgetName: 'newcomponent', // HTML 마크업의 class 속성에 들어갈 컴포넌트 명을 입력합니다.
        //컴포넌트의 setter 함수를 설정하는 곳입니다. 단, 맨 처음에는 새롭게 추가된 widgetName을 꼭 써주셔야합니다.
   setters: ['newcomponent', ...],
   getters: ['getStyle'...], // 컴포넌트의 getter 함수를 설정하는 곳입니다.

        // 새로운 컴포넌트의 동작이나 마크업등을 설정하는 부분입니다. 사용자는 $el을 이용하여 커스텀하게 마크업,
스타일등을 만들어낼 수 있습니다.
   init: function(el, options) {
      var $el = $(el);
      ...
   }

   ... // 기타 settter/getter 함수를 작성 합니다.
});
```

- **widgetName**

  새로 정의할 컴포넌트 이름.

- **setters**

  새로 정의한 컴포넌트의 setter 함수. 단, setters 의 맨 첫 번째에는 컴포넌트 이름을 입력해야합니다.

- **getters**

  새로 정의한 컴포넌트의 getter 함수.

- **init**

컴포넌트가 화면에 렌더링될때 컴포넌트를 초기화하는 함수

- **parameter**
  - ✓ el: HTML element
  - ✓ options: 자바스크립트 호출시 적용된 옵션 값들
- **위의 속성 외에 setters/getters 에 선언한 컴포넌트의 API 를 함수 형태로 정의합니다.**

6.3.6.4. 주의사항

컴포넌트 구현 시 확장에 사용된 스크립트는 반드시 Alopex UI 스크립트가 먼저 실행된 뒤에 실행되어야 합니다. 이는 컴포넌트가 $a.widget.object 를 상속받아서 만들어지기 때문에 제약사항을 반드시 지켜주어야 합니다.

- **스크립트 실행 순서**

  Alopex UI > Alopex UI extension > custom javascript

6.4. 페이지

$a.page 함수는 웹 페이지 로딩에 대한 초기화 시점을 보장하여 Alopex UI 컴포넌트와 애플리케이션 자바스크립트 함수를 규격화된 형태로 작성할 수 있도록 하는 페이지 컨트롤러 함수입니다.

6.4.1. 초기화

$a.page 로 전달하는 함수의 init 함수는 화면이 로드되고 처음으로 호출되어 화면을 초기화하는 함수입니다. init 함수에서는 아래와 같은 사항이 보장됩니다.

- ✓ 웹 페이지가 로드된 이후 호출
- ✓ Alopex UI 컴포넌트의 변환 작업이 완료된 이후 호출
- ✓ Alopex Runtime 적용 시, 네이티브 및 자바스크립트 모듈이 로드된 이후 호출
- ✓ tabs 컴포넌트의 동적 탭이 로드되고 Alopex UI 컴포넌트 변환이 완료된 이후
- ✓ 팝업창이 나타나고 내부의 Alopex UI 컴포넌트 변환이 완료된 이후 호출

코드 작성 규격은 아래와 같습니다. init 함수의 파라미터로는 id 와 param 이 전달되며, id 는 화면의 아이디를 의미하고, param 은 화면 이동 함수 또는 팝업 함수에서 전달되는 파라미터 데이터입니다.

```
$a.page(function() {
    // 초기화 함수
    this.init = function(id, param) {
        ...
    }
});
```

6.4.2. 전달된 파라미터 사용

$a.page 함수의 init 함수의 파라미터로(위 예제의 param) 아래와 같은 데이터를 전달 받습니다.

- ✓ $a.navigate 함수의 파라미터 데이터
- ✓ $a.back 함수의 결과 데이터
- ✓ $a.popup 함수의 파라미터 데이터

6.4.3. 화면 Javascript 파일

- ✓ $a.page 의 함수에 function 을 파라미터로 넣어서 초기화 함수를 작성합니다.
- ✓ 내부 함수(사용자 함수)는 function 으로 생성합니다.

```
<page1.js>

$a.page(function(){

        this.init = function(id, param){
                //초기화 루틴 수행
                func1();
        };

        function func1(){
                //내부 함수
        }

});
```

6.4.4. 모듈용 Javascript 파일

- ✓ $a.page 의 함수에 function 을 파라미터로 넣어서 초기화 함수를 작성합니다.
- ✓ 파일명과 모듈(변수)명은 항상 일치시킵니다(module1.js – var module1)
- ✓ 외부에서 사용할 함수는 this 에 등록하고, 내부 함수는 function 으로 생성합니다.
- ✓ $a.page 를 리턴 받아 글로벌 변수에 등록하고 이를 페이지 등에서 활용합니다.

```
<module1.js>

var module1 = $a.page(function(){

        this.init = function(){
                //초기화 루틴 수행
        };

        this.func2 = function(){
                //외부 함수
        };

        function func1(){
                //내부 함수
        }

});

<page1.js>

$a.page(function(){
        this.init = function(){
                //module1의 외부 함수 호출
                module1.func2();
        };
});
```

6.5. 동적 페이지 구성(for Handlebars Java)

Top, Left, Bottom 등의 공통 메뉴, 메시지, 국제화, 권한 처리를 동적으로 구성하기 위한 가이드입니다. 자세한 사용 방법은 홈페이지의 Handlebars 가이드 를 참고하세요.

6.5.1. 메뉴 처리

외부 html 파일을 삽입하기 위해서는 include 태그를 사용합니다. 삽입을 원하는 페이지의 경로에 있는 파일이 태그 위치에 삽입되어 하나의 파일 형태로 동작합니다.

- **폴더 구조**

```
html
 └── common
          └──header.html
 └── xyz
     └──demo.html
```

- **상대 경로**

  현재 페이지를 기준으로 상대 경로를 입력합니다.

```
<demo.html>
<html>
      {{include "../common/header.html"}} //공통 헤더 영역 include
      <body>…</body>
<html>
```

- **절대 경로**

  html 폴더를 기준으로 절대 경로를 입력합니다.

```
<demo.html>
<html>
      {{include "/common/header.html"}} //공통 헤더 영역 include
      <body>…</body>
<html>
```

6.5.2. 국제화

국제화가 필요한 메시지 부분에 i18n 태그를 사용하여 처리합니다.

- **메세지 설정**

  NEXCORE J2EE(이하 J2EE)의 메시지 파일에 다국어 처리할 메시지를 설정합니다. 자세한 사용 방법은 J2EE 가이드를 참고하세요.

```
<en>
NAME=name

<ko>
NAME=이름
```

- **메세지 사용**

```
<h2>{{i18n "NAME"}}</h2>

<결과>
locale=ko_KR 일경우 => <h2>이름</h2>
locale=en_US 일경우 => <h2>name</h2>
```

- **Locale 변경**

  Locale 정보는 일반적으로 로그인 서비스에서 J2EE 의 UserInfo 객체에 저장합니다. 이후 동적으로 locale 정보를 변경하려면 locale 을 변경하는 서비스를 J2EE 에 생성해야 합니다. 자세한 생성 방법은 J2EE 가이드를 참고하세요.

- **주의사항**

  Locale 정보는 로그인이 되어있는 동안에만 유지된다는 점을 주의해야 합니다.

6.5.3. 권한처리

사용자의 권한에 따른 화면 처리는 acl 태크를 사용합니다.

- **권한처리 설정**

  J2EE 의 UserInfo 객체에 권한 설정을 합니다. ACL 을 Key 로 권한을 Value 로 저장합니다. 일반적으로 로그인 서비스에서 처리합니다. 자세한 사용 방법은 J2EE 가이드를 참고하세요

```
UserInfo.put("ACL", "admin");
```

- **권한처리 사용**

  acl 태그를 사용하여 admin 사용자일 경우에만 삭제 버튼을 표시하는 예제 코드입니다.

```
{{#acl "admin"}}
        <button class=”Button” type=”button”>{{i18n "DELETE"}}</button>
{{/acl}}
```

6.6. 페이지 레이아웃

6.6.1. 탭 페이지 구성

Alopex 의 Tabs 컴퍼넌트를 사용해 탭 페이지를 구성할 수 있습니다.

- **탭 레이아웃 구성**

  아래 코드와 같이 탭 리스트와 각각의 탭에 들어갈 컨텐츠를 작성하면됩니다.

```
<div class="Tabs">
  <ul>
    <li data-content="#tab1">tab1</li>
    <li data-content="#tab2">tab2</li>
    <li data-content="#tab3">tab3</li>
```

```
  </ul>
  <div id="tab1">
    <h3>tab1</h3>
      tab1 내용
    </div>
  <div id="tab2">
    <h3>tab2</h3>
    tab2 내용
  </div>
  <div id="tab3">
    <h3>tab3</h3>
    tab3 내용
  </div>
</div>
```

- **이벤트 처리**

  선택된 탭이 바뀌었을 때 이벤트를 처리할 수 있습니다.

```
$('.Tabs').on('tabchange', function(e, index, index2){
  // e : event object
  // index : 선택된 Depth1 탭 인덱스
  // index2 : 선택된 Depth2 탭 인덱스(Depth2 탭이 있을 경우에 전달)
});
```

- **동적으로 탭 변경**

  동적으로 특정 인덱스의 탭을 선택할 수 있습니다.

```
$('.Tabs').setTabIndex(index); // 선택될 Depth1 탭의 인덱스
$('.Tabs').setTabIndex([index1,index2]) // Depth2 탭이 있을 경우, 해당 위치의 Depth1, Depth2 탭 인덱스를 포함하는 배열
```

6.6.2. 페이지 분할(splitter)

스플리터(splitter)는 가로 또는 세로 방향으로 크기 조정 가능한 분할 윈도우를 구성할 수 있는 컴포넌트입니다. 자세한 내용은 확장 컴포넌트의 스플리터(splitter)를 참고하세요.

6.6.3. Layout Grid

Alopex UI 에서 제공하는 Theme 에는 화면 Layout 을 구성해주는 Layout Grid 스타일을 제공합니다. 화면을 기본 12 등분 기준으로 나누고, 목적에 맞게 영역을 구성을 할 수 있습니다.

6.7. 컴포넌트 사용

6.7.1. 컴포넌트 생성

Alopex UI 의 컴포넌트는 HTML 태그에 class 속성으로 컴포넌트 이름을 명시하여, 자바스크립트 코드 없이 바로 컴포넌트를 생성/사용할 수 있습니다. 아래의 예시는 dateinput 컴포넌트의 생성입니다. class 속성에 Dateinput 값을 넣고 마크업을 작성한 것 만으로도 날짜를 입력할 수 있는 dateinput 컴포넌트가 작동합니다. class 속성에 사용되는 컴포넌트명은 대문자로 시작합니다.

```
<div class="Dateinput" id="date1">
```

```
    <input>
</div>
```

6.7.2. 컴포넌트 옵션

컴포넌트는 각기 사용 가능한 옵션과 API 가 존재합니다. 컴포넌트별 옵션은 data- 로 시작하는 속성을 태그에 입력하여 적용할 수 있습니다. 위에서 생성한 dateinput 컴포넌트의 입력 형식을 바꾸고자 한다면 dateinput 컴포넌트의 data-format 옵션을 사용합니다.

```
<div class="Dateinput" data-format="MM/dd/yyyy" id="date0">
    <input>
</div>
```

6.7.3. 컴포넌트 API 호출

Javascript 코드를 이용하여 컴포넌트의 API 를 호출할 수 있습니다. 위의 dateinput 컴포넌트의 clear API 를 호출하고자 하면 jQuery 를 이용하여 dateinput 컴포넌트가 생성된 태그를 선택하고 .clear() 를 호출합니다.

```
<page.html>

<div id="date1" class="Dateinput">
    <input>
</div>
<button id="btn_clear" class="Button">clear API 호출</button>


<page.js>

$("#btn_clear").on("click", function (){
    $('#date1').clear();
});
```

6.7.4. 컴포넌트 셋업

각 컴포넌트에서 사용되는 옵션 중 글로벌로 적용해서 사용하고자 하는 옵션이 있을 때 $a.setup 함수를 사용합니다. 자세한 내용은 컴포넌트 셋업 가이드를 참고하세요.

6.8. 확장 컴포넌트

Alopex UI 컴포넌트 중 다음의 컴포넌트는 오픈소스를 wrapping 한 구조로 되어 있습니다 이들을 사용하기 위해서는 아래의 2 가지 javascript 와 1 개의 CSS 파일 링크가 필요합니다.

- ✓ 멀티셀렉트(MultiSelect) 컴포넌트
- ✓ 분할(Splitter) 컴포넌트
- ✓ 파일업로드(FileUpload) 컴포넌트

6.8.1. 필요 파일

```
…
<script src="/web/script/lib/alopex/alopex-ui.js"></script>
<script src="/web/script/lib/alopex/src/alopex-ext.js"></script>
<!-ext setup은 alopex-ui 다음에 위치해야 함-→
<script src="/web/script/lib/alopex/src/alopex-ext-setup.js"></script>

<link rel="stylesheet" href="/web/css/lib/alopex/src/alopex-ext.css">
```

유형	file	설명
javascript	alopex-ext.js	jquery-ui.js, jquery-ui-multiselect.js jquery-ui-multiselect-filter.js jquery-splitter.js jquery.uploadfile.js 가 merge 된 파일
	alopex-ext-setup.js	MultiSelect / Splitter / FileUpload 에 대한 widget 확장 셋업
css	alopex-ext.css	MultiSelect / Splitter / FileUpload 에 대한 스타일링

6.8.2. 멀티셀렉트(MultiSelect)

멀티셀렉트는 셀렉트 박스의 다수의 option 을 체크박스로 선택할 수 있는 컴포넌트입니다.

- **셋업(공통)**

  setup 자바스크립트에서 멀티셀렉트의 기본 속성을 공통으로 설정합니다.

```
$a.setup('multiSelect', {
   multiple : true,
   noneSelectedText : '선택하세요',
   header : true,
   minWidth : 200,
   selectedList : 2,
   checkAllText : '전체선택',
   uncheckAllText : '전체해제',
   selectedText : '#개 선택됨',
   filter : true,
   label : '필터',
   placeholder : '검색어를 입력하세요',
   checkedheader : true,
   menuWidth : 'auto'
});
```

주요 속성에 대한 설명입니다.

Property	default	설명
header	true	전체선택/헤제, 닫기 아이콘 등 포함한 헤더 표시 여부
height	175	체크박스 컨테이너의 px 높이
minWidth	180	위젯의 최소 폭. 'auto' 로 세팅하면 자동 너비
checkAllText	전체선택	전체 선택의 텍스트
uncheckAllText	전체해제	선택 해제의 텍스트

noneSelectedText	선택하세요	아무것도 선택되지 않았을 때의 텍스트
selectedText	#개 선택됨	선택된 값들의 표시 방법
selectedList	2	해당 개수까지만 선택된 값에 나열하여 보여주고, 그 이상의 개수는 selectedText 로 표현
show	(null)	Open 시 효과 지정. ['slide', 500]
hide	(null)	Close 시 효과 지정. ['explode', 500]
autoOpen	false	초기화 시 자동 open 옵션
multiple	true	false 일 경우 checkbox 대신 radio 로 표시됨
classes	MultiSelect	자동으로 삽입될 커스텀 클래스명을 명시
filter	true	텍스트 박스 필터링 여부
label	필터	필터링 텍스트 앞쪽에 붙는 레이블
placeholder	검색어를 입력하세요	필터링 텍스트 박스의 기본 표시 문구
checkedheader	true	header 에 checkAll 영역 노출여부
menuWidth	(null)	리스트 영역의 폭. 'auto' 로 세팅하면 자동 너비

- **멀티셀렉트 컴포넌트 스타일링**

  멀티셀렉트 컴포넌트의 마크업 구조는 아래와 같으며 커스텀 스타일링을 위해서 CSS 를 수정합니다.

```
<div class="ui-multiselect-menu ui-widget ui-widget-content ui-corner-all">
	<div class="ui-widget-header ui-corner-all ui-multiselect-header ui-helper-clearfix">
		<ul class="ui-helper-reset">
		<li>
			<a class="ui-multiselect-all" href="#">
				<span class="ui-icon ui-icon-check"></span>
				<span>전체선택</span>
			</a>
		</li>
		<li>
			<a class="ui-multiselect-none" href="#">
				<span class="ui-icon ui-icon-closethick"></span>
				<span>전체해제</span>
			</a>
		</li>
		<li class="ui-multiselect-close">
			<a href="#" class="ui-multiselect-close">
				<span class="ui-icon ui-icon-circle-close"></span>
			</a>
		</li>
	</ul>
	</div>
	<ul class="ui-multiselect-checkboxes ui-helper-reset">
	<li class=" ">
		<label class="ui-corner-all ui-state-hover">
		<input type="checkbox" value="CMOS">
	<span>CMOS</span>
		</label>
	</li>
	</ul>
</div>
```

아래는 주요 css 클래스에 대한 설명입니다.

class	설명
ui-multiselect	여백, 텍스트 정렬 스타일링
ui-multiselect-menu	멀티셀렉트 팝업 창의 여백, z-index 조정
ui-multiselect-header	멀티셀렉트 여백 스타일링
ui-multiselect-checkboxes	Option 의 스타일링
ui-widget-header	헤더의 색상, 폰트, 테두리
ui-widget-content	멀티셀렉트 팝업의 테두리, 배경색, 폰트색 지정
ui-icon	아이콘 스타일링
ui-icon-check	전체선택 체크하는 V 버튼
ui-icon-closethick	전체선택 해제하는 X 버튼
ui-icon-circle-close	멀티셀렉트를 close 하는 둥근 X 표시 아이콘

- alopex-ext.css 의 주석을 확인하여 이를 스타일링하면 됩니다.

- **화면 HTML 에서의 사용 방법**

  HTML 에서는 MultiSelect 라는 클래스만 명시하면, 멀티셀렉트 컴포넌트로 생성됩니다.

```
<select id="familySelect" class="MultiSelect">
</select>
```

- **화면 Javascript 에서의 사용 방법**

  멀티셀렉트 내 시점 별로 이벤트 핸들러를 제공합니다.

```
$('#my-select).multiselect({
        beforeopen: function(){
            //멀티셀렉트 표시 직전 이벤트
        },
        open: function(){
            //멀티셀렉트 표시 시 이벤트
        },
        beforeclose: function(){
            //닫기 직전 이벤트
        },
        close: function(){
            //닫기 시 이벤트
        },
        checkall: function(){
            //전체선택 시 이벤트
        },
        uncheckall: function(){
            //전체선택해제 시 이벤트
        },
        click: function(){
            //체크박스 선택 시 이벤트
        }
});
```

6.8.3. 스플리터(Splitter)

스플리터(splitter)는 가로 또는 세로 방향으로 크기 조정 가능한 분할 윈도우를 구성할 수 있는 컴포넌트입니다.

- **셋업(공통)**

```
$a.setup('splitter', {
        position: '50%',
        limit: 10,
        orientation: 'horizontal'
});
```

Property	default	설명
position	50%	Split 비율. %가 없을 경우 px 로 인식.
limit	10	여백 pixel 값
orientation	horizontal	split 방향(horizontal|vertical)

- **스플리터 컴포넌트 스타일링**

  alopex-ext.css 에서 스플리터에 대한 스타일을 작성합니다.

class	설명
.splitter_panel .vsplitter	세로 방향 스플리터 스타일
.splitter_panel .hsplitter	가로 방향 스플리터 스타일

- **화면 HTML 에서의 사용 방법**

```
<div class="Splitter" data-position="40%" data-orientation="horizontal" style="height:600px;">
        <div id="frame1" class="top_panel">
        상단 콘텐츠
        </div>
        <div id="frame2" class="bottom_panel">
                하단 콘텐츠
        </div>
</div>
```

유형	속성	설명
Parent window	class	Splitter 를 명시합니다.
	data-position	%단위로 분할 비율을 설정 %가 없을 경우 pixel 로 인식
	data-orientation	horizontal | vertical
Child Frame	class	horizontal 분할일 경우 - top_panel: 상단 패널 - bottom_panel: 하단 패널 vertical 분할일 경우 - left_panel: 좌측 패널 - right_panel: 우측 패널

- **화면 Javascript 에서의 사용 방법**

  동적으로 split 을 하고자 할 경우 javascript 에서 사용 가능합니다.

```
$('#foo').split({
   orientation: 'horizontal',
   limit: 10,
   position: '50%'
});
```

6.8.4. 파일업로드(FileUpload)

파일업로드(FileUpload) 컴포넌트는 로컬에 있는 파일을 서버로 업로드 할 수 있는 컴포넌트입니다.

IE10 이상에서 정상적으로 사용할 수 있습니다..

- **셋업(공통)**

```
$a.setup('fileupload', {
   url : 'http://localhost/upload',
   fileName : ‘iploadFiles’
});
```

주요 속성에 대한 설명입니다.

Property	default	설명
url	(null)	파일 업로드를 수행 할 서버 URL 을 지정할 때 사용합니다
method	POST	ajax 통신의 메소드를 지정하고자 할 때 사용합니다. POST | GET
enctype	multipart/form-data	form 의 인코딩 방식을 지정합니다.
formData	false	파일과 함께 전송되어야 할 form data 를 지정합니다. formData : { key1: 'value1', key2: 'value2' }
dynamicFormData	false	파일과 함께 전송되어야 할 form data 를 동적으로 지정합니다.
sequential	true	순차적인 파일 업로드 여부를 지정합니다.
sequentialCount	1	sequential 값이 true 로 설정하면 , sequentialCount 값에 따라 순차적으로 업로드할 파일 갯수를 지정합니다.
maxFileSize	-1	최대 허용 파일 용량을 Byte 단위로 설정합니다.
maxFileCount	-1	최대 허용 파일 갯수를 설정합니다. advance 모드에서 사용 가능합니다.
returnType	(null)	서버에서 반환되는 데이터 형식을 지정할 때 사용합니다. ex) xml | json | script
allowedTypes	*	허용할 파일 확장자를 지정합니다. 여러 확장자를 지정할 경우, 콤마로 구분합니다. ex) "jpg,png,gif"
acceptFiles	*	파일 브라우저에서 허용할 MINE 타입을 지정합니다. ex) "*" | "image/"
fileName	uploadFiles	전송 시 사용 될 input 요소의 이름을 지정합니다.
dragDrop	false	Drag & Drop 기능 사용 여부를 지정합니다.

		advance 모드에서 사용 가능합니다.
autoSubmit	false	파일 선택 시 자동으로 업로드 전송 여부를 지정합니다.
showCancel	true	파일 상태 영역에서 '취소' 버튼의 사용 여부를 지정합니다.
showAbort	true	파일 상태 영역에서 전송 '중단' 버튼의 사용 여부를 지정합니다.
showDone	true	파일 상태 영역에서 전송 '완료' 버튼의 사용 여부를 지정합니다.
showDelete	true	파일 상태 영역에서 업로드 파일 '삭제' 버튼의 사용 여부를 지정합니다. advance 모드에서 사용 가능합니다.
showDownload	true	파일 상태 영역에서 업로드 파일 '다운로드' 버튼의 사용 여부를 지정합니다. advance 모드에서 사용 가능합니다.
showStatusAfterSuccess	true	파일 상태 영역에서 전송 완료된 파일 내역 표시 여부를 지정합니다. true 일 경우, showDone 과 showDelete 값에 의해 해당 기능 버튼이 보여집니다. advance 모드에서 사용 가능합니다.
showFileCounter	false	파일명에 idnex 숫자 표시 여부를 지정합니다.
showFileSize	true	파일명에 파일 사이즈 표시 여부를 지정합니다.
showPreview	true	파일 상태 영역에서 이미지 프리뷰의 사용 여부를 지정합니다.
onLoad	(null)	파일 업로드 컴포넌트가 로드 된 시점에서 처리할 이벤트 함수를 넣어줍니다.
onSelect	(null)	파일 선택 시점에서 처리할 이벤트 함수를 넣어줍니다.
onSubmit	(null)	파일 업로드 수행 시점에서 처리할 이벤트 함수를 넣어줍니다.
onSuccess	(null)	업로드가 성공 된 시점에서 처리할 이벤트 함수를 넣어줍니다.
afterUploadAll	(null)	모든 파일의 업로드가 성공 된 시점에서 처리할 이벤트 함수를 넣어줍니다.
onError	(null)	파일 선택 및 업로드가 실패 된 시점에서 처리할 이벤트 함수를 넣어줍니다.
onCancel	(null)	사용자가 파일을 삭제(취소)한 시점에서 처리할 이벤트 함수를 넣어줍니다.
deleteCallback	(null)	사용자가 업로드한 파일의 '삭제' 버튼을 클릭한 시점에서 처리할 이벤트 함수를 넣어줍니다.
downloadCallback	(null)	사용자가 업로드한 파일의 '다운로드' 버튼을 클릭한 시점에서 처리할 이벤트 함수를 넣어줍니다.

- **파일업로드 컴포넌트 스타일링**

  alopex-ext.css 에서 파일업로드에 대한 스타일을 작성합니다.

class	설명
.preview-container	파일 목록을 담고 있는 컨테이너 스타일
.preview-list	개별 파일 정보를 표시하는 상위 컨테이너 스타일
.preview-contents	개별 파일 정보를 표시하는 하위 컨테이너 스타일

- **화면 HTML 에서의 사용 방법**

```
<Single 파일업로드>
<div id="fileuploaderA" class="Fileupload" data-selectType="basic" ></div>

<Multi 파일업로드>
<div id="fileuploaderB" class="Fileupload" data-selectType="advance" ></div>
```

유형	속성	설명
Single 파일업로드	class	Fileupload 를 명시합니다.
	data-selectType	"basic" 을 명시하여 한 개의 파일을 업로드합니다.
Multi 파일업로드	class	Fileupload 를 명시합니다.
	data-selectType	"Advance" 를 명시하여 여러 파일을 업로드합니다.

6.9. 컴포넌트 간 연동

컴포넌트 간 연동은 하나의 select box option 을 변경했을 때 다른 select box 의 option 에 들어갈 데이터를 변경하는 경우 등이 있습니다. 이 경우 컴포넌트에 change 이벤트를 바인드해 처리할 수 있습니다.

```
<script>
	$a.page(function() {
		this.init = function() {
			$('#language').change(function(){
				var selectedValue = $('#language').getValues()[0],
					$menu = $('#menu');

				if(selectedValue === 'korea'){
					$menu.html('<option value="menu1">메뉴1</option>' +
							'<option value="menu2">메뉴2</option>' +
							'<option value="menu3">메뉴3</option>');
				}else if(selectedValue === 'usa'){
					$menu.html('<option value="menu1">Menu1</option>' +
							'<option value="menu2">Menu2</option>' +
							'<option value="menu3">Menu3</option>');
				}
			});
		};
	});
</script>

<select id="language" class="Select" >
	<option value="korea">Korea</option>
```

```
        <option value="usa">USA</option>
        <option value="japan">Japan</option>
        <option value="china">China</option>
</select>
<select id="menu" class="Select" >
        <option value="menu1">메뉴1</option>
        <option value="menu2">메뉴2</option>
        <option value="menu3">메뉴3</option>
</select>
```

- **language select box 에서 USA option 을 선택한 경우 menu select box 의 option 이 영문으로 변경됩니다.**

```
<select id="menu" class="Select">
        <option value="menu1">Menu1</option>
        <option value="menu2">Menu2</option>
        <option value="menu3">Menu3</option>
</select>
```

6.10. 페이지 이동

Alopex UI 에서는 페이지 이동을 위한 Javascript API 를 제공합니다. 화면이동 API 를 이용하여 클라이언트 영역 내에서 화면 이동은 물론 데이터 전달이 가능합니다. 자세한 내용은 홈페이지의 Navigation API 를 참고하세요

- **화면이동**

```
$a.navigate('customer/list');
```

- **화면이동과 함께 데이터 전달**

```
$a.navigate('customer/list', { key : value });
```

6.11. 팝업

Alopex 는 윈도우 팝업, 레이어 팝업을 $a.popup 하나의 API 로 호출할 수 있도록 제공합니다. 자세한 내용은 홈페이지의 Popup API 를 참고하세요.

6.11.1. 팝업 종류

- **레이어 팝업(iframe 사용)**

  $a.popup API 의 url 파라미터로 전달된 화면을 레이어 팝업으로 표시합니다. 기본적으로 <iframe> 태그를 사용하여 새로운 창을 띄운 것과 동일한 효과를 가집니다. 팝업으로 표시하려는 화면과 기존 화면간에 마크업 요소가 중복되거나 자바스크립트 변수/함수 명칭이 중복될 수 있다면 iframe 을 이용한 팝업 호출이 적합합니다.

```
$a.popup({
  // 팝업에 표시될 HTML
  url: 'popup.html',

  // 팝업이 닫힐 때 호출되는 콜백 함수
  callback: closeCallback
});
```

```
function closeCallback() {}
```

- **레이어 팝업(iframe 사용하지 않음)**

  $a.popup API 의 iframe 옵션을 false 로 지정하여, <iframe> 태그 대신 다이얼로그 컴포넌트의 콘텐츠 영역에 화면 마크업을 직접 불러올 수 있습니다.

  ✓ 여기서 주의할 점은 팝업 소스에 이미 로딩한 라이브러리(alopex-ui,JQuery 에 대한 js,css 등)를 다시 로딩하지 않도록 해야합니다.

```
$a.popup({
   url: 'popup.html',
   iframe: false
});
```

- **윈도우 팝업**

  $a.popup API 의 windowpopup 옵션을 true 로 지정하여 윈도우 팝업을 표시합니다. 이 경우 새로운 브라우저 창 또는 탭에 화면이 표시됩니다.

  윈도우 팝업을 사용하는 경우, W3C window.open API 에서 제공하는 옵션을 'other' 값에 정의하여 설정할 수 있습니다.

```
$a.popup({
   url: 'popup.html',
   windowpopup: true,
   other: 'width=1000,height=400,top=200,left=100,scrollbars=yes'

});
```

✓

6.11.2. 팝업 데이터 처리

- **팝업 화면으로 데이터 전달**

  팝업 화면으로 데이터를 전달하고자 하는 경우, $a.popup API 의 data 옵션을 통해 전달합니다.

```
$a.popup({
   url : 'popup.html',
   data : { key : value }
});
```

- **팝업 화면에서 데이터 사용하기**

  이 데이터는 팝업 화면의 $a.page.init 함수의 두번째 인자로 전달됩니다.

```
$a.page(function() {
   this.init = function(id, param) {
      // $a.popup() 이 넘겨받은 data를 param으로 전달.
   };
})
```

6.11.3. 팝업 결과값 전달

- **팝업 창을 닫을 때는 $a.close API 를 호출하여 팝업 창을 닫고, 팝업을 호출한 화면으로 결과 데이터를 전달합니다.**

```
$a.close(data)
```

- **팝업 창에서 전달한 데이터는 팝업을 호출한 화면의 $a.popup 의 옵션 파라미터로 전달된 callback 옵션의 인자로 전달됩니다.**

```
$a.popup({
   url : 'popup.html',
   callback: function (data) {
      console.log(data);
   }
});
```

6.12. 유효성 검증

사용자 입력 데이터의 유효성을 검증하기 위해 validation 컴포넌트를 사용합니다.

6.12.1. 점검규칙 설정

유효성 검증을 위한 점검규칙 설정 방법입니다.

- **태그 속성 사용**

  HTML 태그의 data-validation-rule 속성에 점검규칙을 설정합니다.

```
// validator 설정. 필수값, 최소길이4, 최대길이8
<input class="Textinput" id="inputfield" data-validation-rule="{ required: true , minlength: 4, maxlength: 8 }">
```

- **API 사용**

  validator 함수에 점검규칙을 설정합니다.

```
<script>
        $a.page(function(){
                this.init = function() {
                        // validator 설정. 필수값, 최소길이2, 최대길이10
                        $('#inputfield').validator({
                                rule : { required:true, minlength:2, maxlength:10}
                        });
                }
        });
</script>

<input class="Textinput" id="inputfield">
```

6.12.2. 검증 실행

설정한 점검규칙에 따라 검증을 실행하는 방법입니다.

- **validator 사용**

validator 함수를 호출하면 필드의 값이 수정될 때 마다 검증을 수행하며, 결과메시지를 지정한 노드에 기록합니다.

```
<script>
	$a.page(function(){
		this.init = function() {
			$('.Textinput').validator();
		};
	});
</script>

// validator 설정. 필수값, 최소길이4, 최대길이8
<input class="Textinput" id="inputfield" data-validation-rule="{ required: true , minlength: 4, maxlength: 8 }">

//검증 결과 메시지가 출력되는 노드 지정
<span data-for="inputfield" style="color:red;"></span>
```

결과 me 최소 4 글자 이상 입력하십시오.

```
<script>
	$a.page(function(){
		this.init = function() {
			// validator 설정. 필수값, 최소길이2, 최대길이10.
			$('#inputfield').validator({
				rule : { required:true, minlength:2, maxlength:10}
			});
		}
	});
</script>

<input class="Textinput" id="inputfield">

//검증 결과 메시지가 출력되는 노드 지정
<span data-for="inputfield" style="color:red;"></span>
```

결과 반드시 입력해야 하는 항목입니다.

- **validate 사용**

validate 함수를 호출하면 즉시 그리고 한 번만 검증을 실행하며, boolean 값을 return 합니다. 결과메시지를 지정한 노드에 기록합니다.

```
<script>
	$a.page(function(){
		this.init = function() {
			$('#button').bind('click', function (){
			  var result = $('#inputfield').validate();
			  alert(result);
			});
		};
	});
</script>

// validator 설정. 필수값, 최소길이4, 최대길이8
<input class="Textinput" id="inputfield" data-validation-rule="{ required: true , minlength: 4, maxlength: 8 }">

//검증 실행 버튼
```

```
<a class="Button" id="button">validation</a>

//검증 결과 메시지가 출력되는 노드 지정
<span data-for="inputfield" style="color:red;"></span>
```

```
<script>
	$a.page(function(){
		this.init = function() {
			// validator 설정. 필수값, 최소길이2, 최대길이10.
			$('#inputfield').validator({
				rule : { required:true, minlength:2, maxlength:10}
			});

			$('#button').bind('click', function (){
			   var result = $('#inputfield').validate();
			   alert(result);
			});
		}
	});
</script>

<input class="Textinput" id="inputfield">

//검증 실행 버튼
<a class="Button" id="button">validation</a>

//검증 결과 메시지가 출력되는 노드 지정
<span data-for="inputfield" style="color:red;"></span>
```

6.12.3. 룰 정의

유효성 검증하기 위해서는 검증을 위한 규칙(rule)이 존재해야 하는데 Alopex 에서는 검증에 사용되는 규칙을 method 라고 부르게 됩니다. 사용자는 룰을 명시하는 JSON 의 method 이름과, method 의 동작 방식을 명시하는 파라미터를 각기 key-value 쌍으로 입력함으로써 method 의 적용 여부를 명시할 수 있습니다.

6.12.3.1. 빌트인 룰

Alopex 에서 빌트인으로 제공하는 점검 규칙 룰입니다.

Method	파라미터	설명
required	ID Selector	해당 필드의 필수 여부. ID selector 를 넣은 경우 ID selector 에 해당되는 필드에 value 가 있어야 자신을...
minlength	number	해당 필드의 최소 문자 개수
maxlength	number	해당 필드의 최대 문자 개수
rangelength	[number, number]	해당 필드의 문자 개수가 주어진 수 안에 위치하는가 여부
minblength	number	해당 필드의 최소 바이트 길이
maxblength	number	해당 필드의 최대 바이트 길이
rangeblength	[number, number]	해당 필드의 길이가 주어진 바이트 길이 안에 위치하는가 여부

min	number	해당 필드가 가지는 value 의 최소값
max	number	해당 필드가 가지는 value 의 최대값
range	[number, number]	해당 필드가 가지는 value 값이 주어진 숫자 안에 위치하는가 여부
email	Boolean	이메일 형식을 지켰는가 여부
url	Boolean	url 형식을 지켰는가 여부(http://..., https://...)
date	Boolean	YYYY/MM/DD 또는 YYYY-MM-DD 형식을 지켰는가 여부
mindate	string, ID Selector	date 형식이면서, 특정 날짜 또는 날짜 이후의 날짜가 입력되었는가 여부
maxdate	string, ID Selector	date 형식이면서, 특정 날짜 또는 날짜 이전의 날짜가 입력되었는가 여부
daterange	[date, date]	date 형식이면서, 특정 날짜 사이의 값이 입력되었는지 여부
oneof	[...]	array 에 포함된 값들 중 하나가 입력되었는지 여부
number	boolean	정수값이 입력되었는가 여부
digits	boolean	숫자만(0~9) 입력이 되었는가 여부
alphabet	boolean	영문 알파벳만 입력이 되었는가 여부
equalTo	ID Selector,value	제시된 값과 동일한 값이 필드에 입력되었는가 여부. ID Selector 가 들어온 경우 select 된 필드와 값이 동일
numalpha	boolean	숫자 또는 알파벳만 입력되었는가 여부
nospace	boolean	스페이스를 허용하지 않는가 여부

6.12.3.2. 사용자 정의 룰

빌트인 룰 외에 사용자 룰을 정의할 수 있습니다. Validator 의 addMethod 함수를 사용합니다.

```
Validator.addMethod(String name, Function handler)
```

- **Parameter**
  - ✓ name {String} Required

    새로 추가하고자 하는 검증 method 의 이름입니다. 이 값을 앞으로 룰 안에서 사용할 수 있게 됩니다.
  - ✓ handler {Function} Required

    검증 method 의 실제 구현 함수입니다. handler 는 검증하고자 하는 input/select element, 필드의 value, 룰에 지정된 method 의 parameter 를 argument 로 받으며, 이 값들을 이용하여 검증을 통과할 경우 true 를, 검증을 통과하지 못 할 경우 false 를 리턴하도록 작성합니다.

```
//무조건 rule 파라미터로 넘어온 값과 일치해야 검증을 통과시킵니다.
Validator.addMethod('customEqual', function(element, value, param) {
   if(value === param) {
      return true;
   } else {
      return false;
   }
} );
```

```
<!-- 아래의 input은 testing이라는 텍스트가 들어와야만 검증을 통과하게 됩니다. -->
<input class="Textinput" data-validation-rule="{customEqual:'testing'}">
```

6.12.4. 오류메시지 정의

6.12.4.1. 빌트인 오류메시지

Validator 컴포넌트에서 기본적으로 제공하는 오류메시지입니다.

Method	기본 오류메시지
required	반드시 입력해야 하는 항목입니다.
minlength	최소 {0}글자 이상 입력하십시오.
maxlength	최대 {0}글자 까지 입력 가능합니다.
rangelength	{0}에서 {1} 글자 사이로 입력하십시오.
minblength	최소 {0}바이트 이상 입력하십시오.
maxblength	최대 {0}바이트 까지 입력 가능합니다.
rangeblength	{0}에서 {1} 바이트 사이로 입력하십시오.
min	최소 입력가능 값은 {0}입니다.
max	최대 입력가능 값은 {0}입니다.
range	{0}에서 {1} 사이의 값을 입력해 주십시오.
email	이메일 형식에 맞게 입력해 주십시오.
url	url 형식에 맞게 입력해 주십시오.
date	날짜를 YYYY/MM/DD 또는 YYYY-MM-DD 형식에 맞게 입력해 주십시오.
mindate	{0} 또는 {0} 이후의 날짜를 입력해 주십시오.
maxdate	{0} 또는 {0} 이전의 날짜를 입력해 주십시오.
daterange	{0}에서 {1} 사이의 날짜를 입력해 주십시오.
oneof	다음중 하나의 값을 입력해 주십시오 : {param}.
number	실수를 입력해 주십시오.
integer	정수를 입력해 주십시오.
digits	숫자만 입력 가능합니다.
alphabet	알파벳만 입력 가능합니다.
equalTo	{0} 값만 가능합니다.
numalpha	숫자 또는 영문자만 입력 가능합니다.
nospace	스페이스는 입력할 수 없습니다.
hangul	한글만 입력 가능합니다.
numhan	숫자 또는 한글만 입력 가능합니다.
phone	대시(-)가 들어간 전화번호 형태를 입력해 주십시오.
mobile	대시(-)가 들어간 휴대전화번호 형태를 입력해 주십시오.
decimal	소숫점 {1}자리를 포함하여 최대 {0}자리까지 허용됩니다.

6.12.4.2. 빌트인 오류메시지 변경

빌트인 오류 메시지가 아닌 별도의 에러메시지를 사용하고자 할 때 data-validation-message 사용합니다.

```
// 마크업 적용 시
<input type="text" id="input1" data-validation-message="{required:'반드시 입력하세요!'}">

// javascript 적용 시
$('#input1').validator({
    rule : {
        required: true
    },
    message: {
        required: '반드시 입력하세요!'
    }
});
```

6.12.4.3. 오류메시지 동작원리

method 에 의한 검증 결과에 따라 오류메시지를 생성하게 됩니다. 생성된 오류메시지는 옵션이 지정하는 바에 따라 label 또는 data-for 속성을 가진 element 의 텍스트로 지정이 됩니다. 메시지는 룰을 명시할 때에 받은 파라미터를 사용하여 치환이 될 수 있으며 {숫자} 형태의 텍스트가 있을 경우 파라미터의 첫번째 값부터 순차적으로 치환하여 포맷팅을 수행하게 됩니다.

```
//minlength를 충족하지 못할때의 기본 메시지는 '최소 {0}글자 이상 입력하십시오.'입니다.
//minlength검증을 통과하지 못하게 되면, minlength의 파라미터인 5를 {0} 자리와 치환하여 최종적으로
//'최소 5글자 이상 입력하십시오.' 의 메시지를 생성하여 지정된 영역에 넣어주게 됩니다.
<input type="text" data-validation-rule="{minlength:5}">
```

6.12.4.4. 사용자 오류메시지 정의

method 에 해당되는 기본 메시지를 별도로 작성하거나 사용자 정의 method 에 대한 메시지를 만들고자 할 경우엔 Validator.setMessage 기능을 활용합니다.

```
Validator.setMessage(String method, {String|Function} message)
```

- **parameter**
  - ✓ method {String} Required

    method 이름입니다.
  - ✓ message {String|Function} Required

    method 검증을 통과하지 못하였을 때 출력할 오류메시지입니다. String 형태로 넣었을 경우 {n} 형태의 텍스트를 이용한 파라미터 포맷팅을 사용할 수 있습니다. 일반적으로 {0}를 입력하면 validate 룰로 지정한 파라미터(true 와 같은 값)로 치환이 됩니다. {attr:속성명} 의 형식을 이용한 스트링을 사용할 경우 검증 대상 element 의 HTML 속성값을 가져와서 치환하게 됩니다. Function 형태로 넣었을 경우, 해당 함수가 실행되면서 검증메시지를 조합하려 리턴하게 되면 리턴된 메시지를 오류메시지로 사용하게 됩니다. 이 때 함수의 첫번째 파라미터는 검증대상 엘리먼트입니다.

```
Validator.addMethod('customEqual', function(elem, value, param) {
   if(value === param 에서 'testing' 부분에 해당)를 치환
Validator.setMessage('customEqual', '{0}만 허용합니다.');

//또는. 검증 대상 element의 attribute 사용
Validator.setMessage('customEqual', '{attr:data-allow-attr}만 허용합니다.');

//또는. 특정 함수를 지정하여 오류메시지 생성
Validator.setMessage('customEqual', function(elem) {
   return $(elem).attr('data-allow-attr') + '만 허용합니다.';
});

<!-- 아래의 input 검증 실패시 'testing만 허용합니다' 라는 오류메시지가 출력됩니다. -->
<input type="text" data-validation-rule=" { customEqual : 'testing' }" data-allow-attr="testing">
```

6.13. 데이터바인드

Alopex 데이터바인드는 data-bind 속성을 사용해서 $a.request 통신으로 받은 데이터를 사용자 화면에 바인딩해주는 기능입니다. 사용자 뷰와 데이터가 연결되어 있기 때문에 화면의 데이터를 가져올 때나 다른 데이터를 넣는 것도 쉽고 빠르게 수행할 수 있습니다. 화면의 선택 영역에 데이터를 입력(set)하는 함수와 화면의 HTML element 영역 내 데이터를 파싱하여 리턴(get)하는 함수를 제공합니다.

6.13.1. data-bind

데이터를 어떤 방식으로 화면에 추가할 지 정의합니다. data-bind 속성값은 아래와 같은 형태를 가집니다. 콜론(:) 캐릭터를 기준으로 왼쪽에 어떤 컨트롤을 사용할 지 지정하고, 오른쪽에 HTML 엘리먼트에 추가될 데이터의 키를 지정합니다.

```
data-bind="control : data_key"
```

아래는 control 의 종류와 사용방법입니다.

- **html**

  데이터가 HTML element 의 innerHTML 에 추가됩니다.

```
<div id="sample">
   <div data-bind="html : data"></div>
</div>

<script>
        $a.page(function() {
                this.init = function() {
                        $('#sample').setData( {
                                data : '<a href="http://ui.alopex.io">Alopex UI</a>'
                        } );
                };
        });
</script>
```

결과 Alopex UI

- **text**

  데이터를 텍스트로 나타냅니다. HTML element 의 innerText 속성에 추가되며, HTML 특수문자가 escape 처리됩니다.

```
<div id="sample">
   <div data-bind="text : data"></div>
</div>

<script>
        $a.page(function() {
                this.init = function() {
                        $('#sample').setData( {
                                data : '<a href="http://ui.alopex.io">Alopex UI</a>'
                        } );
                }
        });
</script>
```

결과 Alopex UI

- **value**

  <input type="text"> 또는 <textarea> element 에 사용되며, 텍스트 입력 값에 데이터를 바인드될 때 사용합니다.

```
<div id="sample">
   <input class="Textinput" data-bind="value : data"/>
</div>

<script>
        $a.page(function() {
                this.init = function() {
                        $('#sample').setData( {
                                data : 'data'
                        } );
                }
        });
</script>
```

결과 data

- **checked(체크박스)**

  체크박스의 경우, 사용되는 패턴이 두 가지가 존재함으로 주의해서 사용합니다

  ✓ N 개 옵션

  INPUT 엘리먼트에 name 속성을 추가하여 여러 체크박스 인풋끼리 그룹핑을 합니다. 데이터는 배열 타입으로 지정되고, 선택된 체크박스의 value 속성값을 가지고 있습니다.

```
<div id="sample">
<input id="checkbox1" type="checkbox" class="Checkbox" name="chk" value="check1" data-bind="checked: checkbox"/>
<label for="checkbox1">Check1</label>
<input id="checkbox2" type="checkbox" class="Checkbox" name="chk" value="check2" data-bind="checked: checkbox"/>
```

```
<label for="checkbox2">Check2</label>
<input id="checkbox3" type="checkbox" class="Checkbox" name="chk" value="check3" data-bind="checked: checkbox"/>
<label for="checkbox3">Check3</label></div>
</div>

<script>
	$a.page(function() {
		this.init = function() {
			$('#sample').setData( {
				checkbox : ['check1', 'check2']
			} );
		}
	});
</script>
```

결과 ☑ Check1 ☑ Check2 ☐ Check3

✓ 1 개 옵션

name 속성을 지정하지 않습니다. 기본 데이터 값은 true/false 로 Boolean 타입으로 지정됩니다.

이 외에 아래의 값들을 기본값으로 지원합니다. 이 경우에는 value 속성을 지정하여, 어떤 형태의 값을 사용할 지 지정합니다.

'y'/'n', 'yes'/'no', 'Y'/'N', 'YES'/'NO', '0'/'1', 'true'/'false', 'TRUE'/'FALSE'

```
<div id="sample">
	<label>
	<input type="checkbox" class="Checkbox" data-bind="checked: subscription" value='y'/>
	 I'd like to subscribe to receive email notification.
	</label>
</div>

<script>
	$a.page(function() {
		this.init = function() {
			$('#sample').setData( {
				subscription : true
			} );
		}
	});
</script>
```

결과 ☑ I'd like to subscribe to receive email notification.

- **checked(라디오버튼)**

  라디오의 경우, 여러 옵션 중 하나의 옵션만 선택하고 선택된 값의 value 값이 데이터로 사용됩니다. 여러 input 을 그룹핑하기 위해 name 속성을 사용합니다.

```
<div id="sample">
<input id="radio1" type="radio" class="Radio" name="radioGroup" value="value1" data-bind="checked: radio"/>
<label for="radio1">Radio1</label>
<input id="radio2" type="radio" class="Radio" name="radioGroup" value="value2" data-bind="checked: radio"/>
<label for="radio2">Radio2</label>
<input id="radio3" type="radio" class="Radio" name="radioGroup" value="value3" data-bind="checked: radio"/>
```

```
<label for="radio3">Radio3</label>
<input id="radio4" type="radio" class="Radio" name="radioGroup" value="value4" data-bind="checked: radio"/>
<label for="radio4">Radio4</label>
</div>

<script>
	$a.page(function() {
		this.init = function() {
			$('#sample').setData( {
				radio : 'value2'
			} );
		}
	});
</script>
```

결과 Radio1 Radio2 Radio3 Radio4

- **options, selectedOptions**

  select element 내 option 과 선택된 값을 바인드합니다.

```
<div id="sample">
<select class="Select" data-bind="options: selectOptions1, selectedOptions: selected1"></select>
</div>
<script>
	$a.page(function() {
		this.init = function() {
			$('#sample').setData({
				selectOptions1: [ { value: 'opt1', text: '첫 번째 옵션' }, { value: 'opt2', text: '두 번째 옵션' } ],
				selected1: 'opt2'
			});
		}
	});
</script>
```

결과 두번째 옵션

- **attr**

  데이터를 HTML 엘리먼트의 속성에 바인드합니다.

```
<div id="sample">
<input id="validation" class="Textinput" data-bind="attr: {data-validate-rule: vrule}"/>
</div>

<script>
	$a.page(function() {
		this.init = function() {
			$('#sample8').setData( {
				vrule: '{ required: true, minlength: 8, maxlength: 14 }'
			} );
		}
	});
</script>
```

결과 <input id="validation" class="Textinput" data-bind="attr: {data-validate-rule: vrule}" data-validate-rule="{required: true, minlength: 8, maxlength: 14}" data-type="textinput" data-classinit="true" data-converted="true">

- **css**

  데이터를 CSS 스타일에 적용합니다.

```
<div id="sample">
<div data-bind="css: {background: background}">여기에 색상이 바뀝니다.</div>
</div>

<script>
	$a.page(function() {
		this.init = function() {
			$('#sample').setData( {
				background : '#F00000'
			} );
		}
	});
</script>
```

결과 여기에 색상이 바뀝니다.

- **foreach**

  배열 타입의 데이터를 화면에 반복적으로 나타냅니다.

```
<div id="sample">
  <p>list 바인딩</p>
  <ul class="List" data-bind="foreach: list" style="padding-left: 0px;">
    <li>
      <img data-bind="attr: {src: image}"/>
      <strong data-bind="text: title"></strong>
      <label data-bind="text: description"></label>
    </li>
  </ul>
  <p>dropdown 바인딩 <b style="color:blue">클릭하세요.</b></p>
  <input class="Textinput" />
  <ul id="dropdown" class="Dropdown" data-bind="foreach: list" style="padding-left: 0px;">
    <li>
      <img style="width:30px;height:30px;vertical-align:middle;" data-bind="attr: {src: image}"/>
      <strong data-bind="text: title"></strong>
      <label data-bind="text: description"></label>
    </li>
  </ul>
</div>

<script>
	$a.page(function() {
		this.init = function() {
			$('#sample').setData({
				list: [{
				image: 'icon/apple.png',
					title: 'Apple',
					description: 'Tool for poisoning a princess'
				}, {
				image: 'icon/crystalshoes.png',
				title: 'Crystal Shoes',
				description: 'Princess Maker'
				}]
			});
		}
	});
</script>
```

- **template**

  트리 컴퍼넌트에서는 각각의 노드는 자식이 존재하는 경우에 반복적으로 렌더링됩니다. 이렇게 특정 조건에서 반복되는 형태는 template 컨트롤을 지정하여 데이터 바인드 합니다.

  "template" 컨트롤의 "name"으로 어떤 템플릿을 사용할지 정의합니다. "foreach"는 데이터 구조에서 어떤 데이터를 참조하여 템플릿을 그릴지 지정합니다. 다음 예제에서는 트리 노드 오브젝트에서 "items"키에 데이터가 있는지에 따라 트리노드를 렌더링 합니다.

```
<ul id="tree" class="Tree" data-bind="template: { name: treeElement, foreach: treedata}"></ul>

<script type="text/html" id="treeElement">
<li>
   <a>
      <img data-bind="attr: {src: iconUrl}" >
      <label data-bind="html: text"></label>
   </a>
   <ul data-bind="template: {name: treeElement, foreach: items}">
   </ul>
</li>
</script>
<script>
        $a.page(function() {
                this.init = function() {
                                var data = {
                                        treedata: [{
                                                id: '1',
                                                text: 'folder',
                                                iconUrl: 'databind/icon/windows/folder.png',
                                                items: [{
                                                id: '1-1',
                                                text: 'subfolder',
                                                iconUrl: 'databind/icon/windows/folder.png',
                                                items: [{
                                                        id: '1-1-1',
                                                        text: 'bmp',
                                                        iconUrl: 'databind/icon/windows/bmp.png',
                                                        items: []
                                                }, {
                                                        id: '1-1-2',
                                                        text: 'txt',
                                                        iconUrl: 'databind/icon/windows/txt.png',
                                                        items: []
                                                }, {
                                                        id: '1-1-3',
                                                        text: 'Word File',
                                                        iconUrl: 'databind/icon/windows/word59.png',
                                                        items: []
                                                }, {
                                                        id: '1-1-4',
                                                        text: 'Archieve',
                                                        iconUrl: 'databind/icon/windows/zip.png',
                                                        items: []
                                                }]
                                                }]
                                        }, {
                                                id: '2',
                                                text: 'Unknown',
                                                iconUrl: 'databind/icon/windows/file.png',
                                                items: []
                                        }]
                                };
                                $('#tree').setData(data);
                                $('#tree').tree(); // 데이터 바인딩으로 생성된 엘리먼트를 Tree 컴포넌트화 하는 과정
```

```
</script>
```

- **with**

  with 컨트롤은 파라미터 데이터로 전달된 데이터가 또 다른 오브젝트를 가지고 있는 경우, 하위 오브젝트가 바인드되는 영역을 지정합니다.

6.13.2. setData

jQuery 셀렉터로 선택한 영역 내 data-bind 속성이 있는 HTML 엘리먼트에 데이터를 추가하는 함수입니다.

- **API**

```
.setData(data)

parameter
        data{object} : 화면에 바인드할 데이터
```

- **예제코드**

  예제코드는 data-bind 의 코드 샘플들을 참고하세요.

6.13.3. getData

jQuery 셀렉터로 선택한 영역 내 HTML 엘리먼트 중 data-bind 속성이 명시된 엘리먼트에서 데이터를 가져오는 함수입니다.

- **API**

```
. getData(option)

parameter
        option : 원하는 데이터만 선별하여 가져올 수 있습니다.
            selectOptions {boolean} : 셀렉트 컴포넌트의 options 값을 가져올 수 있습니다. (default: false)

return
        data{object} : 선택된 화면 영역 내에서 읽어온 데이터
```

- **예제코드**

```
<div id="sample">
        <input id="checkbox1" type="checkbox" class="Checkbox" name="chk" value="check1" data-bind="checked: checkbox"/>
        <label for="checkbox1">Check1</label>
        <input id="checkbox2" type="checkbox" class="Checkbox" name="chk" value="check2" data-bind="checked: checkbox"/>
        <label for="checkbox2">Check2</label>
</div>
<button id="get" class="Button">GET Data</button>

<script>
        $a.page(function() {
```

```
            this.init = function() {
                $('#get').on('click', function() {
                    alert(JSON.stringify($('#sample').getData()));
                });
            }
        });
</script>
```

6.14. 통신(ajax) 처리

6.14.1. 요청/응답 처리

6.14.1.1. 요청

- **통신 요청**

  통신 요청 방법은 아래와 같습니다.

```
$a.request('serviceId', { // 서비스 ID

   data: {}, // 통신 parameter

   success: function(res) {
      // 통신이 성공적으로 이루어 진 경우 호출되는 콜백함수
   },

   fail: function(res) {
      // 통신은 성공적으로 이루어 졌으나, 서버오류가 발생한 경우 호출되는 콜백함수
   },

   error: function(errObject) {
      // 통신이 실패한 경우 호출되는 콜백함수
   }
});
```

- **요청 데이터 설정(자동)**

  $a.request 함수의 data 옵션을 object 로 지정하지 않고 마크업 셀렉터(또는 CSS 셀렉터라고도 불리며 HTML 을 선택하는 문법)로 지정하면, 해당 영역에서 데이터를 파싱하여 통신 파라미터에 추가합니다.

```
data: '#grid',
data: ['#form', '#grid'], // 한 개이상의 영역의 경우 배열의 형태로 셀렉터 정의.
```

  요청 데이터 자동 설정은 Alopex 데이터바인드를 사용하기 때문에, HTML 엘리먼트에 data-bind 속성이 정의되어야 합니다. Alopex 데이터바인드 사용방법은 데이터바인드의 getData 를 참고하세요.

- **요청 데이터 설정(수동)**

  데이터를 수동으로 설정하는 경우에는 object 형태로 작성해야 합니다. 프로젝트 서버 환경에 따라 세부적인 데이터 규격은 달라질 수 있습니다.

```
$a.request('serviceId', { // 서비스 ID
```

```
  data: {
            key : value
      },

      …
});
```

서버 환경이 NEXCORE J2EE 프레임워크인 경우 데이터 규격은 아래와 같습니다.

```
$a.request('serviceId', { // 서비스 ID

  data: {
            dataSet: {
      message: {…},
            fields: {…},
                  recordSets: {…}
            }
      },

      …
});
```

6.14.1.2. 응답

- **통신 응답**

  통신 응답에 대한 처리는 아래와 같이 success, fail, error 함수에서 처리 할 수 있습니다.

```
$a.request('serviceId', { // 서비스 ID
      …
success: function(res) {…},
fail: function(res) {…},
error: function(res) {…},
  …
});
```

- **응답 데이터 바인드(자동)**

  $a.request 함수의 success 옵션을 다음과 같이 특정 영역의 셀렉터로 지정하여 선택된 영역에 데이터를 바인드합니다. 데이터바인드하고자 하는 영역의 셀렉터를 입력합니다.

```
success: '#grid', // 한 영역에 데이터 바인드하는 경우
success: ['#grid', '#form'], // 한 개 이상의 영역에 데이터 바인드하는 경우
success: ['#grid', function(res) {
  // 자동 바인딩 하고, 콜백함수 호출.
}]
```

- **응답 데이터 바인드(수동)**

  응답 데이터를 가공하여 화면에 보여줘야 하는 경우가 있습니다. 이런 경우 응답 데이터에서 원하는 데이터를 추출해 가공하고 데이터 바인드의 setData 를 통해 바인드합니다

  Alopex 데이터바인드 사용방법은 데이터바인드의 setData 를 참고하세요.

6.14.2. 동기/비동기 통신

동기/비동기 통신 방식을 설정할 수 있습니다. true 는 비동기 방식, false 는 동기방식입니다. Alopex 에서 제공하는 통신의 기본값은 비동기 방식입니다.

```
$a.request('serviceId', { // 서비스 ID
        …
    async: true/false
        …
});
```

6.14.3. 프로그레스 처리

일반적으로 통신 중에는 프로그레스를 띄워 사용자에게 통신 중이라는 사실을 알려줍니다. 프로그레스 처리는 모든 통신 요청에 공통적으로 적용되므로 통신 셋업에서 처리합니다. 자세한 처리 방법은 통신 셋업의 글로벌 콜백함수를 참고하세요.

6.15. 프로그레스

6.15.1. 프로그레스 개념

프로그레스는 화면이 이동할 때 혹은 사용자가 통신 호출을 할 때 프로그레스 바가 표시되면서 사용자에게 프로그램이 정상적으로 동작하고 있다는 것을 인식 시켜주며, 화면이 동 및 통신 호출 시간 동안 다른 사용자 액션을 막아 주는 기능을 합니다. Alopex 에서는 빌트인 프로그레스를 제공하고 사용자 정의 프로그레스를 만들 수 있도록 기능을 제공합니다.

6.15.2. 프로그레스 종류

사용자 액션이 진행되는 동안 사용자로부터의 다른 인터랙션을 방지하기 위해 오버레이 및 프로그레스 함수를 제공합니다.

6.15.2.1. 빌트인 프로그레스

- **Overlay**

  선택된 영역과 같은 크기의 오버레이를 영역 바로 위에 생성합니다. 함수 호출 시 셀렉터로 선택된 대상이 document 인 경우 전체화면에 오버레이를 띄우게 됩니다.

```
var overlay = $('#content').overlay();
overlay.remove();
```

- **Progress**

  오버레이를 생성하고, 프로그레스바를 보여주는 함수입니다.

```
var progress = $('#content').progress();
progress.remove();
```

6.15.2.2. 사용자 정의 프로그레스

프로젝트에 특화된 프로그레스를 만들 수 있습니다. createProgress, resizeProgress, removeProgress 함수를 재정의 해서 progress 함수를 호출할 때 parameter 로 전달합니다. 모든 progress 에 적용하기 위해서는 default 로 지정하면됩니다.

```
var customCreateProgress = function() { ... };
var customResizeProgress = function() { ... };
var customRemoveProgress = function() { ... };
```

```
// 다음과 같이 overlay/progress 생성 시 직접 세팅할 수 있습니다.
$('div').progress({
    createProgress : customCreateProgress,
    resizeProgress : customResizeProgress,
    removeProgress : customRemoveProgress
});

// 모든 overlay/progress에 동일하게 적용하기 위해 default로 지정하는 방법입니다.
AlopexOverlay.defaultOption.createProgress = customCreateProgress;
AlopexOverlay.defaultOption.resizeProgress = customResizeProgress;
AlopexOverlay.defaultOption.removeProgress = customRemoveProgress;
```

API 에 대한 자세한 내용은 홈페이지의 Progress 를 참고하세요.

7. 웹 퍼포먼스 향상 가이드

웹 어플리케이션은 PC 또는 모바일 디바이스의 환경에 맞는 최적화가 필요 합니다.

아래 리스트는 웹 성능을 높이기 위한 체크 리스트 입니다. 보다 자세한 내용은 웹 퍼포먼스 향상 가이드를 참고하세요.

7.1. 체크 리스트

(*)표시는 모바일 웹에 주로 해당

구분		설명
Request	Resource	CSS 에 이미지 sprite 적용하여 다수의 이미지 한번에 가져오기
		중복 javascript 다운로드 방지
		작은 css 파일은 html 에 삽입
		CSS 참조 태그는 <head> 태그 최상단에 선언
		Javascript / css 참조에 type (text/javascript 또는 text/css)은 생략(html5)
		Javascript 참조 태그는 </body> 태그 바로 앞에 선언
	Reduce	서비스 요청 최소화
	Async	비동기 통신 사용
	Cache	ETag 헤더 설정
		캐쉬 만료일 설정
		Html5 의 localstorage, sessionstorage 사용(모바일에서 브라우저 캐시보다 빠름) *
	redirect	모바일 사이트로의 url 을 직접 호출(리다이렉트 방지) *
Payload	Minimizing	주석이 제거 된 javascript 등 minimizing 된 리소스를 활용
		이미지 사이즈 최소화
		gZip 압축 설정
	Simply	Html5 요소(header,section..)와 css3(box-shadow,...)를 이용하여 bulit-in 된 element 및 스타일 기법 사용 *
Client	Rendering	용량이 큰 이미지의 경우, 렌더링 지연 적용
		HTML 태그에 STYLE 속성 사용 안함(중복 렌더링 방지)

		CSSS 를 이용한 애니메이션 처리 (-webkit-transform 등) *
		마크업 최적화
	Loading	스크립트의 실행은 페이지의 onload 이후 실행
	Ajax	구조만 갖추어진 어플리케이션을 빠르게 로딩할 수 있도록 한 후, 사용자가 페이지를 보고 있을 동안에 Ajax 를 이용하여 필요한 상세 컨텐츠 데이터 로딩
		Json 데이터 파싱은 eval()대신 JSON.parse() 이용
	Preloading	페이지 onload 이후 예상되는 이후 페이지에 대한 자원을 미리 로딩
	Multi-thread	HTML5 의 웹워커를 토한 multi-thread *
	event	클릭 이벤트는 터치 이벤트로 사용(300ms 속도 개선) *
Alopex UI	DataBinding,Validator, Convert	selector 및 하위 요소를 탐색하며 수행되는 Javascript API 사용 시 selector 영역을 구체적이고, 최소한의 범위로 지정해야 합니다.
	Popup	부모 window 에 include 된 라이브러리(js, css 등)와 Popup 자식 window 에 include 될 라이브러리가 상당 부분 겹치는 경우 {iframe:true} 방식의 Popup 을 사용하면 라이브러리의 불필요한 중복 로딩으로 성능 저하의 원인이 됩니다. 이럴 경우 {iframe:false} 방식의 사용을 고려할 수 있습니다.
	Pregress	duration(Progress overlay 생성 시 animation 효과), durationOff(Progress overlay 제거 시 animation 효과) 등 animation 관련 설정은 성능에 영향을 주기 때문에 상황에 따라 animation 효과를 적절히 설정 합니다.
	Setup	Alopex Component 및 Javascript API 사용 시 공통 js(common.js 등) 내에 setup 으로 코드 양을 줄입니다
	Storage	각 화면에 동일하게 사용되는 데이터는 각 화면에서 통신을 통해 가져오는 방식 보다는 동일 도메인 내 브라우저 종료 전까지 데이터를 저장할 수 있는 $a.session(key, value) API 를 통해 저장 및 재사용 합니다.