

DevOn RichUI

RichUI 개발자 가이드

Ver. 1.1

관리부서 : 모바일/UI 기술그룹



Copyright © LG CNS

LG CNS의 사전 승인 없이 본 내용의 전부 또는 일부에 대한 복사, 배포, 사용을 금합니다.

개 정 이 력

버전	작성일	변경내용 ¹	작성자	승인자
1.0	2013.06.25	최초작성	문혁찬	김순호
1.1	2013.12.02	LCombo 의 목록 구성관련 경고 추가	문혁찬	장문수

¹ 변경 내용: 변경이 발생하는 위치와 변경 내용을 자세히 기록(장/절과 변경 내용을 기술한다.)

목 차

1. RICHUI 소개	1
1.1 X-Internet, RIA.....	1
1.2 플러그인, 웹표준방식 RIA 솔루션 장단점.....	1
1.3 주요특징	2
1.4 솔루션 구성	2
1.5 DevOn RichUI Architecture View.....	3
2. 구조	4
2.1 구조설명	4
2.2 RUI 라이브러리	4
2.3 Component diagram	5
2.4 Class diagram	5
2.4.1 Base Class diagram	5
2.4.2 Core Class diagram.....	6
2.4.3 UI Class diagram	7
2.4.4 Form Class diagram.....	8
2.4.5 Layout/Menu/Tab/Tree Class diagram	8
2.4.6 Grid/Fx Class diagram.....	9
2.4.7 Fx Class diagram	9
2.4.8 Plugins utility Class diagram.....	9
3. 개발환경 설정.....	11
3.1 개요.....	11
3.2 사전 설치 권장 프로그램	11
3.2.1 Chrome 브라우저	11
3.2.2 Chrome 개발자 도구	12
3.3 RichUI 배포본 구성	14
3.3.1 RichUI 설치	14
3.3.2 Folder 구조.....	14
3.3.3 api 폴더	15
3.3.4 js 폴더	15
3.3.5 docs 폴더	16

3.3.6 plugins 폴더.....	16
3.3.7 sample 폴더.....	16
3.3.8 resources 폴더.....	16
4. RICHUI 설정	18
4.1 구조.....	18
4.2 rui_config.js 설정.....	19
4.2.1 RichUI 영역별 요소들	19
4.2.1.1 \$.core	19
4.2.1.2 \$.base.....	20
4.2.1.3 \$.ext	22
4.2.1.4 \$.project.....	23
5. RICHUI 배포	24
5.1 min 및 debug.....	24
6. UI 컴포넌트.....	25
6.1 FORM 컴포넌트.....	25
6.1.1 FORM 컴포넌트 종류	25
6.1.2 FORM 의 기능	26
6.1.3 FORM 컴포넌트 사용	26
6.1.3.1 TextBox.....	26
6.1.3.2 Combo.....	28
6.1.3.3 CheckBox & CheckBoxGroup.....	30
6.1.3.4 Radio & RadioGroup	32
6.1.3.5 Rui.ui.form.LForm.....	32
6.2 Button 사용	35
6.2.1 Button	35
6.3 데이터 처리	36
6.3.1 DataSet.....	36
6.3.1.1 DataSet 의 필수 Config.....	36
6.3.1.2 DataSet 의 생성.....	37
6.3.1.3 DataSet 의 이벤트	37
6.3.2 Field	38
6.3.3 Record	38
6.3.3.1 Record 의 생성	38

6.3.4 JSON DataSet 과 Delimiter DataSet.....	39
6.3.5 DataSetManager.....	39
6.3.5.1 DataSet 의 변경 데이터 서버 전송.....	40
6.3.5.2 여러 개의 DataSet 조회.....	40
6.3.6 Bind 사용	40
6.3.6.1 Bind 의 Config.....	41
6.4 Validation.....	41
6.4.1 ValidatorManager 선언 및 사용.....	41
6.4.2 Validator 구조	43
6.4.3 Validator 의 종류	43
6.4.4 프로젝트용 공통 Custom Validator 만들기	45
6.5 Grid 컴포넌트	46
6.5.1 Buffered Grid	46
6.5.2 Grid 의 생성	46
6.5.3 ColumnModel.....	47
6.5.4 ColumnModel 의 Config	47
6.5.5 Column 의 Config	47
6.5.6 Column 속성의 renderer	48
6.5.7 rederer 인수 설명	48
6.5.8 Grid 의 편집기.....	48
6.5.9 Grid 의 편집 가능/불가능 설정.....	49
6.5.10 소계.....	50
6.5.11 합계.....	50
6.5.12 합계의 구현.....	50
6.5.13 Tree Grid	51
6.5.14 Tree Grid 의 구현	51
6.5.15 TreeGrid 의 필수 config.....	52
6.5.16 Rui.ui.LDialog.....	53
6.5.17 Dialog(대화상자)의 구현	53
7. 디버깅	56
7.1 Chrome 개발자 도구.....	56
7.1.1 Elements 패널	57
7.1.1.1 DOM 선택	58

7.1.1.2 DOM 속성 변경	58
7.1.1.3 DOM 분석	59
7.1.1.4 Elements 패널 참고	60
7.1.2 Resources 패널	60
7.1.2.1 Resources 패널 참고	61
7.1.3 Network 패널	61
7.1.3.1 Network 자원의 세부정보	63
7.1.3.2 Network 패널 참고	64
7.1.4 Sources 패널	64
7.1.4.1 소스코드 검사	65
7.1.4.2 Breakpoint 와 디버깅	65
7.1.4.3 디버깅을 위해 제공되는 정보	66
7.1.4.4 Sources 패널 참고	67
7.1.5 Console 패널	67
7.1.5.1 Console 의 다양한 이용	67
7.1.5.2 Console 패널 참고	68
7.1.6 기타 패널	68
8. RICHUI 다국어 처리	69
8.1 개요	69
8.1.1 다국어 지원 처리 흐름	69
8.1.2 설정 파일	69
8.1.3 주의사항	70
8.2 설정	70
8.2.1 디렉토리 및 파일 규칙	70
8.2.2 Locale 파일 작성 방법	70
8.2.3 신규 locale 파일 작성 방법	71
8.2.4 메시지 작성 규칙	71
8.2.5 Core 단계의 변수 설명	71
8.2.6 LDateLocale 작성 방법	72
8.2.7 LDateLocale 변수 설명	72
8.3 구현	72
8.3.1 다국어 설정값 변경 방법	72
8.3.2 메시지 사용 방법	73
8.3.3 그리드에서의 다국어 처리	73

8.3.4	그리드에서의 사용자 renderer 처리	74
8.3.5	유틸리티를 이용한 값 변환 처리	74
9.	버전관리	76
9.1	상위버전의 추가된 기능 사용.....	76
9.1.1	RichUI 1.x 환경에서 RichUI 2.0 사용	76
9.1.1.1	RichUI 2.0 을 workspace 에 추가.....	76
9.1.1.2	추가한 RichUI 2.0 의 라이브러리로 개발	77
9.1.1.3	주의사항.....	77
10.	대용량 데이터 처리.....	78
10.1	대용량 데이터 로드.....	78
10.1.1	LJsonDataSet 과 LDelimiterDataSet 의 차이점	78
10.1.2	LDelimiterDataSet 방식 구조.....	79
10.1.3	LDelimiterDataSet Load 방법.....	79
10.2	대용량 데이터 변경.....	80
10.2.1	Batch 수행 시 무시되는 이벤트	80
10.2.2	Batch 메소드	81
10.2.2.1	설명	81
10.2.2.2	메소드 실행 파라미터.....	81
10.2.3	Batch 처리 방법	81
10.2.3.1	신규 건의 데이터를 반복적으로 추가하는 예.....	81
10.2.3.2	첫번째 DataSet 을 두번째 데이터셋에 복사하는 예.....	81

1. RichUI 소개

이 문서는 DevOn RUI 의 소개 및 구조를 설명하는 문서로써 주요특징 및 기능, Architecture View 와 UML 로 Diagram 을 표현한 문서입니다.

1.1 X-Internet, RIA

X-Internet, RIA는 웹브라우저에서 데스크탑 어플리케이션의 기능, 성능과 풍부한 미디어를 지원하기 위해 발생한 개념으로 현재는 같은 의미로 사용되는 경우가 많음

X-Internet : Extended, Executable Internet

- by Forrester Research (2000)

RIA : Rich Internet Application

- by Macromedia for Flash MX (2002)

RIA 솔루션의 유형

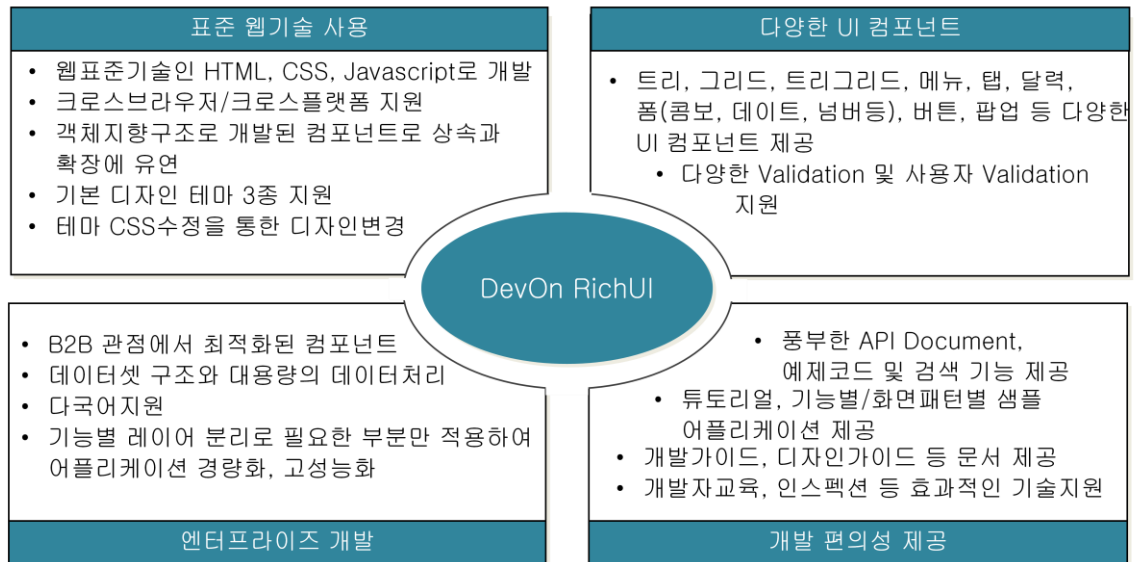
- 플러그인방식 : 전용 런타임 혹은 브라우저 플러그인 형식으로 구현
- 웹표준방식 : Javascript Ajax 등 웹표준기술을 이용하여 구현

1.2 플러그인, 웹표준방식 RIA 솔루션 장단점

	플러그인 방식	웹표준 방식
생산성	높음 (독립적인 플랫폼)	낮음 (브라우저별로 구현)
성능	높음 (컴파일 언어)	낮음 (인터프리터 언어, OS 및 브라우저에 따라 성능 차이가 심함)
확장성	낮음 (추가적인 기능을 만들 수 없음)	높음 (언어 특성상 확장이 가능)
호환성	낮음 (신규 OS 및 브라우저별 지원 안됨)	높음 (웹표준을 따르는 OS 및 브라우저는 사용 가능)
기능성	높음 (OS 에 맞춰서 다양한 기능을 구현)	낮음 (브라우저에 종속적이므로 OS 레벨의 제어가 불가능)

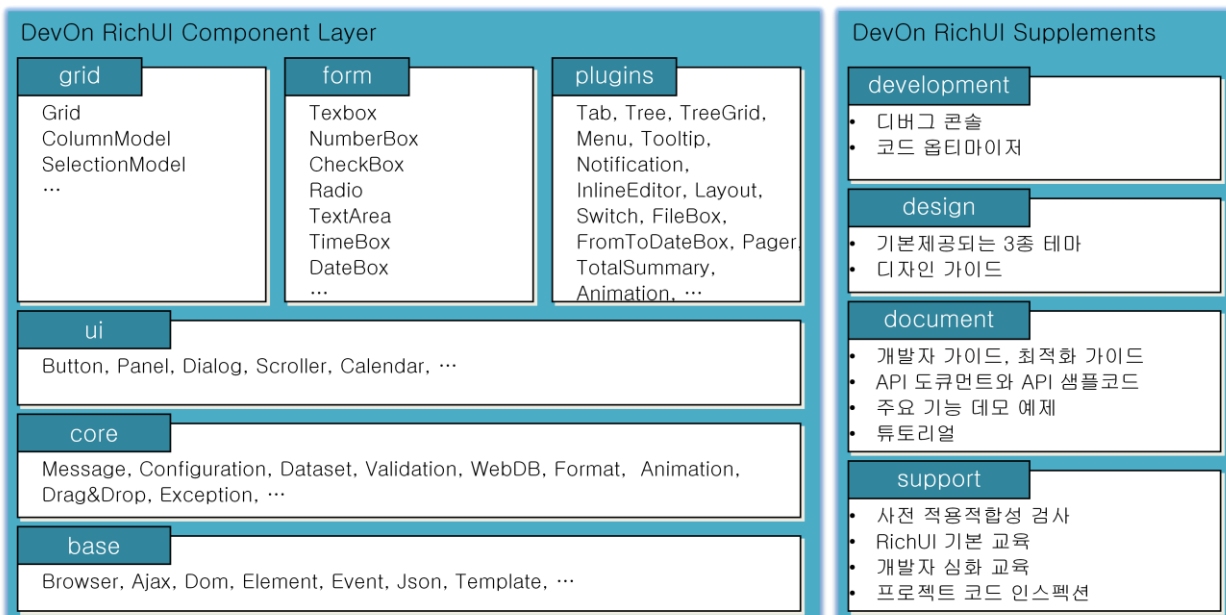
1.3 주요특징

DevOn RichUI는 크로스브라우저/크로스플랫폼을 지원해야하는 기업용 웹 프로젝트 수행을 위한, 다양하고 풍부한 기능과 최적화된 성능을 제공하는 웹표준기술을 활용한 UI 컴포넌트 라이브러리 입니다.

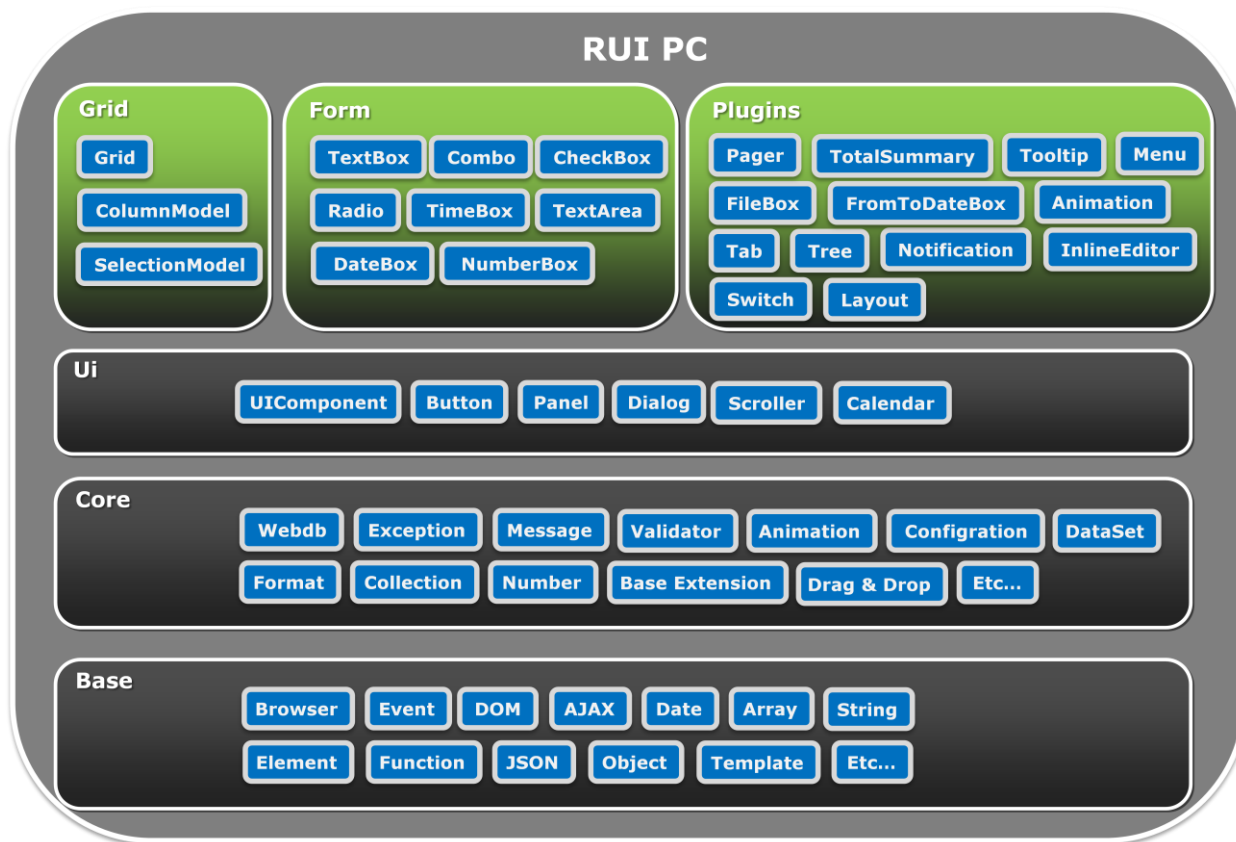


1.4 솔루션 구성

기업형 웹 어플리케이션을 개발하는데 필요한 다양한 UI 컴포넌트 및 데이터 관리 기능을 제공



1.5 DevOn RichUI Architecture View



DevOn RUI 는 Base, Core, UI, Grid, Form 외 plugins 로 구성되어 있습니다.

Base	Ajax, Dom 등 HTML Element 및 ajax 를 제어하는 기본 Layer
Core	RichUI 의 UIComponent 를 제외한 화면 제어시 데이터 처리 및 기본 정보(Configuration, message 등)를 설정하는 Layer
UI	화면에 출력되는 UI 콤포넌트들의 기본 Layer (UIComponent, Panel, Button 등)
Grid	그리드를 출력하기 위한 UI 콤포넌트
Form	TextBox, Combo 등 Form Object 들을 출력하기 위한 UI 콤포넌트
Plugins	Base, Core, UI, Grid, Form 외에 기능들을 탑재하여 더 다양한 UI 를 구현할 수 있는 플러그인들

2. 구조

2.1 구조설명

DevOn RUI 는 다양한 기능들과 복잡한 이벤트를 처리하기 위해서 javascript 로 Java 와 같이 OOP(Object Oriented Programming) 를 적용한 상속 구조로 구현되었다. 컴포넌트로서 동작하는 대부분의 객체들은 상속구조로 구현되었으며, 단순하게 사용 가능한 객체들은 static 으로 접근 가능한 객체로 이루어져 있다.

DevOn RUI 는 총 4 가지 객체로 구성되어 있다.

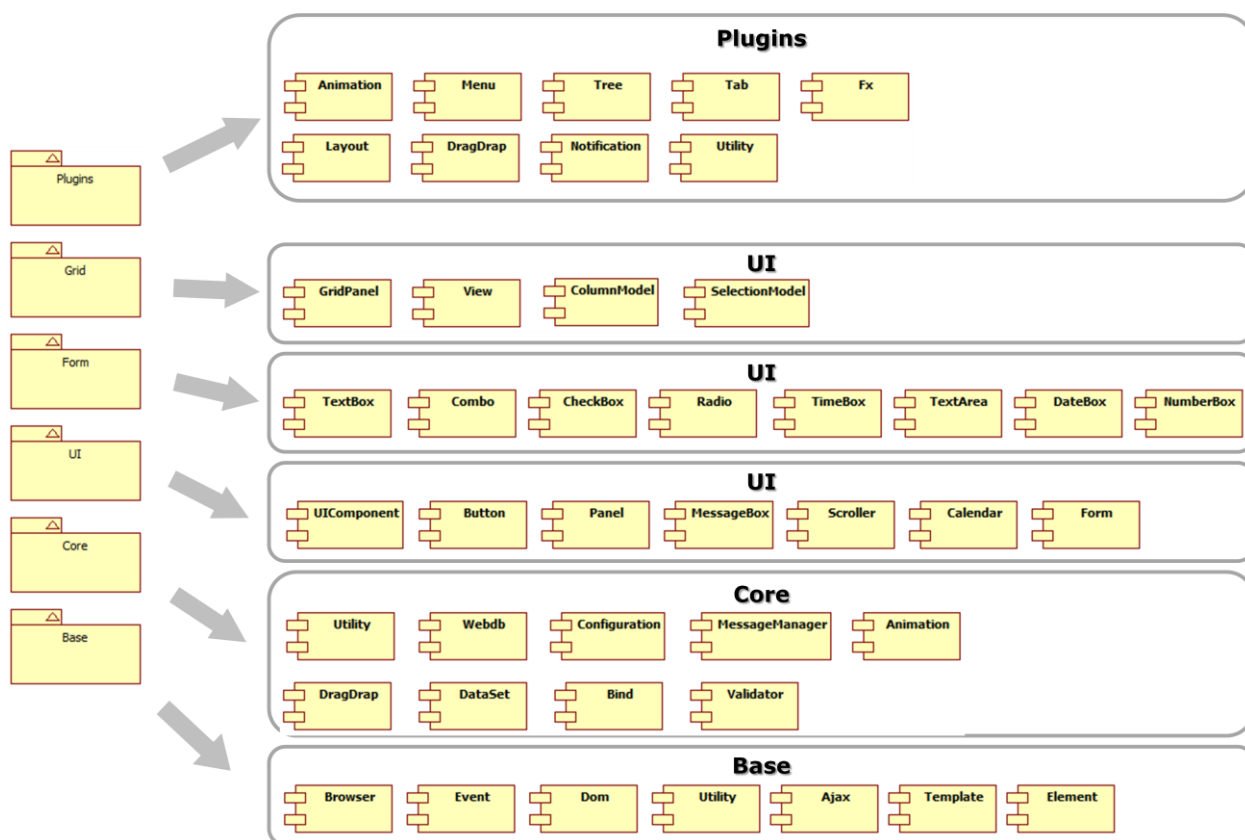
Static Class	객체를 생성하지 않는 static method 만으로 이루어져 있는 Class (Utility 등...)
Event Class	객체를 생성하고 수행 시 event 를 탑재하여 처리하는 Class (DataSet, Bind, Config, Message 등 UI 가 없는 데이터 처리 컴포넌트)
Non event Class	객체를 생성하나 event 등을 탑재하지 못하는 Class (Validation, WebDB, Element 등)
UI Class	객체를 생성하고 event 를 탑재할 수 있으며, 화면에 해당 기능을 수행하는 영역이 존재하는 Class (Grid, Tree, Tab, Menu 등...)

Static/None event Class 는 성능 및 사용성이 단순하고 Event/UI Class 는 성능 및 사용성이 복잡하다.

2.2 RUI 라이브러리

Base	RUI 를 사용하기 위한 필수 라이브러리로서 유틸리티, DOM 제어와 같은 Class 등을 포함하고 있다.
Core	화면에는 출력되지 않지만 UI 컴포넌트와 유기적으로 연동하는 데이터 처리, Animation, 유효성 체크와 같은 Class 등을 포함하고 있다.
UI	화면에 보이는 객체들로 사용자와 상호작용하는 일련의 뷰를 제공한다.
Form	TextBox, Combo, DateBox 와 같은 Form 입력 Class 등을 포함하고 있다.
Grid	Grid 관련 핵심 기능을 포함하고 있다.
Plugins	기본적으로 Core 와 UI 와 함께 제공되지는 않지만 특정 구현 화면 마다 구현하여 사용자에게 더 다양한 기능을 제공할 수 있다.

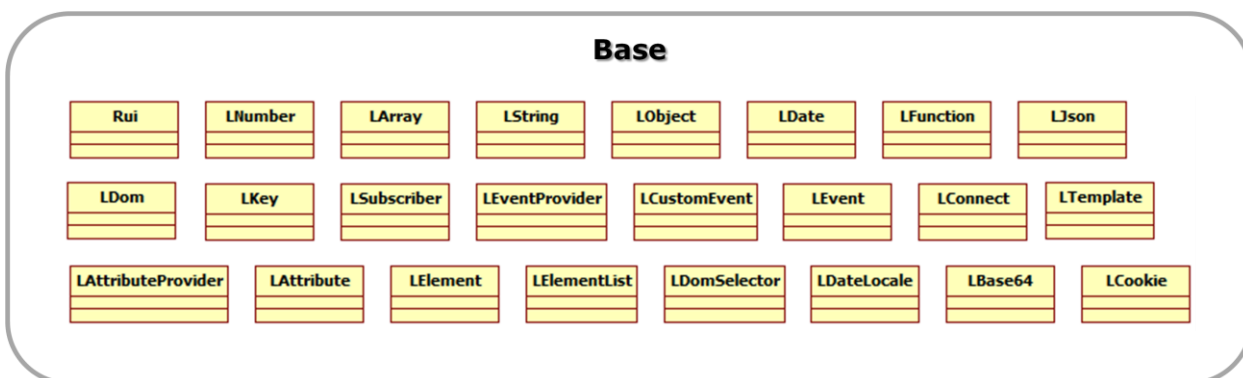
2.3 Component diagram



2.4 Class diagram

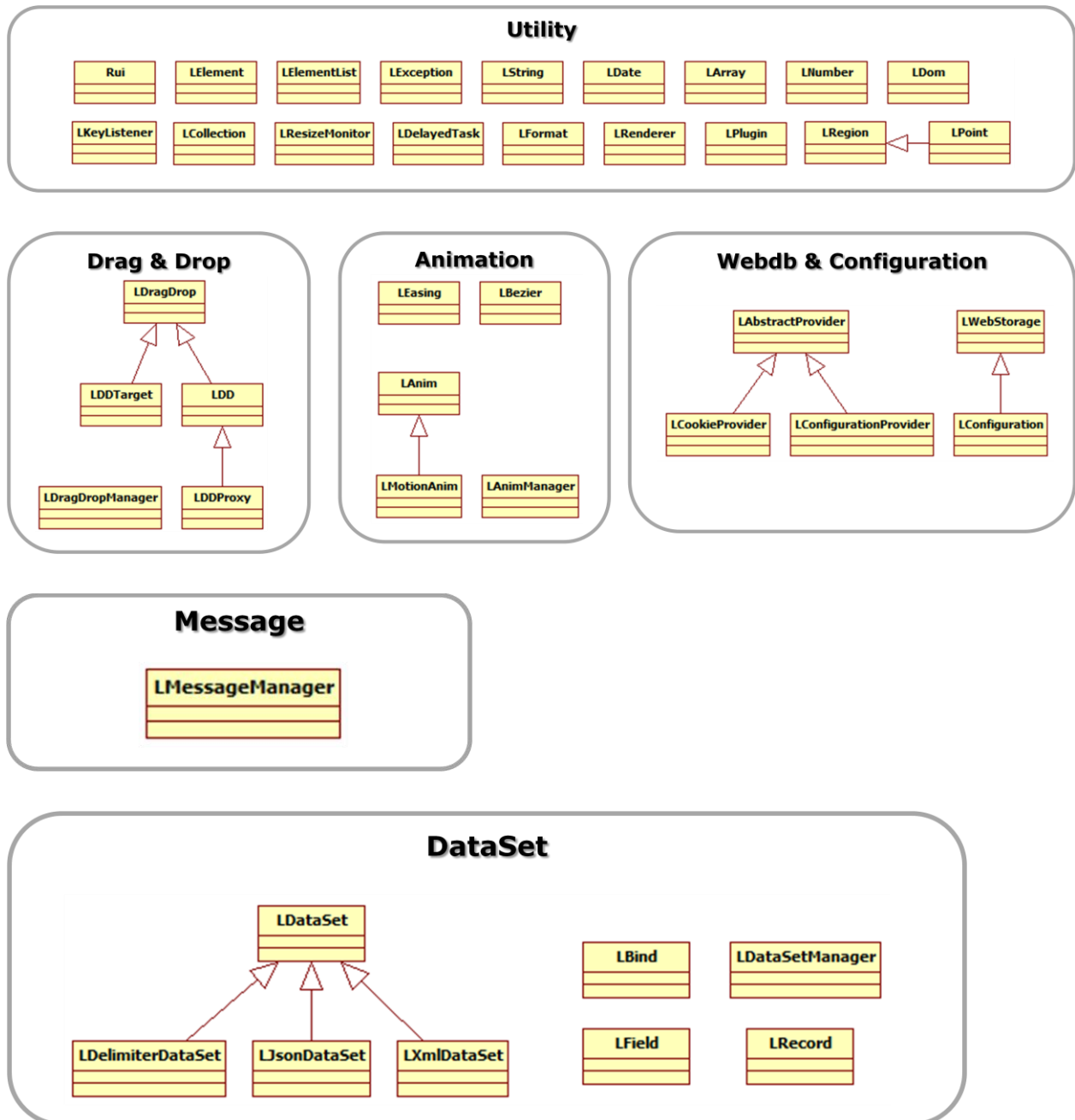
2.4.1 Base Class diagram

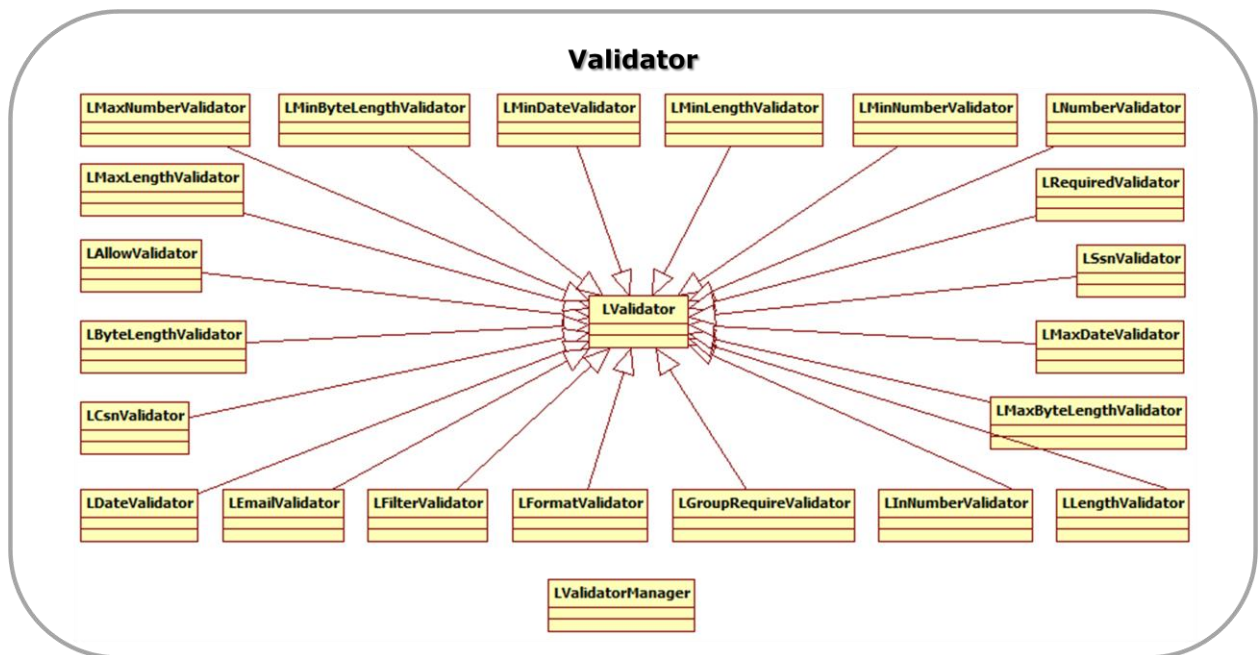
Base Layer 는 독립적으로 수행되는 클래스 객체들로 HTML 을 제어할 경우 최소한의 웹표준 기술을 적용할 수 있는 유틸리티로 구성되어 있습니다.



2.4.2 Core Class diagram

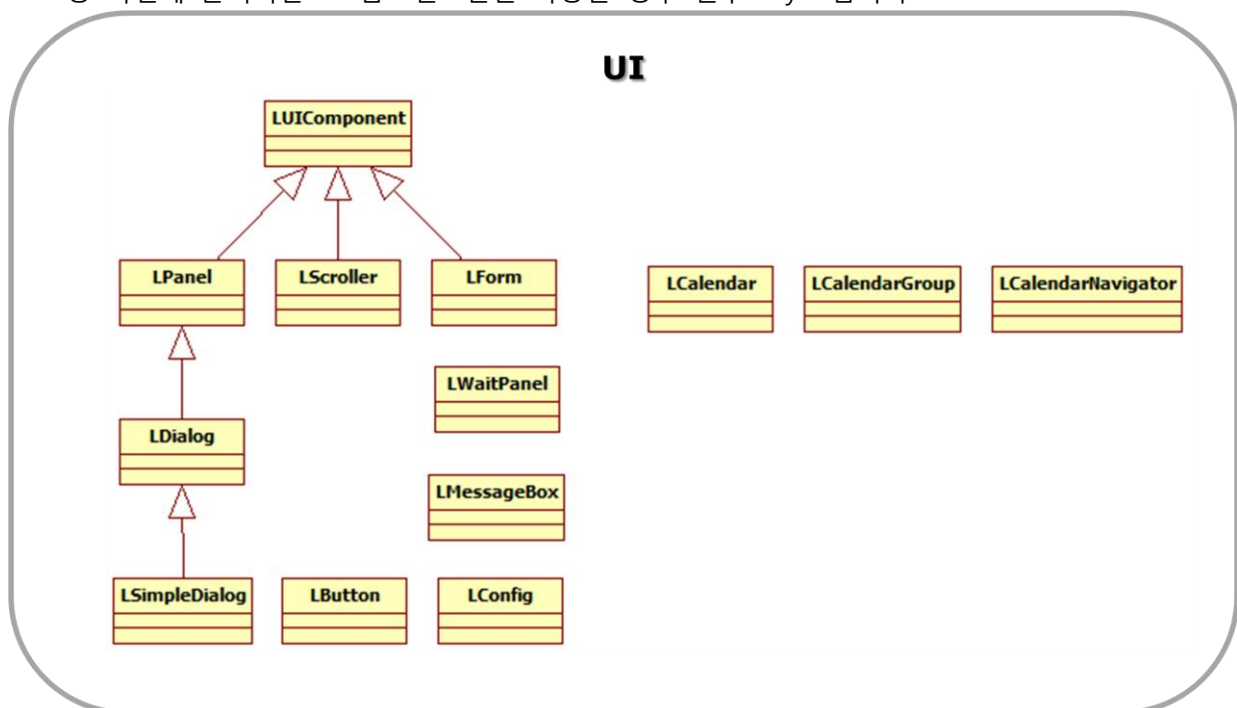
Core Layer 는 데이터 처리, 유효성 체크, Drag & Drop 등 UI 컴포넌트들을 제외한 Base Layer 에서 처리하지 않는 컴포넌트들을 제공합니다.





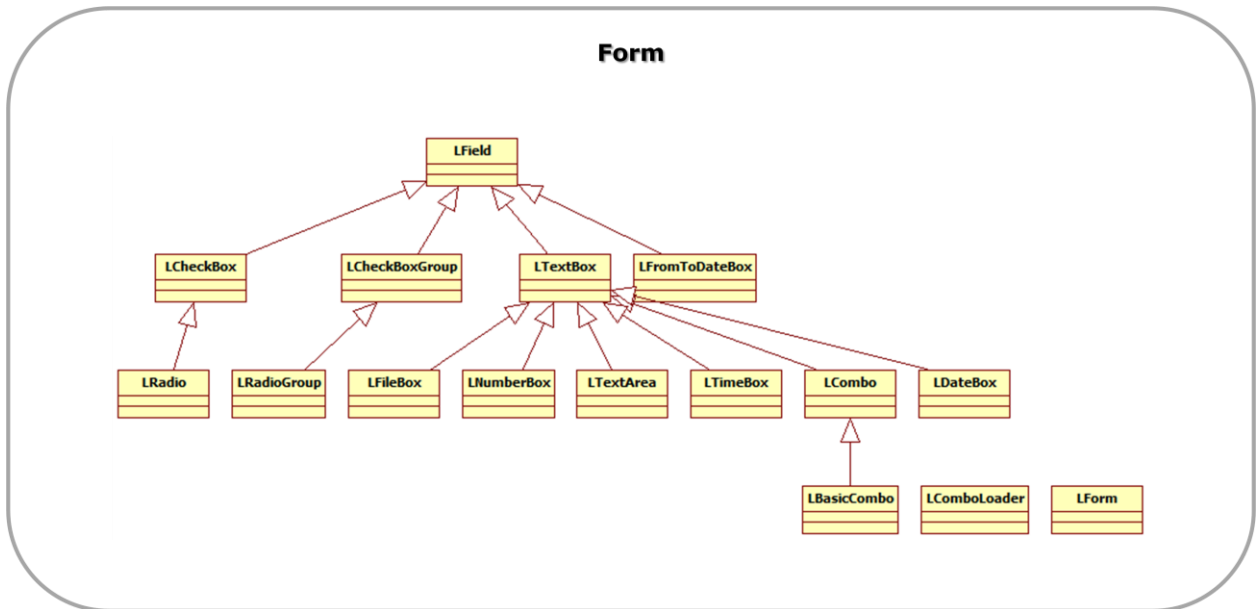
2.4.3 UI Class diagram

UI 컴포넌트들을 처리하기 위한 기본 Layer 로써 Panel, Button, Calender 등을 제공합니다. Grid, Tab, Tree 등 화면에 출력하는 UI 컴포넌트들을 사용할 경우 필수 Layer 입니다.



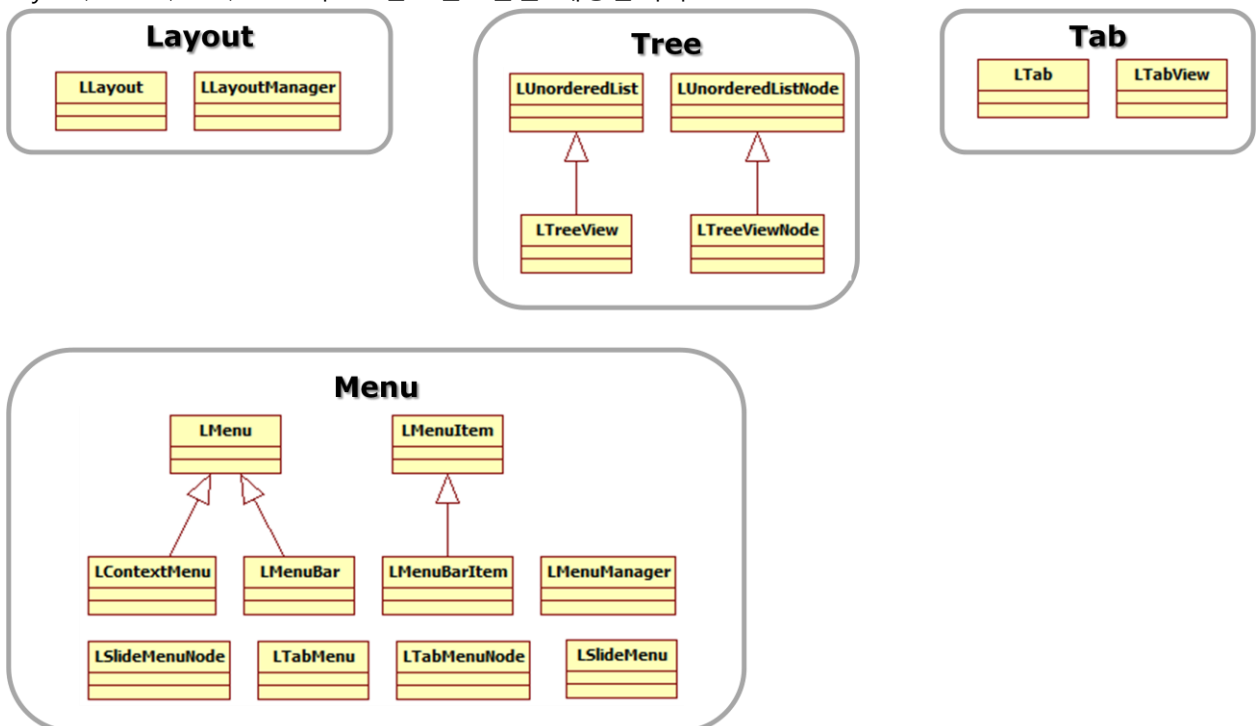
2.4.4 Form Class diagram

TextBox, Combo, DateBox 등 Form Object 관련 UI 컴포넌트들로서 기본 HTML 의 Form Object 에서 제공하는 기능보다 더 다양한 기능들 제공합니다.



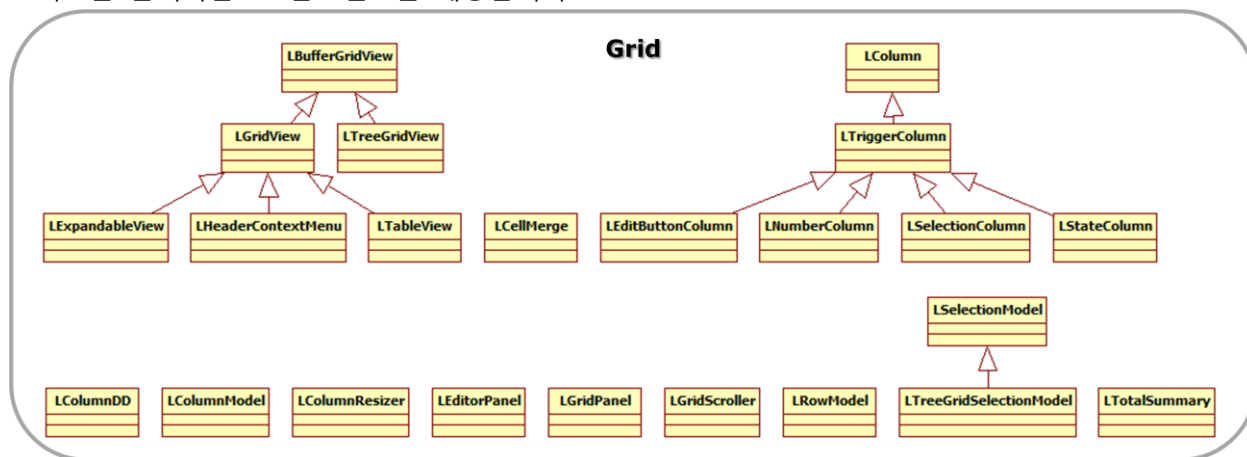
2.4.5 Layout/Menu/Tab/Tree Class diagram

Layout, Menu, Tab, Tree 의 UI 컴포넌트들을 제공합니다.



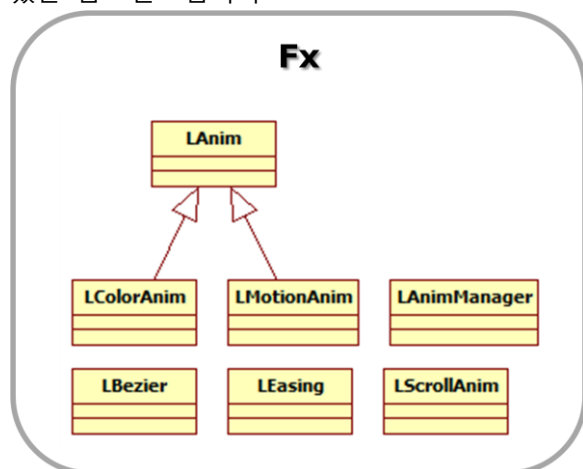
2.4.6 Grid/Fx Class diagram

그리드를 출력하는 UI 컴포넌트를 제공합니다.



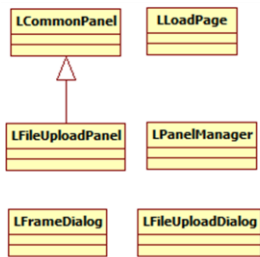
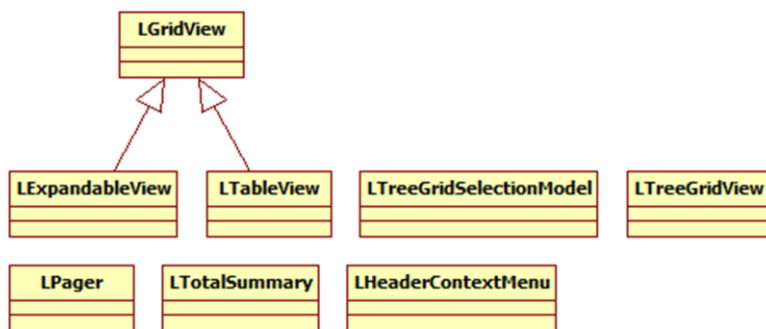
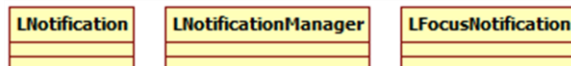
2.4.7 Fx Class diagram

다양한 Animation 효과 처리할 수 있는 UI 컴포넌트들로서 사용자에게 다이나믹한 UX 를 제공 할 수 있는 컴포넌트입니다.



2.4.8 Plugins utility Class diagram

Plugins 유틸리티는 기본 Layer 에서 제공되지 않는 다양한 UI 컴포넌트들을 제공합니다.

Panel**DragDrop****Animation****UnorderedList****Grid****Notification****Etc...**

3. 개발환경 설정

3.1 개요

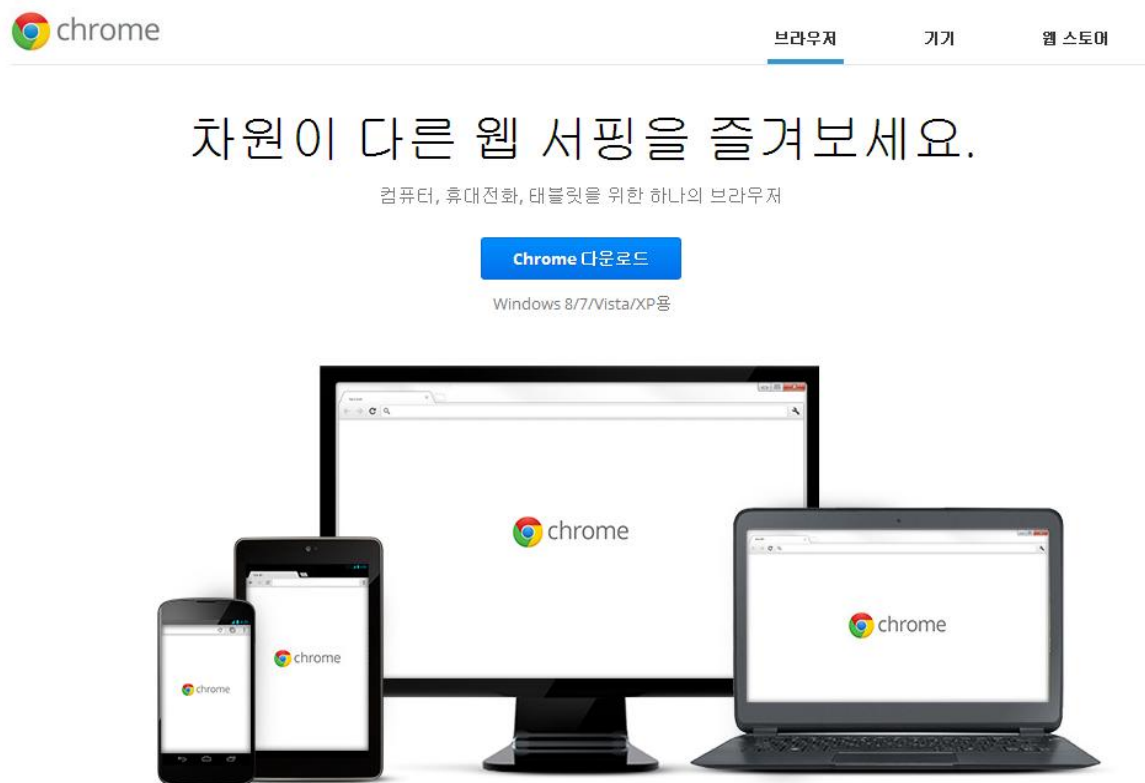
DevOn RichUI 를 개발하기 위한 개발환경 설정 방법을 설명하여 개발자로 하여금 개발환경을 설정하는 있어서 시행착오를 겪지 않고 일관된 개발환경을 갖출 수 있도록 가이드를 제공하는데 그 목적이 있다.

3.2 사전 설치 권장 프로그램

3.2.1 Chrome 브라우저

현재 웹 브라우저들 중에서 개발용으로는 Google 의 Chrome 브라우저의 선호도가 가장 높은 편이다. 또한 2013 년 상반기 기준 전세계 점유율 1 위의 브라우저인 만큼 웹 표준을 지키려면 반드시 사용법을 익혀둘 필요가 있다. Chrome 을 다운로드하기 위해서는 다음 주소를 방문하여 설치한다.

<https://www.google.com/intl/ko/chrome/browser/>

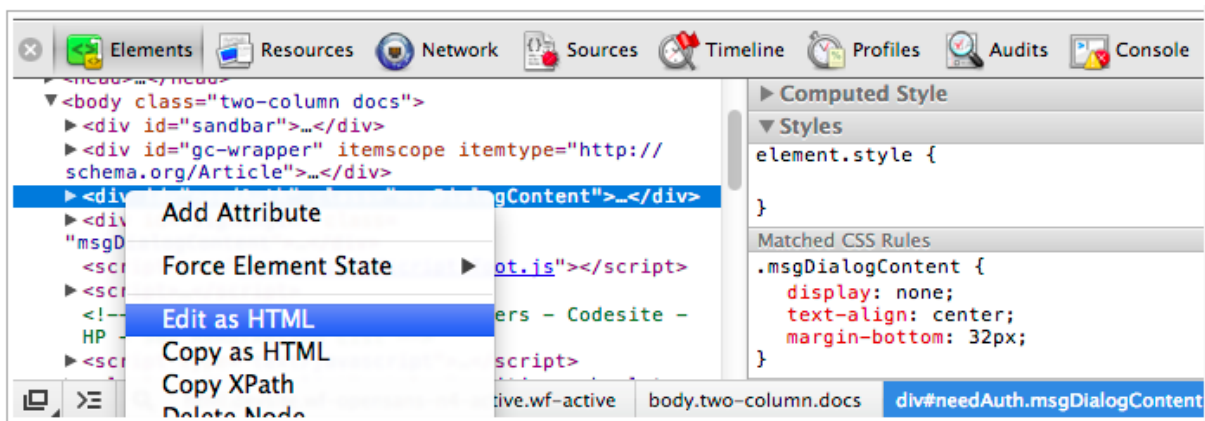


3.2.2 Chrome 개발자 도구

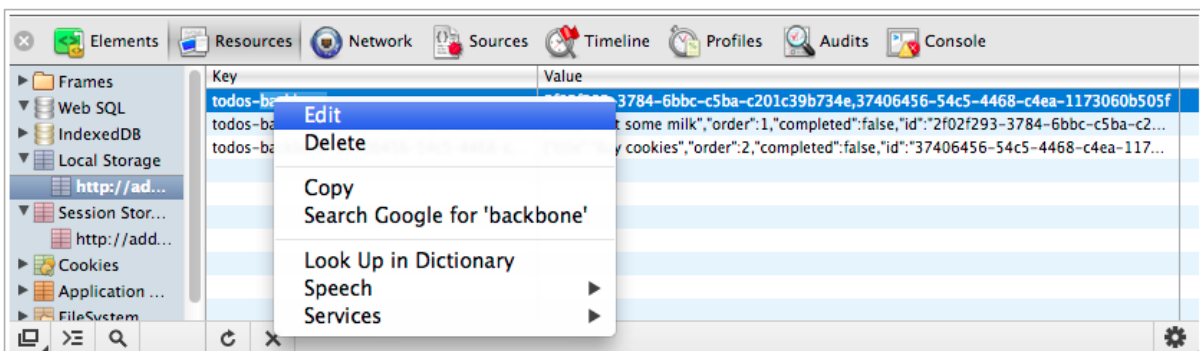
Javascript 개발 시 사용되는 개발자 도구는 현재 출시된 최신 브라우저들이 대부분 제공하고 있으며, 제공되는 기능도 메인 기능은 크게 차이가 없을 정도로 상향 평준화되어 있다. 이런 다양한 개발자 도구들 중 대표적으로 많이 사용되는 것이 Chrome 개발자 도구와 Firefox 의 개발자 도구인 Firebug 이다. 둘 다 높은 품질의 개발자 도구이며 그 중에서 본 가이드는 Chrome 의 개발자 도구를 살펴보기로 한다.

위에서 Chrome 설치가 끝나면 Chrome 을 실행한 후 F12 키를 눌러 개발자 도구를 연다. 브라우저 하단에 개발자 도구가 열린 모습을 볼 수 있다.

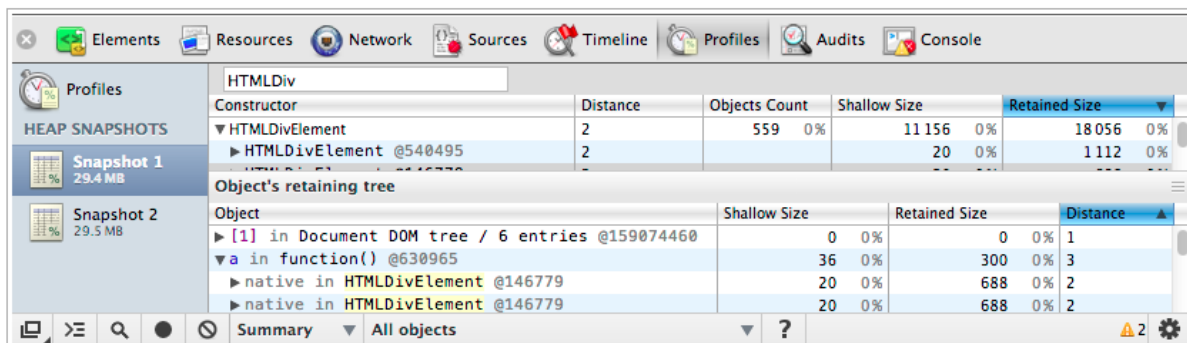
상단 각 탭 별 패널들의 주요 기능은 다음과 같다.



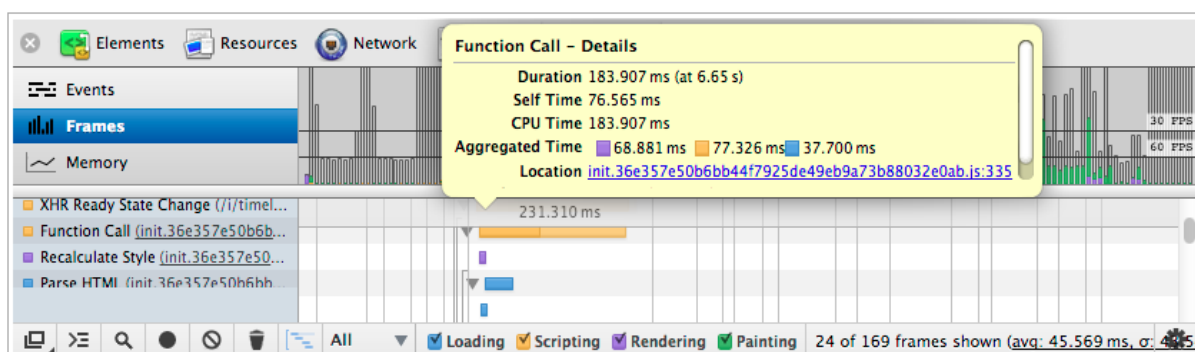
Elements 탭은 HTML Dom Element 와 그의 Style 정보를 확인할 수 있으며 임의로 속성을 변경하여 적용된 모습을 즉시 확인할 수 도 있다.



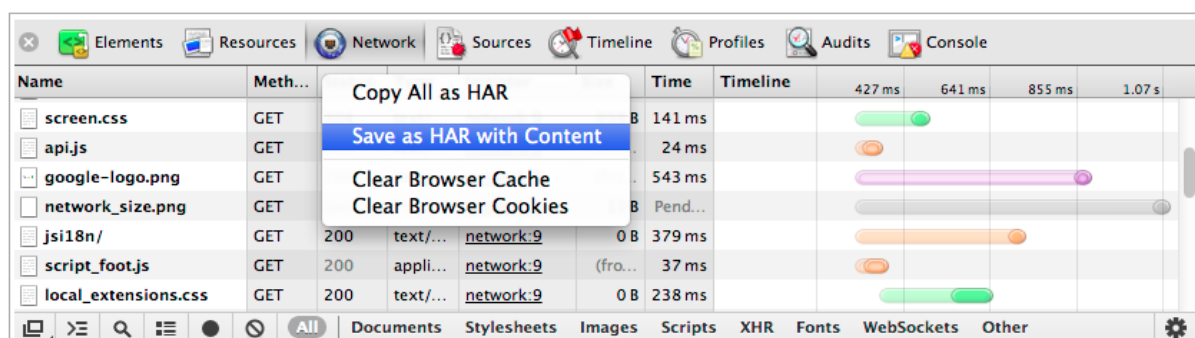
Resource 탭은 디버깅 모드에서 Local Storage, 캐시, 쿠키, 웹 SQL 등 각종 리소스 정보 들을 모니터링 할 수 있다.



Profile 탭은 실행시간이나 메모리 사용 상황 등을 분석하여 보여주는 등 종합적인 javascript 의 성능을 모니터 할 수 있다.



Timeline 탭은 웹 페이지가 실행되고 로딩되는데 소요된 시간들에 대한 정보를 시간의 흐름 순서대로 보여준다.



Network 탭은 실시간으로 요청과 다운로드 된 리소스들이 기대했던 것보다 더 오래 걸렸던 요청들을 구별하기 위한 기능을 제공한다.

기타 자세한 사용법은 Google 의 Chrome 개발자도구 설명 사이트에 방문하여 Chrome 을 이용한 디버깅 도움말들을 찾아 보도록 한다.

<https://developers.google.com/chrome-developer-tools/?hl=ko-KR>

참고로 Chrome 브라우저에서 기본적으로 방문한 페이지들이 캐시 되는데 새롭게 빌드 된 페이지를 Refresh 해도 변경된 페이지가 제대로 표시되지 않는 경우가 종종 발생한다. 이런 경우 "Ctrl+H"키를 눌러서 "방문 기록(H)"을 선택하고 "인터넷 사용정보 삭제"버튼을 선택하면 캐시 된 정보를 모두 버리고 화면이 새롭게 갱신된다. 개발 중 이 기능을 자주 사용하기를 추천한다.

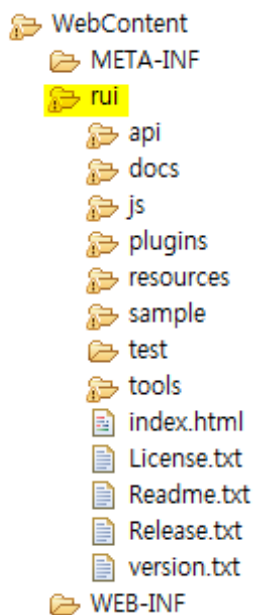
3.3 RichUI 배포본 구성

3.3.1 RichUI 설치

개발환경의 구성은 간단하다. RichUI 담당자로부터 제공 받은 RichUI 압축 파일을 웹 docRoot 에 "rui" 폴더 하위에 복사하여 넣으면 끝이다.

3.3.2 Folder 구조

아래 그림은 웹 docRoot 가 WebContent 인 폴더에 RichUI 를 설치한 모습이다.



주의!!

프로젝트 환경에 따라 웹 docRoot 바로 아래가 아닌 별도의 위치로 옮겨 설치할 경우 RichUI 의 각종 데모 및 샘플 등을 참조할 수 없으며, 문제 발생시에도 RichUI 담당자의 정상적인 지원을 받기가 어렵습니다.

각 폴더 별 역할을 간략히 살펴보면 아래 표와 같다.

폴더명	폴더 구성내용
api	API 웹 문서

docs	개발자 가이드문서, 튜토리얼, 릴리즈 정보
js	RichUI 의 기본 라이브러리 5 종
plugins	RichUI 의 기본 라이브러리에서 제외된 플러그인들
resources	RichUI 의 샘플 등에 사용되는 css, image, lang 파일, data, skin 등
sample	RichUI 의 데모 및 샘플페이지

3.3.3 api 폴더

<http://localhost:xxxx/rui/api> 를 브라우저에서 실행하면 API Document Page 가 열린다.

좌측은 RichUI 컴포넌트들의 API 트리 이며, 우측은 선택된 컴포넌트의 API 문서이다.

The screenshot shows the DevOn RichUI javascript framework API page. On the left, a search bar contains the text 'dataset'. Below it, a tree view lists various API components, with 'Rui.data.LDataSet' selected. The right pane displays the API documentation for 'Rui.data.LDataSet'. It includes the following information:

- Class:**
 - Rui.util.IEventProvider
 - Rui.data.LDataSet
 - Rui.data.LDelimiterDataSet
 - Rui.data.LJsonDataSet
 - Rui.data.LXmlDataSet
- public instance Class Rui.data.LDataSet**

컴포넌트와 연계된 데이터를 처리하는 객체
- Parameters:**

config <Object> The initial LDataSet.
- Constructors : 1개**

속 성	이 름	설 명
C	Rui.data.LDataSet	컴포넌트와 연계된 데이터를 처리하는 객체
- Configs : 7개**

속 성	이 름	설 명
*	defaultFailureHandler	rui_config.js 에 있는 기본 failure handler를 사용할지 여부를 리턴한다.

위 그림은 DataSet 에 대한 API 문서를 검색 한 모습이다.

좌측 상단 검색 창에서 dataset 을 입력 하면, dataset 을 키워드로 검색된 결과 목록이 나오며, 그 중에서 원하는 컴포넌트 이름을 선택하면 좌측 트리에 검색된 DataSet 컴포넌트의 API 들이 나열된다.

3.3.4 js 폴더

RichUI 의 주요 핵심 라이브러리 5 종을 모아놓은 폴더이다.

파일의 종류는 *-min.js, *-debug.js, *.js 등 세가지 파일로 구성되어 있으며,

*-min.js 파일은 스크립트를 불필요한 공백을 제거하여 압축한 형태이며 운영 환경 적용 시 최적의 성능을 발휘하기 위한 목적으로 배포된다.

*-debug.js 파일은 개발시점에 디버깅을 목적으로 배포되는 파일로 소스에 주석 및 공백 등이 모두 포함되어 있다.

*js 파일은 debug 파일에서 주석과 일부 디버깅 기능을 제거한 것으로 debug 파일을 사용하기 힘든 환경에서 사용할 수 있다.

3.3.5 docs 폴더

RichUI 관련 문서와 가이드들을 모아놓은 폴더들로, 하위에 guide 폴더는 RichUI 그리드 기능 사용법을 가이드 하는 문서, release 폴더는 버전 별 기능비교 문서, tutorials 는 RichUI 를 처음 사용하는 개발자를 위한 튜토리얼 문서가 각각 위치해있다.

3.3.6 plugins 폴더

js 폴더에 제공되는 기본 컴포넌트 이외에 추가적으로 제공되는 확장된 기능 컴포넌트들이 위치해 있다.

3.3.7 sample 폴더

RichUI 를 활용하여 만든 각 컴포넌트 별 샘플들을 모아놓은 폴더이다.

폴더명	내용
data	샘플에서 사용되는 각종 json 데이터
general	각 컴포넌트 사용 샘플
pattern	기존 프로젝트에서 분석된 화면 패턴 샘플

3.3.8 resources 폴더

RichUI 를 구성하고 있는 컴포넌트들의 레이아웃 및 스타일들을 적용하기 위한 CSS 파일들과 이미지 파일, 스킨 CSS 파일 등이 위치해있다.

특히 이 폴더 하위에는 다국어 관련 메시지를 제공하기 위한 locale 관련 파일들도 위치한다.

참고!!

Resource 폴더에서 rui_adapter.css 파일을 볼 수 있는데 이 파일은 DevOn UI Designer Template 을 RichUI 와 같이 사용하는 경우 일부 CSS 충돌로 인해 RichUI 의 컴포넌트들이 정상 동작하지 않는 경우가 발생하는데 이 때 사용해야 하는 파일이다.

이 경우 CSS 파일 링크 순서는

- (1) 기존 css 파일
- (2) rui_adaptor.css
- (3) RICHUI 관련 css 파일(rui_skin_xxx.css)

4. RichUI 설정

DevOn RichUI 의 rui_config.js 는 각각의 컴포넌트들이 가진 다양한 기본 설정 값들을 일괄 변경 하는데 사용된다. rui_config.js 의 구조를 파악하고 프로젝트 환경에 맞게 변경하는 방법을 알아본다.

4.1 구조

```
예) rui_config.js의 구조
(function() {
    ...
    var contextPath = "";
    var ruiRootPath = '/rui';
    Rui.config.ConfigData = {
        core: {
            ...
            defaultLocale: 'ko_KR',
            contextPath: contextPath,
            ruiRootPath: ruiRootPath,
            ...
        },
        base: {
            ...
        },
        ext: {
            ...
        },
        project: {
            ...
        }
    };
    ...
})();
```

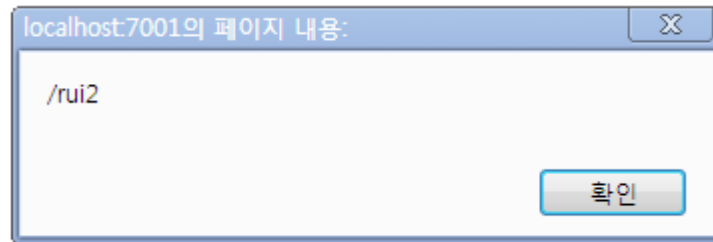
rui_config.js 의 구조는 Rui.config.ConfigData 객체(Json Object)로 구성되며 크게 core, base, ext 영역으로 나뉘어 있다.

이 값에 대한 접근은 Rui.getConfig() 메소드를 이용하여 우선 Rui.config.LConfiguration 객체를 얻은 후 Rui.config.LConfiguration 의 getFirst() 메소드를 호출하며 필요한 JsonPath 를 매개변수로 전달하면 된다.

```

예) rui_config.js의 설정값 얻어오기
var contextPath = Rui.getConfig().getFirst('$.core.contextPath'),
    ruiRootPath = Rui.getConfig().getFirst('$.core.ruiRootPath');
alert(contextPath + ruiRootPath);

```



Rui.config.LConfiguration 은 rui_config.js 에 정의된 Rui.config.ConfigData 로부터 제공된 JsonPath 에 해당하는 위치의 설정 값들을 사용할 수 있도록 지원하는 컴포넌트로 Rui.getConfig()은 이를 한번 더 감싸 짧은 코드로 간편하게 설정 값들을 사용할 수 있도록 제공된 것이다.

JsonPath 는 root 에 해당하는 "\$" 기호를 시작으로 하위 Object 의 Key 이름을 마침표(".")를 구분자로 하여 나열하면 된다.

이해를 돕기 위해 JsonPath "\$.core.contextPath"를 풀어 설명하면 아래 표와 같다.

\$	Root 에 해당하는 객체로 rui_config.js 의 Rui.config.ConfigData 에 해당한다.
core	\$.의 하위 core 에 해당하는 객체로 이 위치의 다른 값들로는 base, ext, project 등이 있다.
contextPath	\$.core 의 하위 contextPath 에 해당하는 값으로 이 위치의 다른 값들로는 defaultLocale, ruiRootPath 등이 있다.

4.2 rui_config.js 설정

4.2.1 RichUI 영역별 요소들

4.2.1.1 \$.core

core 영역은 RichUI 의 기본 설정 값들을 가진다. RichUI 의 버전 정보, RichUI 가 설치된 위치, locale, logger 설정, 디버깅 지원 도구, 웹접근성 사용 및 Optimizer 들의 설정값 들이 포함된다.

applicationName	"DevOn RichUI"
version	RichUI 의 버전인 "2.0"
configFileVersion	rui_config.js 의 버전
defaultLocale	기본값은 ko_KR 이며 시스템 환경에 맞게 변경 가능하다.

contextPath		WAS 에 설정된 별도의 contextPath 가 있는 경우 정의하여 사용한다.
ruiRootPath		RichUI 가 설치된 위치로 기본값은 <code>"/rui"</code> 이다.
message	defaultMessage	기본으로 사용하게 될 message 처리를 위한 locale 객체를 지정한다. 기본값은 <code>Rui.message.locale.ko_KR</code> 객체 이다.
	localePath	message locale 파일들이 위치하는 경로로서 다국어 지원을 위한 <code>lang-xx_XX.js</code> 파일들이 모여있는 위치이다. 기본값은 <code>"/resource/locale"</code> 이다.
Jsunit		RichUI 의 jsunit 을 이용하여 unit test 를 구현할 수 있으며 이 때 필요한 jsUnitCore.js 의 위치이다.
css		RichUI 의 css 가 load 될 때 사용할 character set 등을 지정한다. 기본값은 <code>"utf-8"</code> 이다.
debug		개발 중 오류 또는 경고사항 발생 시 이를 개발자에게 알리고 또한 잠재적 오류의 가능성이 발생 시에도 알려주는 RichUI 개발자용 디버깅 도구를 제공한다.
	notice	RichUI 디버깅 도구에서 잘못된 개발 또는 잠재적 위험성을 알려주는 Notice 기능을 활성화 할 지 여부로 기본값은 <code>true</code> 이다.
	servers	이곳에 등록된 IP 및 Host 들만이 Notice 기능을 사용할 수 있다.
optimizer		잘못된 RichUI 사용이나 과도한 컴포넌트 사용 등으로 인하여 성능 및 자원 활용에 악영향을 초래할 상황이 발생할 경우 경고하는 기능인 Optimizer 의 각종 제한 값들을 설정한다.
useAccessibility		RichUI 의 웹접근성 관련 WAI-ARIA 지원 기능을 사용할지 여부

4.2.1.2 \$.base

base 영역은 RichUI 의 기본 기능들에 대한 설정 값들을 가진다. RichUI 의 ajax 기능, button, DataSet 및 사용자 가이드에 대한 설정이 포함된다.

dataSetManager	Rui.data.LDataSetManager 를 사용하는데 필요한 기본 설정으로 defaultProperties 를 통해 기본 동작을 정의하고 실패 시의 핸들러와 오류 발생 시의 핸들러 등을 지정한다.
button	RichUI 의 Rui.ui.LButton 의 더블클릭 방지 기능을 설정한다.
dataSet	Rui.data.LDataSet 을 사용하는데 필요한 기본 설정으로 이 또한 Rui.data.LDataSetManager 와 유사하게 defaultProperties 를 통해 기본 동작을 정의하고 실패 및

		오류 발생 시의 Handler 를 지정한다.
guide		RichUI 가 제공하는 각종 가이드 및 튜토리얼 기능 사용에 대한 설정 값이다.
	show	가이드를 화면에 보일지 여부로 기본값은 false 이다.
	limitGuideCount	한번에 보여질 가이드의 개수를 제한한다. 기본값은 3 이다.

base 영역부터 사용되고 있는 defaultProperties 는 각 컴포넌트의 생성자 파라미터와 동일하게 동작한다.

ruiconfig 가 없다면 아래 예제와 같이 컴포넌트 생성시마다 생성자 매개변수로 변경되어야 할 속성값들을 넣어줘야 한다.

예) 만약 ruiconfig.js 가 없다면!

```
var dataSet = new Rui.data.LJsonDataSet({
  id: 'dataSet',
  focusFirstRow: 0,
  defaultFailureHandler: true,
  isClearUnFilterChangeData: false,
  readFieldFormatter: { date : Rui.util.LFormat.stringToTimestamp },
  writeFieldFormatter: { date : Rui.util.LRenderer.dateRenderer('%Y%m%d') }
  fields: [
    { id: 'col1' },
    { id: 'col2' },
    { id: 'col3' },
    ...
    { id: 'date1', type: 'date' }
  ]
});
```

ruiconfig 를 사용할 경우 각 컴포넌트별 설정 값에 정의된 defaultProperties 들은 자동으로 생성시에 반영되게 된다.

예) ruiconfig.js 를 사용한다면

```
var dataSet = new Rui.data.LJsonDataSet({
  id: 'dataSet',
  fields: [
    { id: 'col1' },
    { id: 'col2' },
    { id: 'col3' },
```

```

    ...
    { id: 'date1', type: 'date' }
  ]
});

```

4.2.1.3 \$.ext

ext 영역은 위에 설명된 core, base 를 제외한 전부를 포함하며, RichUI 의 확장된 기능들에 대한 설정 값들을 가진다. 모든 UI 컴포넌트들 및 플러그인 들이 이에 해당된다. ext 영역 또한 컴포넌트 별 defaultProperties 를 사용하게 되며 이 값은 API 문서를 통해 그 쓰임새 및 설정 값을 확인할 수 있다.

browser	recommend	RichUI 를 사용할 때 웹 표준을 지원하지 않는 브라우저로부터 접속했을 경우 이를 경고한다. 그리고 브라우저의 업그레이드를 추천하는 메시지를 출력하는 기능이 포함되어 있으며 이 기능을 활성화 하려면 recommend 값을 true 로 설정한다.
	recommendCount	브라우저 추천을 최대 몇 회 출력할지를 결정한다. 기본값은 3 회이다.
	link	사용자가 브라우저 추천을 받고자 할 경우 이동할 사이트 주소를 설정한다.
dialog		Rui.ui.LDialog 를 설정한다.
combo		Rui.ui.form.LCombo 를 설정한다. Combo 의 경우 Combo 의 목록 출력을 위해 Rui.data.LDataSet 을 사용하며 데이터 연결을 위한 설정도 포함된다.
textBox		Rui.ui.form.LTextBox 를 설정한다. TextBox 의 경우도 입력 자동완성 기능을 사용할 경우 DataSet 을 사용하며 연결을 위한 설정이 포함된다.
textArea		Rui.ui.form.LTextArea 를 설정한다.
numberBox		Rui.ui.form.LNumberBox 를 설정한다.
checkBox		Rui.ui.form.LCheckBox 를 설정한다.
radio		Rui.ui.form.LRadio 를 설정한다.
dateBox		Rui.ui.form.LDateBox 를 설정한다.
slideMenu		Rui.ui.menu.LSlideMenu 를 설정한다.
tabView		Rui.ui.tab.LTabView 를 설정한다.
treeView		Rui.ui.tree.LTreeView 를 설정한다.
calendar		Rui.ui.calendar.LCalendar 를 설정한다.
gridPanel		Rui.ui.grid.LGridPanel 을 설정한다.
grid		Rui.ui.grid.LBufferGridView 를 설정한다. Grid 의 경우 excel

	download 를 위한 excelDownloadUrl 속성 또한 지정된다.
gridScroller	Rui.ui.grid.LGridScroller 를 설정한다.
treeGrid	Rui.ui.grid.LTreeGridView 를 설정한다.
pager	Rui.ui.LPager 를 설정한다.

4.2.1.4 \$.project

Project 영역은 RichUI 의 컴포넌트를 위한 설정이 아닌, 프로젝트를 위해 할당된 영역이다. Project 영역에 필요한 설정 값들을 지정한 후 Rui.getConfig()을 이용하여 사용하면 된다.

5. RichUI 배포

DevOn RichUI 를 이용하여 개발된 어플리케이션들을 배포할 때 고려해야 할 사항들이 있다.

5.1 min 및 debug

DevOn RichUI 는 5 개의 기본 라이브러리를 제공한다. base, core, ui, form, grid 로 분할되어 있는 RichUI 의 기본 라이브러리는 파일들은 rui_를 prefix 로 하며 js 를 확장자로 하여 배포되고 있다.

rui_base.js , rui_ore.js	RichUI 의 코어와 유틸리티, 설정 기능들이 포함되어 있다.
Rui_ui.js, rui_form.js, rui_grid.js	RichUI 의 기본 제공 UI 컴포넌트들이 이곳에 모두 포함되어 있다. (플러그인 UI 들은 /plugins 폴더로 별도 제공)

어플리케이션을 개발할 때 위 기본 라이브러리들을 사용하게 되는데, RichUI 는 개발 중 사용할 라이브러리와 배포 시 사용할 라이브러리를 구분하고 있다.

rui_base-debug.js rui_core-debug.js rui_ui-debug.js rui_form-debug.js rui_grid-debug.js	어플리케이션 개발 중 사용할 라이브러리로 파일마다 -debug suffix 가 붙어있으며, 이 라이브러리 파일 속엔 RichUI 개발 시 필요한 모든 주석 및 API 들이 포함되어 있다. RichUI 의 js 들 중 에서 파일의 크기가 가장 크기 때문에 해당 파일 사용 시 성능은 다소 떨어지나 오류 발생 시 정확한 위치 추적이 용이하다.
rui_base-min.js rui_core-min.js rui_ui-min.js rui_form-min.js rui_grid-min.js	어플리케이션 개발이 모두 끝나고 서비스가 운영될 서버로 배포할 때 사용할 라이브러리로 파일마다 -min suffix 가 붙어 있다. 이 라이브러리 파일은 모든 주석 및 space, CR-LF 등이 제거되어 압축되어 있으므로 RichUI 의 js 들 중 파일들의 크기가 가장 적어 최적의 성능을 발휘한다.

어플리케이션 개발도중 오류가 발생하여 디버깅을 하고자 할 경우 -debug 파일을 사용하기를 권장하며, 개발이 끝난 어플리케이션을 운영 서버로 배포할 경우 반드시 -min 파일로 변경하기를 권장한다.

이러한 debug, min 파일 구조는 일부 RichUI 플러그인도 동일하게 적용되므로, 플러그인 사용 시에도 배포 전에 배포할 파일에 대한 확인 작업이 필요하다.

6. UI 컴포넌트

6.1 FORM 컴포넌트

DevOn RichUI 의 FORM 컴포넌트를 설명한다.

6.1.1 FORM 컴포넌트 종류

웹 브라우저가 지원하는 기본 표준 FORM 컴포넌트들을 보다 쉽게 사용할 수 있고 강력한 기능들을 제공하는 RichUI 의 FORM 컴포넌트들은 아래 목록과 같이 제공된다.

컴포넌트	주요 기능
Rui.ui.form.LTextBox	<INPUT type="text">를 기반으로 구현된 기능이다. 대표적인 기능으로는 Placeholder, Mask, AutoComplete (자동완성, 제시어)기능 등을 제공한다.
Rui.ui.form.LTextArea	<TEXTAREA>를 기반으로 구현된 기능이다. TextBox 를 상속받고 있기 때문에 TextArea 이전에 TextBox 의 기능을 고스란히 사용할 수 있다.
Rui.ui.form.LCheckBox	<INPUT type="checkbox">를 기반으로 구현된 기능이다.
Rui.ui.form.LRadio	<INPUT type="radio">를 기반으로 구현된 기능이다.
Rui.ui.form.LCheckBoxGroup	CheckBox 를 두개 이상 사용할 경우 그룹으로 묶어 한번에 관리할 수 있다.
Rui.ui.form.LRadioGroup	Radio 의 경우 일반적으로 두개 이상이 그룹으로 묶여 사용되어야 한다.
Rui.ui.form.LCombo	TextBox 에 목록을 붙여 조합된 형태의 기능이다. 표준 Combo 와 다르게 사용자가 직접 값을 Key in 할 수 있으며, 이 역시 TextBox 를 상속받고 있어 TextBox 의 기능들도 활용할 수 있다. 별도로 <SELECT>를 기반으로 제공되는 BasicCombo 플러그인이 제공된다.
Rui.ui.form.LNumberBox	TextBox 에 Numeric 값들만 입력이 가능하도록 기본적인 Mask 기능이 입혀진 기능이다. 당연히 TextBox 를 상속받고 있으며 min/max 속성을 통해 입력 받을 값의 범위를 지정할 수 있다.
Rui.ui.form.LDateBox	TextBox 에 Rui.ui.calendar.Lcalendar 를 더한 기능으로 사용자가 날짜를 선택함에 편의성을 제공한다. 역시 TextBox 를 상속받고 있다.
Rui.ui.form.LTimeBox	TextBox 에 시간(12:00)을 표현한 Mask 기능을 더한 기능이다.

이러한 FORM 객체들은 Rui.data.LDataSet 과 Rui.data.LBind 등과 연계되어 값들이 관리되며, Rui.validate.LValidatorManager 에 의해 손쉽게 입력된 값에 대한 검증도 가능하다.

위 FORM 객체들은 향후 설명될 RichUI Grid 의 편집기로도 사용되며, Grid 의 편집기에서도 위 모든 기능들이 동일하게 지원된다.

6.1.2 FORM 의 기능

Rui.ui.form.LForm 은 <FORM>을 기반으로 한다. FORM 과 동일하게 Method, Action 등을 지정 후 Submit 하는 일반적인 기능뿐 아니라, FORM 내부 FORM 객체들의 값을 간편하게 변경하는 기능과, DataSet 의 레코드를 FORM 객체들에 일괄 반영하는 기능 등의 편의기능들을 제공한다.

특히 Rui.ui.form.LForm 을 이용하면 비동기식 Multipart FORM 전송도 가능하다.

6.1.3 FORM 컴포넌트 사용

6.1.3.1 TextBox

TextBox 의 사용 방법을 알아보도록 한다.

예) textbox를 생성할 위치의 Markup

```
...
<div>
    <input type="text" id="phoneNumber">
</div>
...
```

우선 TextBox 가 생성될 위치에 Markup 을 작성한다. INPUT, DIV 태그 등을 이용할 수 있으며, TextBox 생성시 이 INPUT 태그는 속성에 따라 재 작성 된다.

예) textbox 생성 스크립트

```
var phoneNumberBox = new Rui.ui.form.LTextBox({
    applyTo: 'phoneNumber',
    width: 110,
    placeholder: '전화번호',
```

```

mask: '999-9999-9999',
name: 'phone',
attrs: {
  tabIndex: 1,
  maxLength: 13'
}
});

```

TextBox 를 생성하는 스크립트다. new Rui.ui.form.LTextBox()를 실행하며 생성인자를 JSON 형태로 제공하는 모습이다.

위 예제에서 사용된 생성인자의 값들은 다음과 같다.

생성인자(Config)	기능
applyTo	applyTo 는 TextBox 를 포함한 모든 컴포넌트가 생성되어 그려질(랜더링) 위치가 될 DOM 의 ID 이다.
width/height	TextBox 의 크기를 조절한다. width 와 height 으로 지정할 수 있으며 값은 숫자 형으로 부여해야 한다.
placeholder	TextBox 에 값이 입력되기 전 빈 상태일 때 사용자에게 보여질 문자열을 입력한다. HTML5 부터 기본 제공되는 기능이나, HTML5 가 제공되지 않는 브라우저를 위해 이 기능을 사용하는 것이 좋다.
mask	TextBox 에 값이 입력되는 형식(Format)을 지정한다. 위 예제는 전화번호 형식에 맞게 부여한 예이다.
name	FORM 전송될 INPUT 태그의 name 속성값을 지정한다.
attrs	attrs 속성은 Markup 에 직접 부여하고자 하는 속성을 지정한다. tabIndex: 1 을 지정한 위 예는 TextBox 가 생성된 후 랜더링된 INPUT DOM 에 tabIndex 속성을 1 로 부여한 것과 동일하다.

위 예제를 실행한 모습의 그림이다.



Placeholder 에 의해 "전화번호" 라는 문자열이 입력 전에 표시되고 있으며, Mask 에 의해서 입력 가능한 만큼 "_"로 표기되고 있다.

6.1.3.2 Combo

Combo 의 사용 방법을 알아보도록 한다.

Combo 의 경우 다른 컴포넌트와 다르데 Combo 의 목록 생성을 위한 데이터 URL 이 필요하다.

예) combo를 생성할 위치의 Markup

```
...
<div>
    <select id="code"></select>
</div>
...
```

우선 Combo 가 생성될 위치에 Markup 을 작성한다. SELECT, DIV 태그 등을 이용할 수 있으며, Combo 생성시 이 SELECT 태그는 향후 INPUT 태그로 재 작성 된다. RichUI 의 Combo 는 목록과 목록을 펼치는 버튼아이콘을 가진 TextBox 이기 때문이다.

예) combo 생성 스크립트

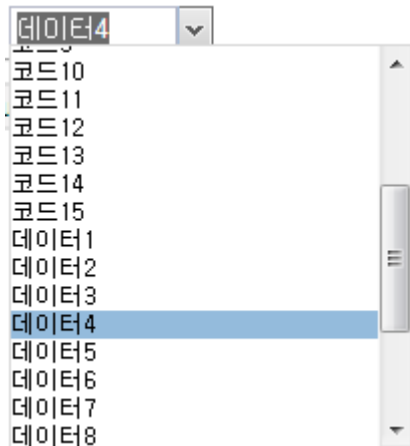
```
var combo = new Rui.ui.form.LCombo({
    applyTo: 'code',
    name: 'code',
    url: './codes.json'
});
```

Combo 를 생성하는 스크립트다. new Rui.ui.form.LCombo()를 실행하며 생성인자로 목록을 구성할 URL 을 지정한 모습이다.

Combo 는 위에 설명한 대로 TextBox 에 목록을 붙인 형태이다. 결국 TextBox 를 상속받아 구현된 컴포넌트기에 "url" 생성인자 등을 제외하곤 TextBox 와 생성인자가 동일하다.

생성인자(Config)	기능
url	Combo 의 목록을 구성할 데이터의 URL
dataSet	URL 대신 목록을 구성할 dataSet 을 직접 지정할 수 있다.
valueField	목록 DataSet 에서 값(value)으로 사용할 필드의 ID 기본값은 'code'이다.
displayField	목록 DataSet 에서 표시값(text)으로 사용할 필드의 ID 기본값은 'value'이다. 주의) 이 값이 정확하지 않을 경우 목록이 나타나지 않는다.

위 예제를 실행한 모습이다.



Combo 의 목록을 URL 과 DataSet 만으로 구성할 수 있는 것은 아니다. "SELECT" 태그 안에 있는 "OPTION" 태그를 이용하여 목록을 구성할 수도 있다.

예) 목록의 값이 구성된 combo를 생성할 위치의 Markup

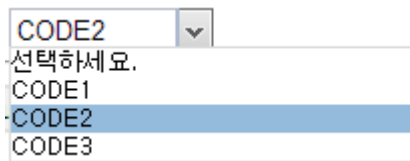
```
...
<div>
  <select id="code">
    <option value="code1">CODE1</option>
    <option value="code2">CODE2</option>
    <option value="code3">CODE3</option>
  </select>
</div>
...
```

이런 경우 Combo 생성에는 URL 이나 DataSet 이 필요없다.

예) URL을 사용하지 않는 combo 생성 스크립트

```
var combo = new Rui.ui.form.LCombo({
  applyTo: 'code',
  name: 'code'
});
```

결과를 보면 아래와 같이 "SELECT" 태그의 "OPTION" 값들이 목록에 나타난 것을 확인할 수 있다.



6.1.3.3 CheckBox & CheckBoxGroup

Checkbox 의 사용방법을 알아보도록 한다. 또한 Checkbox Group 을 이용하여 하나 이상의 Checkbox 를 보다 쉽게 생성하고 관리하는 방법을 알아본다.

예) checkbox를 생성할 위치의 Markup

```
...
<div>
  <input type="checkbox" id="cuttlefish" />
  <div id="peanut"> </div>
</div>
...
```

우선 Checkbox 가 생성될 위치에 Markup 을 작성한다. INPUT, DIV 태그 등을 이용할 수 있으며, Checkbox 생성시 이 INPUT 태그는 속성에 따라 재 작성 된다.

예) 두개의 checkbox 생성 스크립트

```
var cuttlefish = new Rui.ui.form.LCheckBox({
  applyTo: 'cuttlefish',
  name: 'appetizer',
  label: 'Cuttlefish',
  value: '오징어'
});
var peanut = new Rui.ui.form.LCheckBox({
  applyTo: 'peanut',
  name: 'appetizer',
  label: 'Peanut',
  label: '땅콩'
});
```

Checkbox 를 생성하는 스크립트다. new Rui.ui.form.LCheckBox() 로 생성하며 생성인자를 JSON 형태로 제공하는 모습이다.

위 예제는 Cuttlefish 와 Peanut 두개의 Checkbox 를 각각 생성하였는데 이번엔 Checkbox Group 을 이용하여 한번에 관리하는 방법을 알아보도록 한다.

예) checkbox group을 생성할 위치의 Markup

```
...  
<div>  
    <div id="fruits"></div>  
</div>  
...
```

우선 Checkbox Group 이 생성될 위치에 Markup 을 작성한다. INPUT 태그는 이용 할 수 없고, DIV 태그 등을 이용해야 한다.

예) 세개의 checkbox를 checkbox group을 이용하여 생성하는 스크립트

```
var fruits = new Rui.ui.form.LCheckBoxGroup({  
    applyTo: 'fruits',  
    name: 'fruit',  
    items: [  
        new Rui.ui.form.LCheckBox({  
            label: 'Apple',  
            name: 'fruit',  
            value: 'apple'  
        }), {  
            label: 'Grape',  
            value: 'Grape'  
        }, {  
            label: 'Strawberry',  
            value: 'Strawberry'  
        }  
    ]  
});
```

Checkbox Group 생성시의 생성인자인 items 속성에 정의된 대로 Apple, Grape, Strawberry 이렇게 3 개의 Checkbox 가 하나의 Group 으로 묶여 생성된다.

위 예제에서 사용된 생성인자의 값들은 다음과 같다

생성인자(Config)	기능
label	Checkbox 의 label 로 출력할 값
name	FORM 전송 등으로 서버에 전송될 때 인식될 키 값
value	객체가 checked 상태일 때의 값
items	CheckBox Group 에 포함될 Checkbox 속성 셋 배열

Checkbox 나 Checkbox Group 의 값이 변경된 경우 다음과 같이 이벤트로 알 수 있다.

```
예) checkbox 또는 checkbox group의 값이 바뀌면 해당 값을 alert으로 출력

fruits.on('changed', function(e){
    Rui.log('Changed Checkbox Value : ' + e.target.getValue());
});
```

다음은 위 예제를 실행한 모습의 그림이다.

☒ Cuttlefish ☒ Peanut ☐ Apple ☒ Grape ☐ Strawberry

6.1.3.4 Radio & RadioGroup

Radio 는 Checkbox 를 상속 받고 있어 그 기본이 Checkbox 와 같으며, 사용방법 또한 같다. Radio 의 특성상 같은 name 속성이면 단 하나의 Radio 만 선택된다.

☐ Spear ☒ Shield ☐ Rock ☒ Paper ☐ Scissors

6.1.3.5 Rui.ui.form.LForm

Form 객체를 활용하여 form 을 구성하고 있는 폼 요소들을 다룰 수 있다. 이 Form 객체를 어떻게 활용하는지 알아보겠다.

Form 을 구성하는 요소에는 input, textarea, select 등이 있으며 이 개체들을 RichUI 에서 제공하는 DataSet 또는 Validation 기능을 이용하여 편리하게 폼 객체들을 이용할 수가 있다.

Form 을 사용하는 방법은 다음과 같다.

Form 에서는 fieldset 을 구성하고 있는 각 요소들에 대해서 일괄적으로 disable 하거나 개별적으로 특정 Form 요소를 핸들링 할 수 있다.

예) Form 사용을 위한 Markup

```
<form id="frm" name="frm" method="post" action="./.././data/savedCallback.json">
<fieldset>
  <input type="hidden" id="test" name="test"/>
  <br/>
  col1:<input type="text" id="col1" name="col1" />
  <br/>
  col2:<input type="text" id="col2" name="col2" />
  <br/>
  col3:
  <select id="col3" name="col3">
    <option value="all">ALL</option>
    <option value="tes3">TES3</option>
    <option value="tes4">TES4</option>
  </select>
  <br/>
  col4:<input type="radio" id="col4_1" name="col4" value="RADIO1"/>RADIO1
  <input type="radio" id="col4_2" name="col4" value="RADIO2"/>RADIO2
  <br/>
  col5:<input type="checkbox" id="col5_1" name="col5" value="CHECKBOX1"/>CHECKBOX1
  <input type="checkbox" id="col5_2" name="col5" value="CHECKBOX2"/>CHECKBOX2
  <br/>
  col6:<textarea id="col6" name="col6" rows="5" cols="50"> </textarea>
  </fieldset>
</form>
```

예) Form 객체 생성

```
var form = new Rui.ui.form.LForm('frm');
```

이번에는 Form 객체를 이용하여 입력된 값을 서버로 전송시 입력 필드의 값에 대한 유효성을 체크하는 부분이다. 폼을 구성하는 요소는 매우 많으므로 하나하나 개별적으로 직접 핸들링 하기에는 많은 시간이 소요될 것이다. RichUI 에서는 폼 필드요소들에 대한 일괄적인 유효성을 검사하여 개발 생산성을 높여준다.

예) Form 객체를 이용한 유효성 검사

```
var submitBtn = new Rui.ui.LButton('submitBtn');
```



```

submitBtn.on('click', function(){
    var validatorManager = new Rui.validate.LValidatorManager({
        validators: [{
            id: 'col1',
            validExp: 'Col1:true:minLength=6'
        }, {
            id: 'col2',
            validExp: 'Col2:true:length=6'
        }, {
            id: 'col8',
            validExp: 'Col8:true'
        }]
    });

    validatorManager.add('col1', new Rui.validate.LMaxLengthValidator('col1', 8));
    form.on('beforesubmit', function(){
        return validatorManager.validateGroup('frm');
    });
    form.submit();
});

```

위 예에서 'submit' 버튼을 눌렀을 경우 validatorManager 를 통해 각 해당 Form 필드 요소에 대한 유효성 체크 조건(validExp)을 기술하고 서버로 전송하기 전에 'beforesubmit' 이벤트에서 전체 Form 인 'frm'에 대한 유효성을 체크하고 값을 전송한다.

다음은 폼요들에 대한 활성화 또는 비활성 기능을 제공하는 예제이다.

```

var enableBtn = new Rui.ui.LButton('enableBtn');
enableBtn.on('click', function(){
    form.enable();
});

var disableBtn = new Rui.ui.LButton('disableBtn');
disableBtn.on('click', function(){
    form.disable();
});

```

다음은 DataSet 과 Form 필드를 연동하여 값을 변경하기 위한 예제이다.

```

var dataSet = new Rui.data.LJsonDataSet({
    id: 'dataSet',
    fields: [{

```

```

        id: 'col1'    // 마크업 필드 col1 와 매칭
    }, {
        id: 'col2'    // 마크업 필드 col2 와 매칭
    }, {
        id: 'col3'    // 마크업 필드 col3 와 매칭
    }
    });

dataSet.load({
url: './.././../sample/data/data.json',
    method: 'get'
});

var updateRecordBtn = new Rui.ui.LButton('updateRecordBtn');
updateRecordBtn.on('click', function(){
if (dataSet.getCount() > 0)
form.updateRecord(dataSet.getAt(0)); // 폼필드에 데이터셋 첫번째 레코드값을 반영
});

```

위 코드를 살펴보면 Form 필드의 값을 DataSet 과 연동하여 변경하기 위해 먼저 DataSet 필드를 Form 필드와 동일하게 정의하고 'updateRecordBtn'을 선택했을 경우 form.updateRecord 함수를 통해 DataSet 첫번째 레코드를 Form 필드에 업데이트하여 col1, col2, col3 등 Form 필드의 값이 변경되는 것을 확인할 수가 있다.

6.2 Button 사용

HTML 의 button 기능을 확장한 object 이다.

6.2.1 Button

Button 을 적용 가능한 html element 는 <input type="button">, <button>이다. 혹은 document.body 에 직접 rendering 할 수 있다. LElement 를 상속 받으므로, LElement 의 대부분의 기능을 사용할 수 있고 addClass() 메소드 등을 사용해 CSS 를 추가한 후 아래 그림처럼 버튼에 별도의 image 를 첨가할 수도 있다. 실제로는 span element 가 button 의 container 가 된다.



```

var button1 = new DU.widget.LButton('HTMLElement_id',{
    label : "save",

```

```

        disableDbClick : true
    });
    button1.on("click", function(){
        // 버튼 클릭 처리
    });

```

생성인자(Config)	기능
label	버튼의 text 를 기술하지 않으면 tag 에 있는 값을 사용한다.
disableDbClick	반복 요청 제한을 위한 더블클릭 방지 기능. Default 0.5 초

6.3 데이터 처리

RichUI 에서 데이터 처리의 가장 핵심은 Rui.data.LDataSet 과 Rui.data.LDataSetManager 두개의 컴포넌트 이다.

데이터를 서버로부터 조회하여 그 변경된 값과 상태 등을 관리하는 DataSet 과 이를 다시 서버로 보내 저장하는 DataSetManager 는 RichUI 의 대부분의 UI 컴포넌트(Grid, Menu, Tree 등)에서 사용되며 이들의 데이터소스 가 된다.

6.3.1 DataSet

Rui.data.LDataSet 은 Rui.data.LRecord 와 Rui.data.LField 로 구성되어 있으며, 데이터 값의 상태 관리 및 이벤트를 제어하며, 모델 객체로써 UI 콤포넌트에 출력할 데이터를 관리한다.

6.3.1.1 DataSet 의 필수 Config

필수 생성인자는 rui_config.js 에 기본으로 정의되어 있어 생성시 rui_config.js 에 정의되지 않는 config 만 정의하면 된다.

생성인자(Config)	기능
id	DataSet 을 구분하는 아이디이다.
fields	LField 정보를 선언하는 배열이다.
focusFirstRow	서버에서 데이터를 load 한 후 처음 rowPosition 의 값일 지정한다. rui_config.js 에 0 으로 설정되어 있다.
remainRemoved	DataSet 에서 Record 를 삭제시 row 를 내부 배열에서 삭제할 것인지 상태값만 변경할지를 결정한다. 기본값은 false 로

	Record 가 삭제되면 내부 배열에서 삭제된다.
defaultFailureHandler	서버에서 데이터 load 시 에러가 발생하면 rui_config.js 에 정의된 loadExceptionHandler 가 수행될지 여부를 결정한다. 기본값으로 true 이면 일반적으로 에러는 모두 rui_config.js 에 정의한 규칙을 따른다.

6.3.1.2 DataSet 의 생성

DataSet 을 id 와 fields 정보를 필수로 입력 받아 생성한다. 생성시에는 JSON 방식, XML 방식, CSV(Delimiter) 방식에 따라서 생성자 객체를 사용할 수 있다. 일반적으로는 JSON 방식인 Rui.data.LJsonDataSet 을 사용한다.

서버로부터 데이터를 조회하려면 load 메소드를 이용한다.

```
var dataSet = new Rui.data.LJsonDataSet({
    id: 'dataSet',
    fields: [
        { id: 'col1', type: 'number', defaultValue: 0 },
        { id: 'col2', type: 'date', defaultValue: new Date() },
        { id: 'col3' },
        { id: 'col4' },
        { id: 'col5' }
    ]
});

dataSet.load({
    url: './test.rui'
});
```

DataSet 이 생성된 후 서버로부터 데이터가 load 되면 행의 위치값(rowPosition)을 가진다. DataSet 의 기본 위치값은 -1 이나 rui_config.js 에 설정된 기본값에 따라서 0 으로 바꿀 수 있다.

6.3.1.3 DataSet 의 이벤트

DataSet 이 지원하는 다양한 이벤트들 중 주로 사용되는 몇몇 이벤트이다.

이벤트명	기능
add	Record 객체가 추가될 경우 수행하는 이벤트

update	Record 객체의 값이 변경될 경우 수행하는 이벤트
remove	Record 객체가 삭제될 경우 수행하는 이벤트
load	데이터 값이 load 된 후 호출되는 이벤트
rowPosChanged	Row 의 Position 값이 변경 된 후 수행하는 이벤트
canRowPosChange	Row 의 Position 값이 변경되기 전 변경 가능 여부를 체크하는 이벤트로 일반적으로 유효성 체크로 사용한다.

6.3.2 Field

Field 는 DataSet 의 기준이 되는 키의 설정 정보로써 type 과 신규 추가시 기본 설정값(defaultValue)으로 구성된다. Type 은 string, number, date 의 3 가지 속성값을 가진다. Type 을 정의하지 않으면 기본으로 string 값으로 처리된다. defaultValue 는 DataSet 에서 신규 추가시 기본적으로 가지는 값을 정의한다.

6.3.3 Record

Record 는 DataSet 에서 row 정보를 처리하는 객체로 서버에서 받은 row 의 실제 데이터를 가지고 있다. 또한 row 의 상태관리, 값 변경, 데이터 serialize(데이터 직렬화)를 처리한다.

6.3.3.1 Record 의 생성

Record 객체는 DataSet 의 createRecord 메소드를 통해서 생성한다. 생성된 Record 는 DataSet 에 add 메소드를 실행 하여야 DataSet 에 추가된다. Record 가 추가되면 "add" 이벤트가 발생되며 Record 객체의 set 메소드를 이용하여 값을 변경하면 DataSet 객체에서 update 이벤트가 발생한다.

```
var data = {
    col1: '값1',
    col2: '값2'
};
var record = dataSet.createRecord(data);
dataset.add(record);           //add 이벤트 발생
record.set('col1', '값 변경'); //update 이벤트 발생
```

6.3.4 JSON DataSet 과 Delimiter DataSet

DataSet 은 기본적으로 JSON 형 문자열을 사용하며, 이 외에도 XML, Delimiter 뿐 아니라 원한다면 어떠한 포맷이든 사용할 수 있도록 확장이 가능하다. LJsonDataSet, LXmlDataSet 은 모두 LDataSet 을 상속받고 있으며, 입출력 기능만을 담당하고 있으며 이외의 모든 기능은 LDataSet 을 통해서 제어된다.

성능으로는 Delimiter > JSON > XML 순이며, 사용 편리성으로는 JSON > XML > Delimiter 순서이다.

```
var dataSet = new Rui.data.LJsonDataSet({    // json 방식
    id: 'dataSet',
    fields: [
        .....
    ]
});

var dataSet = new Rui.data.LXmlDataSet({    // xml 방식
    id: 'dataSet',
    fields: [
        .....
    ]
});

var dataSet = new Rui.data.LDelimiterDataSet({ // delimiter 방식
    id: 'dataSet',
    fields: [
        .....
    ]
});
```

6.3.5 DataSetManager

DataSet 은 서버로부터 데이터를 가져올 경우에 만 사용 가능하고 변경된 데이터를 서버로 다시 전송하고자 할 경우에는 DataSetManager 를 사용한다. 또한 여러 개의 데이터를 한번에 서버로부터 가져와 여러 개의 DataSet 에 각각 탑재할 경우에도 DataSetManager 를 이용한다.

6.3.5.1 DataSet의 변경 데이터 서버 전송

DataSet 내의 변경된 데이터를 서버로 전송하기 위해서는 DataSet의 updateDataSet 메소드를 실행한다. 이때 서버로 전송되는 데이터는 DataSet 내의 전체 데이터가 아닌 변경된 데이터 뿐이다. 이는 서버에서 변경된 데이터만을 처리(저장)하기 위함이며 필요시 전체 데이터를 모두 전송할 수 있다.

서버에서 데이터 처리가 완료되면 success 이벤트가 발생하며 실패시 failure 이벤트가 발생한다. failure 이벤트는 rui_config.js에 기본 정의되어 있으며 defaultFailureHandler 값에 따라서 failureHandler가 기본 수행된다.

```
var dataSetManager = new Rui.data.LDataSetManager();
dataSetManager.on('success', function(e) {
    alert('성공적으로 처리되었습니다.');
```

```
});
dataSetManager.updateDataSet({
    dataSets: [ dataSet ],
    url: './saveData.rui'
});
```

6.3.5.2 여러 개의 DataSet 조회

여러 개의 DataSet을 한번에 서버에서 조회할 경우에는 loadDataSet 메소드를 호출 하면 된다. 호출시 dataSets Config에 배열값으로 DataSet들을 순차적으로 정의하면 서버에서 받은 데이터 값의 metaData에 정의된 dataset의 id와 자동으로 매핑하며 데이터를 로딩한다.

```
dataSetManager.loadDataSet({
    dataSets:[ dataSet1, dataSet2 ],
    url : 'load.rui'
});
```

6.3.6 Bind 사용

Bind는 DataSet과 FORM 객체들을 자동 연결하여 FORM 객체에 값을 넣어주며, FORM 객체의 값이 변경되면 자동으로 DataSet에 그 값을 반영한다.

```
var bind = new Rui.data.LBind({
    groupId: 'frm',
    dataSet: dataSet,
```

```

        bindInfo: [
            { id: 'col1', ctrlId: 'col1', value: 'value' },
            { id: 'col2', ctrlId: 'col2', value: 'value' }
        ]
    });

```

6.3.6.1 Bind 의 Config

Bind 를 사용할 때의 생성인자는 다음과 같다.

생성인자(Config)	기능
groupId	FORM 객체들이 포함되어 있는 HTML Element 의 id
dataset	FORM 객체들과 동기화 할 DataSet
bindInfo	동기화할 규칙을 정의한다. 규칙은 id: DataSet 의 field 의 id 값 ctrlId: FORM 객체의 id 또는 name value: 동기화할 값의 키 (value or text)

6.4 Validation

Form object 나 Grid object, dataSet 의 Data 유효성을 검사한다. 기본적으로 유효성은 RUI.validate.LValidatorManager 에 정의하고 ValidatorManager 가 Form object, Grid Object, dataSet 을 대상으로 유효성 체크를 하면 각 객체의 관점에서 invalid 처리를 함으로써 ValidatorManager 에 정의하는 방식만 동일하면 유효성 체크에 대해서 반복적인 작업을 할 필요가 없다. 그리고 Validator 를 작성하는 규칙을 이해한 후 프로젝트용 LValidator 를 작성하면 모든 페이지에서 사용이 가능하다.

6.4.1 ValidatorManager 선언 및 사용

```

var validatorManager = new RUI.validate.LValidatorManager({
    validators:[
        { id: 'col2', validExp:'Col2:true:length=4'},
        { id: 'col3', validExp:'Col3:true:minLength=6'},
        { id: 'col4', validExp:'Col4:true:byteLength=4'},
        { id: 'col5', validExp:'Col5:true:minByteLength=8'},
        { id: 'col6', validExp:'Col6:true:number'},
        { id: 'col7', validExp:'Col7:true:number=3.2'},
    ]
});

```



```

    { id: 'col8', validExp:'Col8:true:minNumber=100'},
    { id: 'col9', validExp:'Col9:true:maxNumber=100'},
    { id: 'col10', validExp:'Col10:true:inNumber=90~100'},
    { id: 'col11', validExp:'Col11:true:date=YYYYMMDD'},
    { id: 'col12', validExp:'Col12:true:minDate=2008/11/11(YYYY/MM/DD)'},
    { id: 'col13', validExp:'Col13:true:maxDate=20081111'},
    { id: 'col14', validExp:'Col14:true:ssn'},
    { id: 'col15', validExp:'Col15:true:csn'},
    { id: 'col16', validExp:'Col16:true:filter=%;<;WWh;WW;;haha'},
    { id: 'col17', validExp:'Col17:true:allow=WWa;WWn'},
    { id: 'col18', validExp:'Col18:true:email'},
    { id: 'temp1' validExp:'Temp1:true', fn:function(val, row) {
// alert 및 기타 focus된 객체가 blur 이벤트가 발생하는 모든 소스는 사용하면 안됨
        if(true) {
            this.message = '유효하지 않습니다.';
            return false;
        }
        return true;
    }}
    ]
});

```

```

// object 값은 { key: value }의 json 구조
if(validatorManager.validate(object) == false) {
    // invalid 처리
}

```

```

if(validatorManager.validateEl(objectEl) == false) {
    // invalid 처리
    // 기본적으로 el에 invalid 속성이 적용
}

```

```

if(validatorManager.validateDataSet(dataSet) == false) {
    // invalid 처리
    // dataSet의 validate를 체크할 경우 dataset 자체에 있는 invalid/valid 이벤트가
    호출되고 해당 dataSet과 연동된 객체 (grid/bind등의 다른 객체들에게 통보되어 자동으로
    invalid/valid 처리가 진행된다.
}

```

```
// invalid시 메시지 제어
if(validatorManager.validateGroup('dom object id') == false) {
    alert(Rui.getMessageManager().get('$.base.msg052') + 'WrWn' +
        validatorManager.getMessageList().join('WrWn') );
}

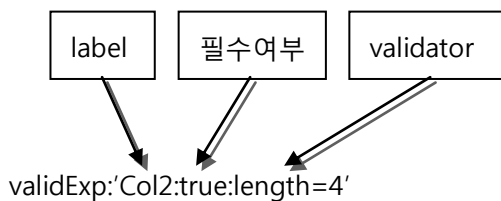
// invalid시 invalid 객체 제어
if(validatorManager.validateGroup('dom object id') == false) {
    var arrayObject = validatorManager.getInvalidList(); // invalid list 배열을 받아서 객체
    제어
}
```

6.4.2 Validator 구조

Rui.validate.LValidator는 validate 메소드와 message 속성으로 되어 있다.

모든 Validator는 Rui.validate.LValidator를 상속받으며 validate 메소드를 override하고 invalid시 출력할 message를 가지고 있다.

validExp는 ':'값을 기준으로 3가지 항목으로 구성된다.



Validator 선택

'L' + length + 'Validator' <= LLengthValidator가 적용

Validator선택은 복수로 지정할 수 있다. 복수의 정의는 '&'로 구분한다.

validExp:Col2:true:number&length=4' <= LNumberValidator와 LLengthValidator가 적용된다.

6.4.3 Validator 의 종류

LLengthValidator	전체 글자의 길이를 체크하는 validator { id: 'col2', validExp:'Col2:true:length=4'}
LDateValidator	날짜의 유효성을 체크하는 validator { id: 'col11', validExp:'Col11:true:date=YYYYMMDD'}
LFormatValidator	format 에 맞는 값인지를 체크하는 validator { id: 'col17', validExp:'Col17:true:format=abc'}

LNumberValidator	숫자 여부를 판단하는 validator { id: 'col6', validExp:'Col6:true:number'}
LRequiredValidator	필수 여부를 체크하는 validator { id: 'col6', validExp:'Col6:true' }
LByteLengthValidator	byte 로 길이를 체크하는 validator { id: 'col4', validExp:'Col4:true:byteLength=4'}
LFilterValidator	지정된 문자가 들어있을 경우 유효하지 않은 것으로 판단한다. {id: 'col16', validExp: 'Col16:true:filter=%;<;WWh;WW;;haha'}
LInNumberValidator	범위안에 숫자가 존재하는지 체크하는 validator { id: 'col10', validExp:'Col10:true:inNumber=90~100'}
LMinByteLengthValidator	byte 로 최소 길이를 체크하는 validator { id: 'col5', validExp:'Col5:true:minByteLength=8'}
LMinDateValidator	최소 입력 날짜인지 확인하는 validator { id: 'col12', validExp:'Col12:true:minDate=2008/11/11(YYYY/MM/DD)'}
LMinLengthValidator	최소 길이를 확인하는 validator { id: 'col3', validExp:'Col3:true:minLength=6'}
LMinNumberValidator	최소 숫자를 확인하는 validator { id: 'col8', validExp:'Col8:true:minNumber=100'}
LMaxByteLengthValidator	byte 로 최대 길이를 체크하는 validator { id: 'col5', validExp:'Col5:true:maxByteLength=8'}
LMaxDateValidator	기준 날짜를 초과하는지 체크하는 validator { id: 'col13', validExp:'Col13:true:maxDate=20081111'}
LMaxLengthValidator	최대 길이를 초과하는지 체크하는 validator { id: 'col3', validExp:'Col3:true:maxLength=6'}
LMaxNumberValidator	최대 숫자를 초과하는지 체크하는 validator { id: 'col9', validExp:'Col9:true:maxNumber=100'}
LSsnValidator	주민번호인지 체크하는 validator { id: 'col14', validExp:'Col14:true:ssn'}
LCsnValidator	사업자 번호인지 체크하는 validator { id: 'col15', validExp:'Col15:true:csn'}
LEmailValidator	이메일인지 체크하는 validator { id: 'col18', validExp:'Col18:true:email'}
LAllowValidator	지정된 문자가 들어있을 경우 유효한것으로 판단한다. { id: 'col17', validExp:'Col17:true:allow=WWa;WWn'}
LGroupRequireValidator	Checkbox 나 Radiobox 의 필수 여부를 체크하는 validator { id: 'col18', validExp:'Col18:true:groupName=col8'}

6.4.4 프로젝트용 공통 Custom Validator 만들기

프로젝트용 validator 는 Rui.validate.LValidator 를 상속받아 구현한다. 해당 객체의 이름은 L 로 시작하고 Validator 이름으로 끝나야 한다.

L + '유효성 명' + Validator

Override 로 validate 메소드를 구현하고 내부적인 속성으로 invalid 시 출력할 message 를 가지고 있어야 한다. 이때 생성자에서 superclass 의 생성자를 반드시 호출해야 한다.

```
// 공통 js에 아래의 내용을 추가하여 작성
LCustomValidator = function(id, oConfig) {
    LCustomValidator.superclass.constructor.call(this, id, oConfig); ← superclass 반드시
    호출
    this.config = oConfig || {};
    this.message = '유효하지 않습니다.' ← invalid시 출력할 메시지
}

Rui.extend(LCustomValidator, Rui.validate.LValidator, { ← LValidator를 상속받아 구현
    validate : function(value) {
        // 값이 test인 문자는 입력할 수 없습니다.
        return value != "test"; ← 리턴값이 false에 invalid
    }
});
```

위와 같이 만들어진 프로젝트용 validator 는 다음과 같이 ValidatorManager 에서 사용할 수 있다.

```
// LValidatorManager에서의 LCustomValidator 사용 예
var validatorManager = new Rui.validate.LValidatorManager({
    validators:[
        { id: 'col2', validExp:'Col2:true:custom'} ← Custom validator 사용
    ]
});
```

6.5 Grid 컴포넌트

Grid 는 ActiveX 의 Grid 와 같은 기능을 웹표준으로 구현한 UI 컴포넌트이다. Grid 는 기본적으로 GridPanel, GridView, ColumnModel, RowModel, SelectionModel 로 구성되면 Pager, TotalSummary, TreeGrid, Expandable Grid 등 Plugin 을 통해서 더 다양하게 출력이 가능하다.

6.5.1 Buffered Grid

RichUI 의 Grid 는 Buffered 개념을 적용하여 DataSet 은 조회하고자 하는 데이터를 서버에서 모두 받아와 브라우저에서 가지고 있지만 출력은 사용자가 보고 있는 row 기존에서만 HTML Element 를 생성한다. 또한 사용자가 Grid 를 스크롤 할 경우 보이는 영역의 row 가 바뀌게 되면 다시 랜더링 하여 메모리 사용량도 줄이고 대량건의 데이터도 처리가 가능하게 구현되었다

6.5.2 Grid 의 생성

Grid 는 GridPanel 에 DataSet, ColumnModel, width, height 를 Config 로 생성한 후 생성할 위치의 HTML Element 의 id 를 지정하여 render 메소드를 호출하면 출력된다. Grid 출력된 후에는 dataSet 이 서버에서 데이터를 load 하면 Grid 는 ColumnModel 의 규칙에 따라서 데이터를 출력한다.

```
var dataSet = new Rui.data.LJsonDataSet({
    id: 'dataSet',
    fields: [
        { id: 'col1' },
        { id: 'col2' },
        { id: 'col3' }
    ]
});

var columnModel = new Rui.ui.grid.LColumnModel({
    columns: [
        new Rui.ui.grid.LSelectionColumn(), // 기본 LColumn 객체
        new Rui.ui.grid.LStateColumn(),
        new Rui.ui.grid.LNumberColumn(),
        { field: 'col1', label: 'Col1' }, // LColumn 객체로 변환
        { field: 'col3', label: 'Col2' },
        { field: 'col2', label: 'Col3' }
    ]
});
```

```
});

var gridPanel = new Rui.ui.grid.LGridPanel({
    columnModel: columnModel,
    dataSet: dataSet,
    width: 600,
    height: 300
});

gridPanel.render('defaultGrid');
```

6.5.3 ColumnModel

ColumnModel 은 DataSet 이 가지고 있는 데이터를 Grid 에 출력할 순서, 유형, 방식 등을 정의한다.

6.5.4 ColumnModel 의 Config

Config	기능
defaultSortable	전체 Column 의 정렬 가능 여부를 설정한다.
freezeColumnId	Grid 의 틀고정할 Column 의 id 를 지정한다.
autoWidth	Column 들의 width 를 자동 계산하여 Grid width 에 맞춘다.
groupMerge	셀 병합을 할 때 앞 데이터의 기준에 따라 병합할지 여부를 결정한다.

6.5.5 Column 의 Config

Config	기능
id	Column 의 고유 아이디
Field	DataSet 의 출력할 id 값을 설정한다.
Label	Grid 의 Column 의 header 에 출력할 문자열을 설정한다.
width	Column 의 width 값을 설정한다.
align	Column 의 좌측(left), 중앙(center), 우측(right) 정렬 방식을 설정한다.
editor	Grid 의 편집기를 설정한다.
vMerge	데이터셋의 값을 비교하여 세로 병합 여부를 설정한다.

hMerge	데이터셋의 값을 비교하여 가로 병합 여부를 설정한다.
sortable	Column 의 정렬 가능 여부를 설정한다.
renderer	DataSet 의 데이터를 출력시 출력 방법을 설정한다.
summary	Grid 에 소계를 설정한다.

6.5.6 Column 속성의 renderer

renderer 는 Column 의 디자인이나 값을 변경할 수 있는 function 을 지정하여 기본 데이터의 출력 방법을 변경한다. value, params, record, row, col 로 순차적으로 function 의 인수로 호출된다. 단 renderer 안에서는 record 의 값 변경이 발생해서는 안된다. Renderer 는 자주 사용할 수 있는 기능은 RichUI 에서 기본으로 제공한다. Rui.util.LRenderer 를 이용하여 별도의 function 을 생성하지 않고 사용할 수 있다.

```
var columnModel = new Rui.ui.grid.LColumnModel({
    columns: [
    ...
    { field: 'col2', renderer: function(value, params, record, row, col) {
        return value.substring(0, 6) + '-' + value.substring(6);    // 출력할 문자열 변경
    } },
    { field: 'col3', renderer: Rui.util.LRenderer.dateRenderer() }, // 기본 LRenderer 사용
    ],
});
```

6.5.7 rederer 인수 설명

인수	기능
value	DataSet 의 실제 출력할 값
params	출력시 css 제어나 editable 여부를 설정하는 Json 객체
record	DataSet 의 현재 출력할 Record 객체
row	DataSet 의 현재 출력할 row 위치
col	Column 객체의 index 위치

6.5.8 Grid 의 편집기

Grid 에서 편집을 하려면 Column 의 config 인 editor 에 Form Object 컴포넌트를 정의하여 편집기로 사용할 수 있다.

```

var columnModel = new Rui.ui.grid.LColumnModel({
  columns: [
    { field: 'col1', editor: new Rui.ui.form.LTextBox() },
    { field: 'code', editor: col1Combo },
    { field: 'col2', editor: new Rui.ui.form.LRadioGroup({
      items:[
        { label : 'R1', value: 'R1' },
        { label : 'R2', value: 'R2', checked: true }
      ]
    }) },
    { field: 'date1' , editor: new Rui.ui.form.LDateBox() }
  ]
});

```

6.5.9 Grid 의 편집 가능/불가능 설정

Grid 에서 편집 가능 및 불가능 설정은 초기 렌더링시 처리하는 renderer 처리하는 방식과 렌더링 후 cellConfig 처리 방식으로 구분된다. 단순히 출력시에만 판단해도 될 경우에는 renderer 의 params config 값으로 처리한다.

```

var columnModel = new Rui.ui.grid.LColumnModel({
  columns: [
    { field: 'col2', renderer: function(value, p, record, row, col) {
      if (value == '7602111111113') {
        // 셀의 편집여부를 결정한다.
        p.editable = false;
      }
      return value;
    } },
  ]
});

.....
gridPanel.setCellConfig(0, 'col2', 'editable', false); // 로직에 따라 편집 여부 결정

```


6.5.10 소계

소계는 Column 의 config 속성중 summary 을 적용하면 된다. Summary config 의 규칙은 ids 에 summary 규칙이 적용될 기준이 되는 Column 들의 id 를 배열로 적용한다.

```
var columnModel = new Rui.ui.grid.LColumnModel({
  columns: [
    { field: 'company' },
    { field: 'dept' },
    { field: 'monthSum', summary: { ids: [ 'company', 'dept' ] } },
    { field: 'balance', summary: { ids: [ 'company', 'dept' ] } }
  ]
});
```

6.5.11 합계

합계는 Plugin 으로 제공되는 기능으로 <ru root>/plugins/ui/grid/LTotalSummary.js 와 LTotalSummary.css 를 HTML head 에 포함해야 한다.

```
<script type="text/javascript" src="/ru/plugins/ui/grid/LTotalSummary.js"> </script>
<link rel="stylesheet" type="text/css" href="/ru/plugins/ui/grid/LTotalSummary.css"/>
```

6.5.12 합계의 구현

합계는 LTotalSummary 를 생성하여 Grid 의 viewConfig 의 plugins config 값으로 Grid 를 생성한다. DataSet 이 조회되거나 변경될 경우 renderTotalCell 이벤트가 발생하며, 이벤트 인수의 e.value 값에 출력할 데이터를 적용하면 된다.

```
var summary = new Rui.ui.grid.LTotalSummary();
summary.on('renderTotalCell', function(e){
  if(e.col == 2) {
    e.value = 'Total';
  } else {
    if (e.colId == 'monthSum') {
      e.value = Rui.util.LFormat.moneyFormat(dataSet.sum(e.colId));
    }
  }
});
```

```
var grid = new Rui.ui.grid.LGridPanel({
    columnModel: columnModel,
    dataSet: dataSet,
    viewConfig: {
        plugins: [ summary ]
    },
    width: 800,
    height: 400
});
```

6.5.13 Tree Grid

Tree Grid 는 사용자에게 Tree 의 형식과 Grid 의 형식을 혼합한 형태를 제공한다. 또한 Tree 의 경우 데이터가 많을 경우 성능이 심하게 저하 되는 반면 TreeGrid 는 Buffered 방식으로 구현되어 성능에 영향을 덜 받는다. 다만 데이터 순서 변경에 따른 출력 오류를 모두 개발자가 처리 해야 하므로 개발 복잡도가 증가된다. Tree Grid 는 Plugin 으로 제공되며, 아래의 스크립트들을 HTML head 에 포함해야 한다.

```
<script type="text/javascript" src="/rui/plugins/ui/grid/LTreeGridSelectionModel.js"> </script>
<script type="text/javascript" src="/rui/plugins/ui/grid/LTreeGridView.js"> </script>
<link rel="stylesheet" type="text/css" href="/rui/plugins/ui/grid/LTreeGridView.css"/>
```

6.5.14 Tree Grid 의 구현

Tree Grid 는 LTreeGridView 와 LTreeGridSelectionModel 을 생성하여 GridPanel 의 config 로 정의한다. 출력 방식은 부모/자식 관계가 아닌 Record 의 depth 만으로 계층을 구분하며, depth 값이 0 이거나 없으면 최상위 계층으로 인식한다.

```
var dataSet = new Rui.data.LJsonDataSet({
    id: 'dataSet',
    fields: [
        { id: 'depth', type: 'number', defaultValue: 0 },    // depth 필드는 필수
        { id: 'treeNodeId' },
        .....
    ]
});
```

```

var columnModel = new Rui.ui.grid.LColumnModel({
    columns: [
        { field: 'id' },
        .....
    ]
});

var treeGridView = new Rui.ui.grid.LTreeGridView({
    columnModel: columnModel,
    dataSet: dataSet,
    fields: {
        depthId: 'depth'
    },
    treeColumnId: 'treeNodeId'    // 아이콘을 출력할 Column의 아이디
});

var grid = new Rui.ui.grid.LGridPanel({
    columnModel: columnModel,
    dataSet: dataSet,
    view: treeGridView,
    selectionModel: new Rui.ui.grid.LTreeGridSelectionModel(),
    width: 800,
    height: 250
});

```

6.5.15 TreeGrid 의 필수 config

생성인자(Config)	기능
columnModel	ColumnModel 객체
dataset	DataSet 객체
fields	내부에서 구분할 필드들의 배열. fields 의 필수 config 는 depthId 를 지정해야 한다.
treeColumnId	TreeGrid 에서 사용자에게 펼침/닫힘 아이콘을 제공할 Column 의 아이디

6.5.16 Rui.ui.LDialog

LDialog 의 상속구조는 LUiComponent>LPanel>LDialog 이다.

Dialog 레이아웃 구성은 내부적으로 header, body, footer 등 기본 구성 영역을 가지고 있으며, 주로 내용은 body 영역에 표현된다. 기타 header 및 footer 는 옵션에 따라 사용할 수 있다.

6.5.17 Dialog(대화상자)의 구현

우선 다이얼로그를 생성하기 위한 마크업을 추가한다. 헤더(hd)에는 대화상자의 타이틀을 표시하기 위해 기술하며 풋터(ft)에는 대화상자의 맨아래 부분에 내용을 표시할 경우 사용된다.

예) Dialog 마크업생성

```
<div id="dialog1">
<div class="hd">Please enter your information</div>
    <div class="bd"></div>
    <div class="ft"></div>
</div>
```

이제 Dialog 를 생성하는 부분을 알아보자

대화상자를 생성하는 방법은 대부분 동일하나 데이터를 처리하는 방식에는 몇가지 방식이 있다.

이 예제에서는 서버에 값을 전송하고 결과를 기다리지 않고 비동기방식으로 서버에서 값을 보내주면 처리하도록 구현되어있다.

대화상자 생성인자중 postmethod: 'async'라는 속성이 있는데 이것이 비동기로 서버로 값을 전송하고 결과를 수신할 경우 사용하기 위한 속성값이다.

대화상자 하단에 표시할 버튼들의 속성을 보면 핸들러를 정의해서 'submit' 버튼을 눌렀을 경우 별도 로직을 수행하기 위해 handleSubmit 이라는 함수를 따로 만들어 주었고 조건을 만족하면 최종적으로 submit 함수를 호출하여 서버로 전송하도록 하였다.

여기서 서버로 전송했지만 서버에서 처리완료하고 난뒤 그 결과를 클라이언트측에서 나중에 결과값을 받는 부분에 대한 처리는 callback 이라는 속성에 성공했을 경우 handlesuccss 에서 로직을 수행하고 실패했을 경우 handleFailure 를 수행하도록 하였다.

예) Dialog 생성

```
// Submit 버튼 클릭시 수행할 함수
var handleSubmit = function() {
    if(validatorManager.validateGroup('dialog1') != true) return;
    this.submit(true);
};
// Cancel 버튼 클릭시 수행할 함수
var handleCancel = function() {
```

```

        this.cancel(true);
    };

    //성공시 핸들러
    var handleSuccess = function(o) {
        var response = o.responseText;
        response = response.split('<')[0];
        document.getElementById('resp').innerHTML = response;
    };

    //실패시 핸들러
    var handleFailure = function(o) {
        alert('Submission failed: ' + o.status);
    };

    // Instantiate the Dialog
    var dialog1 = new Rui.ui.LDialog({
        applyTo: 'dialog1',
        width : 400,
        postmethod:'async',
        buttons :[
            { text:'Submit', handler:handleSubmit, isDefault:true }, // 전송버튼
            { text:'Cancel', handler:handleCancel } // 취소버튼
        ],
        callback : { // 성공여부에 따라 실행
            success: handleSuccess,
            failure: handleFailure
        }
    });

    var showBtn = new Rui.ui.LButton('showBtn');
    showBtn.on('click', function(){
        dialog1.clearInvalid(); // 폼객체의 invalid 된 객체를 모두 초기화
        dialog1.show(true); // 대화상자를 화면에 표시한다.
    });

```

위 예제에 사용된 생성인자에 대한 설명은 다음과 같다.

생성인자(Config)	기능
postmethod	'async'는 서버로 전송할 경우 비동기 방식으로 처리를 원할 경우

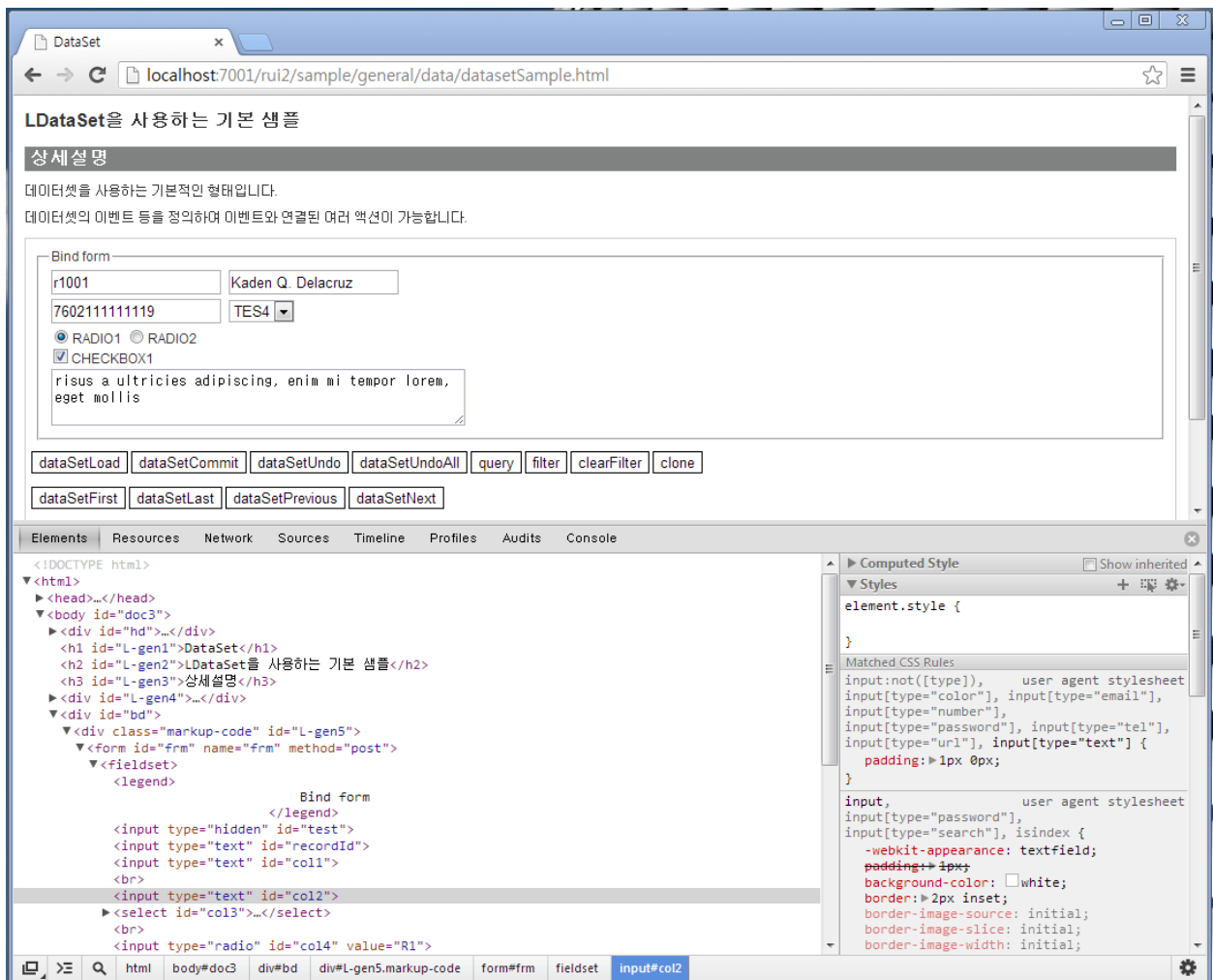
	설정한다.
buttons	대화상자에서 사용자와 응답을 결정하는 버튼을 지정해 주는 속성으로 버튼을 눌렀을 경우 Handler 를 통해서 별도 사용자 로직을 추가하여 사용한다.
callback	서버로부터 성공,실패시 결과에 따라 로직을 수행할 수 있도록 메소드를 추가할 수 있다.
width	대화상자의 가로 너비 지정

7. 디버깅

7.1 Chrome 개발자 도구

앞서 설치한 Chrome 브라우저의 개발자 도구가 지원하는 다양한 기능들을 살펴본다.

우선 Chrome 브라우저를 열고 "F12"키를 누르면 개발자 도구가 열린다.



위 그림처럼 브라우저 하단에 개발자 도구가 열린 모습을 볼 수 있다. 개발자 도구는 Chrome 의 메뉴를 이용해서 열 수도 있다. Chrome 우측 상단 메뉴의 "도구 - 개발자도구(Ctrl + Shift + I)"

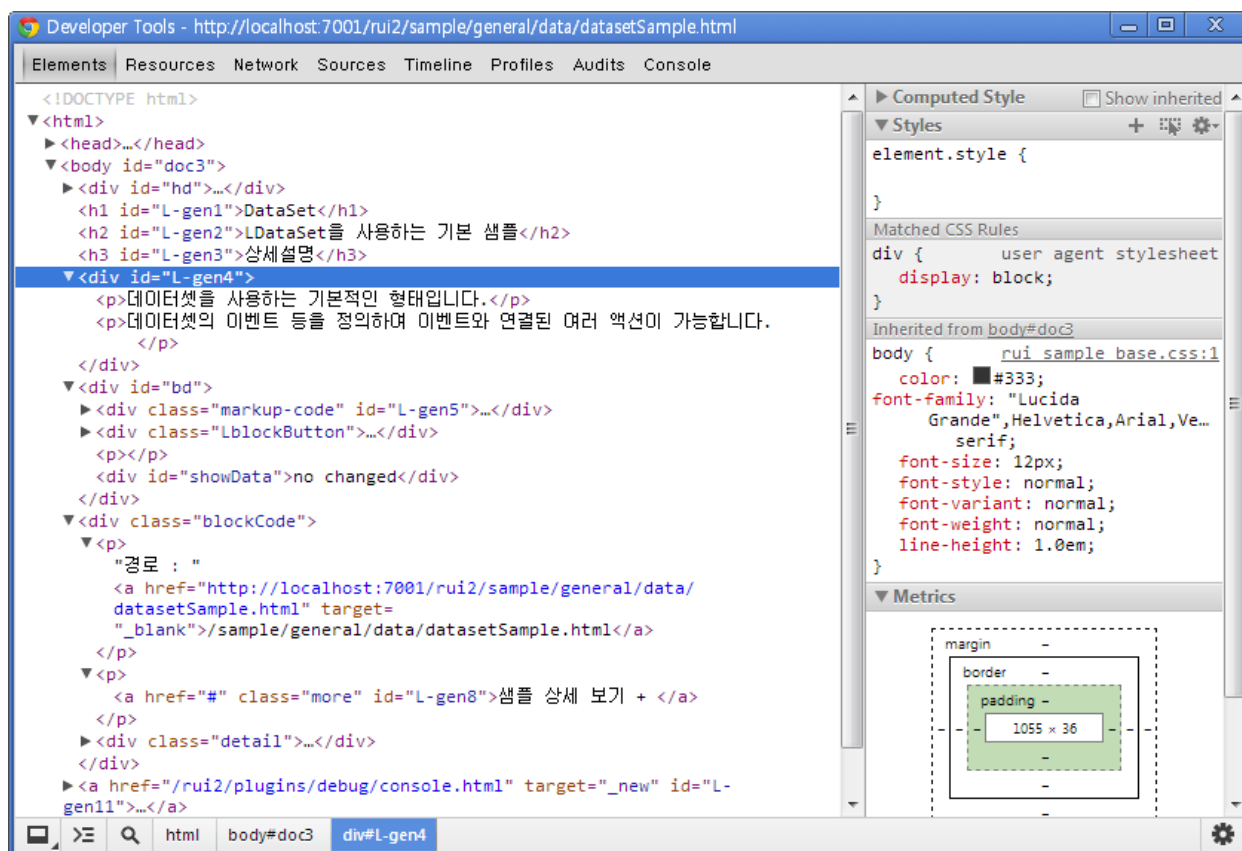
상단에 배치된 Elements 부터 Console 까지의 8 개의 탭들은 개발자도구가 제공하는 대표 기능들이다.

Elements	HTML Markup 및 DOM 구조를 표시한다. 특히 DOM 에 적용된 스타일, 속성, 크기 등을 확인 할 수 있다.
Resources	각종 자원들을 표시한다. 대표적으로 html, script, css, cookie, local storage, session storage 등이 가진 값을 이곳에서 확인 할 수 있다.

Network	서버로 주고받은 모든 자원의 정보를 표시한다. HTML Document, Stylesheet, Script, Image, Font, Ajax 등의 URL 및 해당 URL의 Type, Method, Status, Size, 로딩시간 등을 Timeline에 따라 확인할 수 있다.
Sources	Script와 Stylesheet의 소스코드를 표시한다. Breakpoint를 지정하여 소스코드를 디버깅할 수 있다.
Timeline	Timeline에 따라 브라우저에 발생한 이벤트를 표시하고, Memory 사용량, paint, style 갱신, layout 갱신, script parsing 등을 표시한다.
Profiles	Script의 CPU 점유, CSS Selector 사용, Heap Snapshot 등을 표시한다.
Audits	페이지를 분석하여 성능 향상을 위한 튜닝 방법 등을 제시한다.
Console	경고 및 오류등과 Script가 남긴 로그 등을 표시한다.

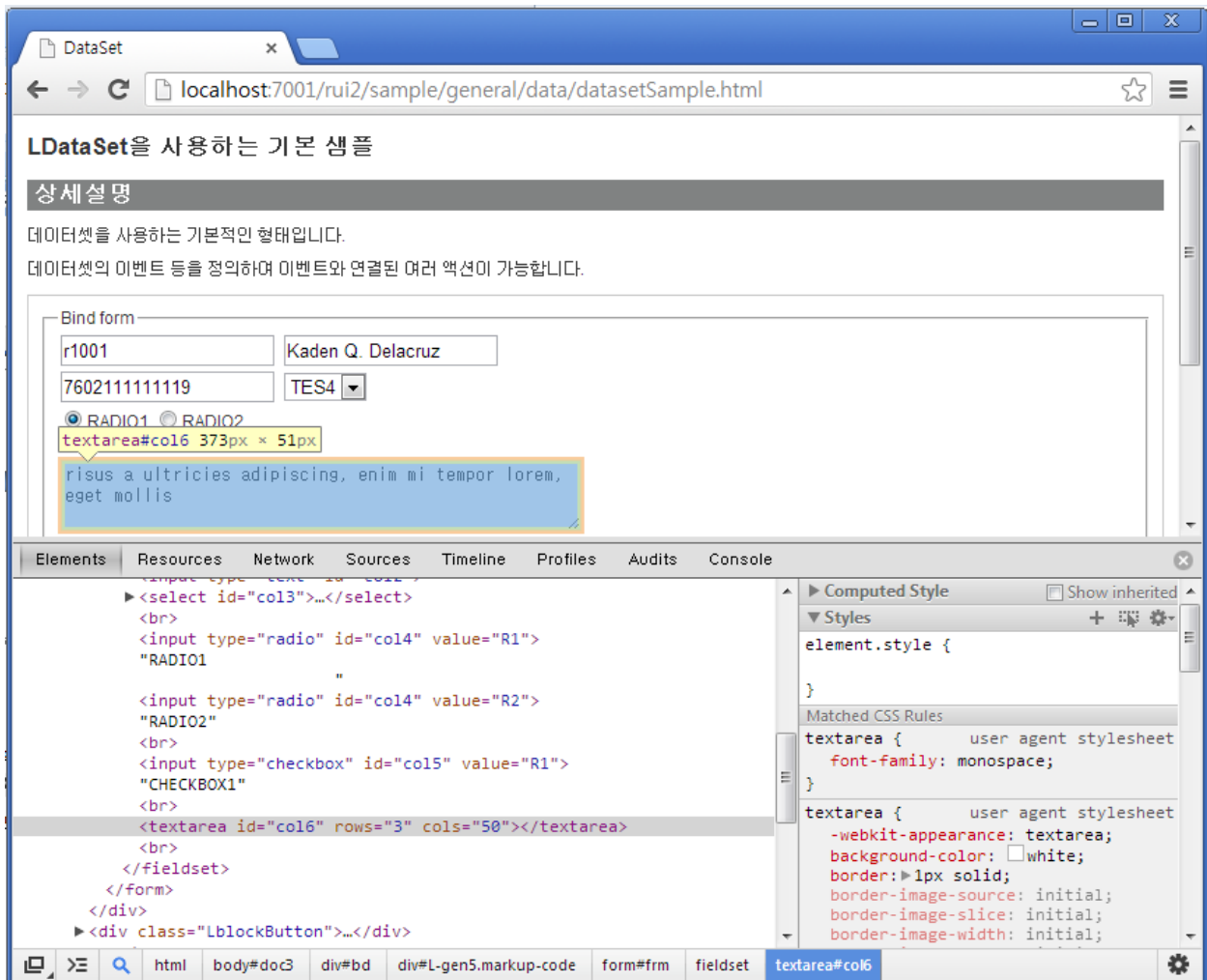
7.1.1 Elements 패널

Elements 패널은 개발자도구 중 가장 많이 활용되는 기능으로 현재 브라우저에 열려있는 HTML의 Markup 및 DOM 구조를 볼 수 있으며, 자세히 보고자 하는 DOM을 선택하면 DOM에 적용된 Style, Metrics, Properties 등의 값들을 볼 수 있다.



7.1.1.1 DOM 선택

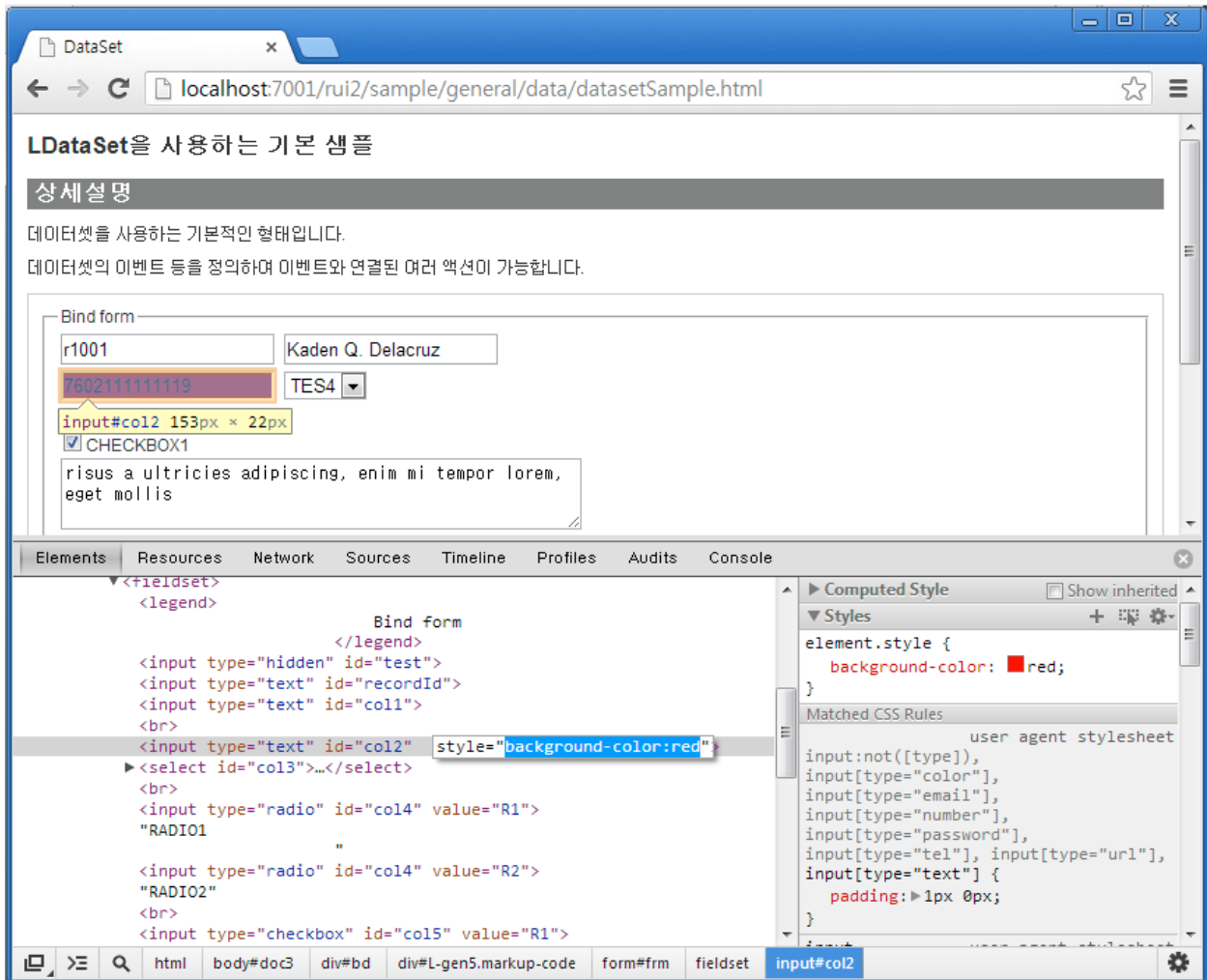
수 많은 DOM 들 중에서 원하는 DOM 을 찾기는 쉽지 않다. 이를 위해 개발자 도구는 검사하고자 하는 DOM 을 쉽게 찾을 수 있도록 Selector 를 제공한다. 하단의 🔍 버튼을 클릭하면 브라우저에서 원하는 요소 또는 영역을 마우스로 클릭함으로 DOM 이 자동 선택되는 기능을 제공한다.



그림과 같이 하단의 🔍 버튼을 클릭한 후 마우스를 원하는 요소 위로 올려놓고 클릭하면 선택된 요소의 DOM 을 정확하게 선택해준다. 또한 반대로 원하는 요소 위에서 마우스 우측 버튼을 클릭하면 나오는 컨텍스트 메뉴의 "요소검사"를 통해서도 DOM 을 선택할 수 있다.

7.1.1.2 DOM 속성 변경

선택된 DOM 의 속성은 동적으로 추가가 가능하다. 디자인 또는 디버그의 목적으로 임의의 속성을 DOM 에 추가할 수 있는 기능이 제공되며, 추가된 속성은 즉시 반영된다.



그림과 같이 Elements 탭 내 DOM의 속성 또는 빈 공간을 마우스로 더블 클릭하면 편집기가 열린다. 이 편집기에 속성명과 속성값을 입력하면 즉시 브라우저의 화면에 반영된다. 위 그림은 <INPUT> 태그에 style="background-color:red" 속성을 입력하여 빨간색 배경색을 추가한 모습이다.

7.1.1.3 DOM 분석

선택된 DOM을 분석하기 위해 우측 패널에서 속성 값들을 확인할 수 있다. 속성 및 값은 아래 표와 같다.

Computed Style	DOM에 적용된 전체 style이 나열되며, 우선순위에 따라 적용된 Style은 제외된다. 또한 해당 style의 소스 및 위치가 표시된다.
Styles	DOM에 적용될 수 있는 style들을 css selector 별로 나열되며, 해당 style이 적용되었는지 여부, 소스 및 위치 등이 표시되며, 적용된 style을 간편하게 제거 또는 새로운 style을 추가해볼 수도 있다.
Metrics	DOM의 metric을 표시한다. DOM의 크기, padding, border, margin 등의 수치를 표시하며 수치를 변경하면 즉시 반영된다.

Properties	DOM 의 객체 속성이 모두 표시된다. 계층 구조상 최하위 계층인 DOM 의 속성으로 시작하여 최 상위인 Object 까지 전체 속성 값들이 표시된다.
------------	--

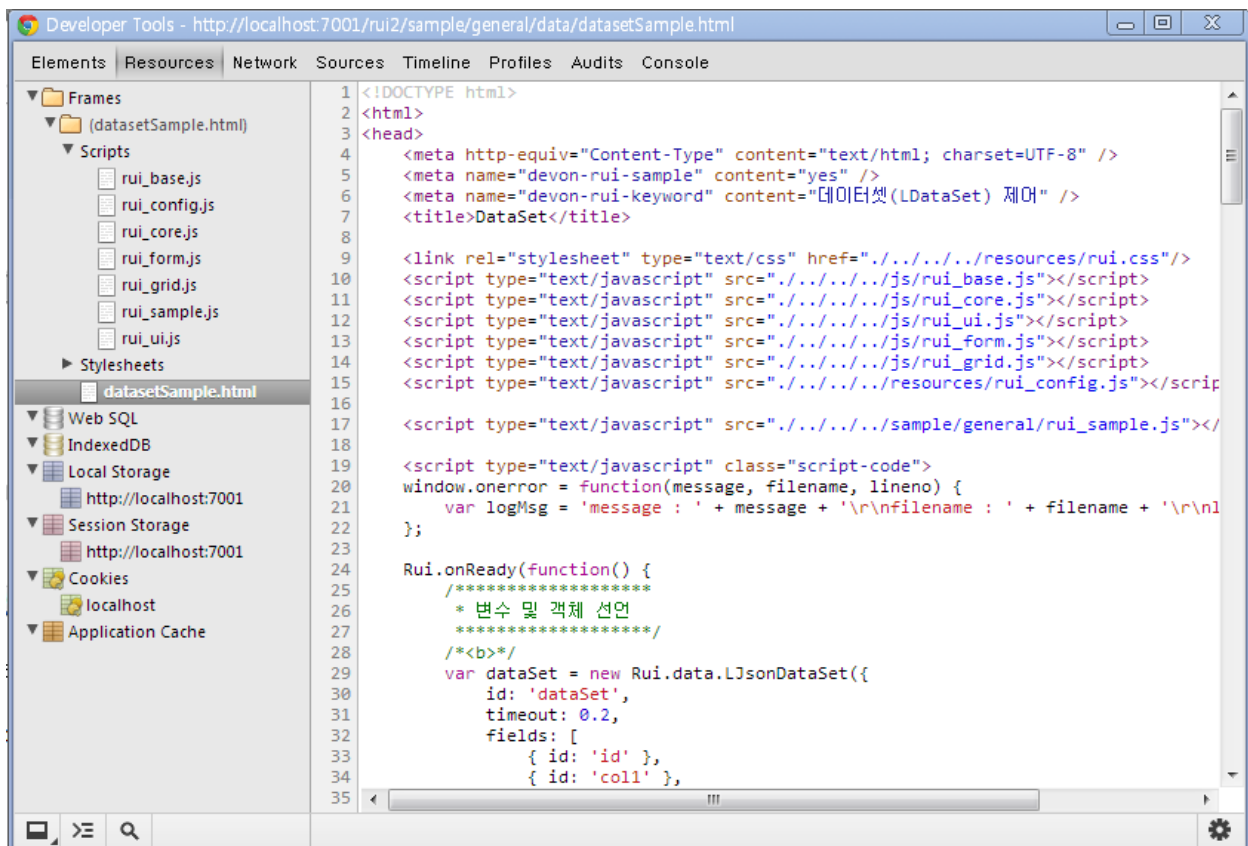
7.1.1.4 Elements 패널 참고

Elements 패널의 전체 기능은 아래 링크를 통해 확인해볼 수 있다.

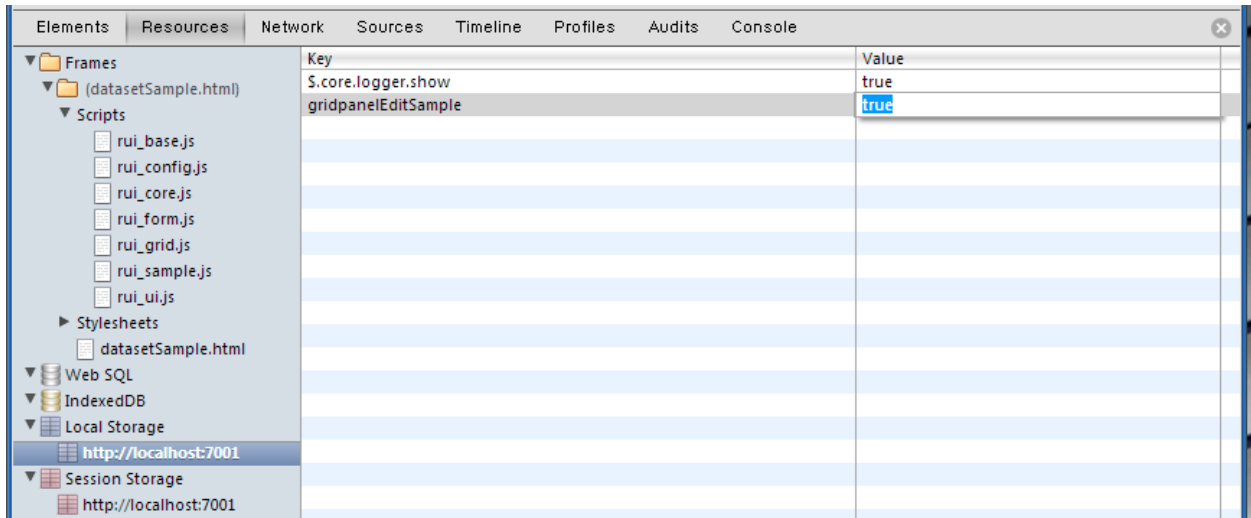
<https://developers.google.com/chrome-developer-tools/docs/elements?hl=ko-KR>

7.1.2 Resources 패널

Resources 패널은 현재 어플리케이션의 모든 자원들을 검사한다. Web SQL, IndexedDB, Local 및 Session Storage, Cookie, Application Cache 등을 검사할 수 있으며, HTML, script, image, font, style 등도 빠르게 찾아 검사할 수 있다.



Local Storage 의 값을 변경하여 어플리케이션을 테스트하고자 한다면 아래 그림과 같이 Local Storage 의 값을 선택하여 변경할 수 있다.



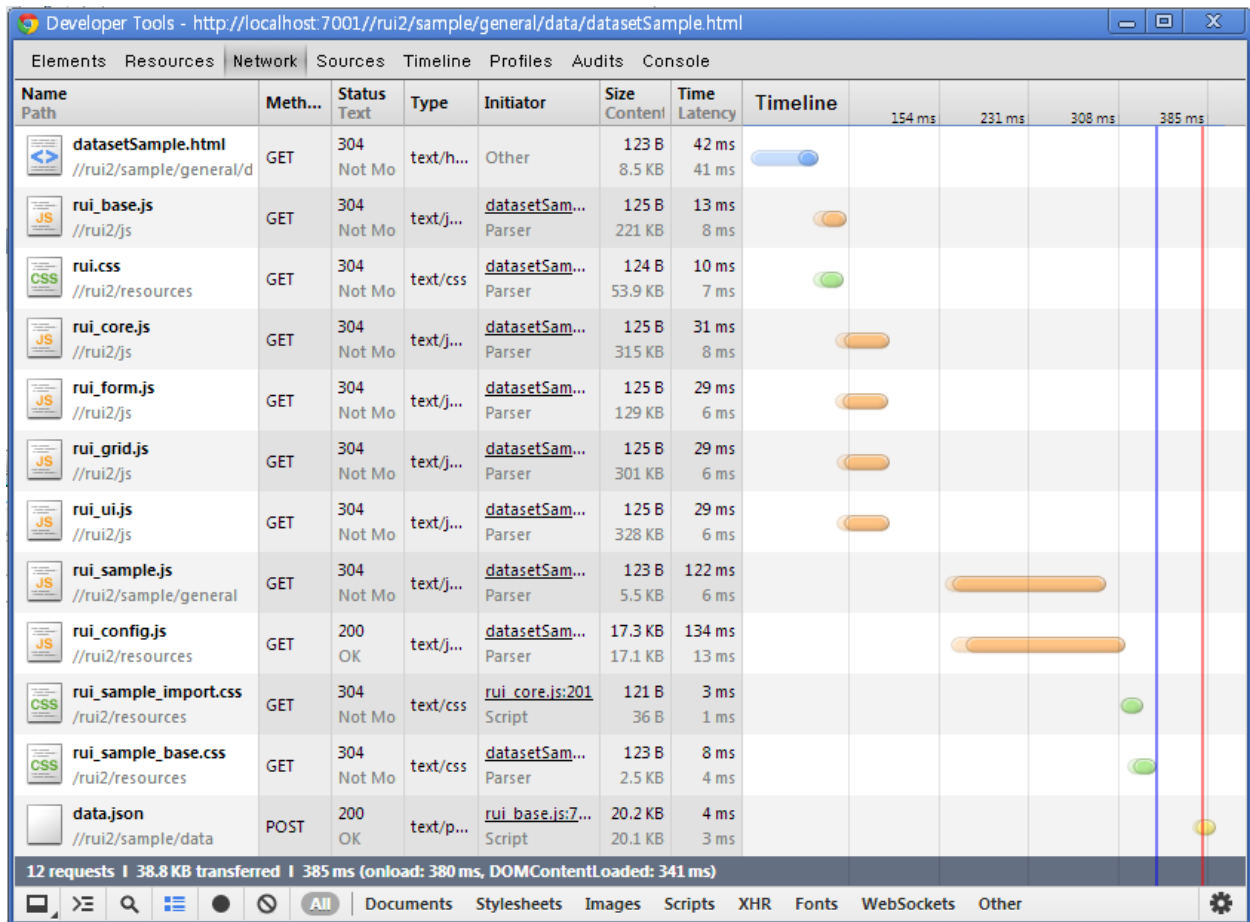
7.1.2.1 Resources 패널 참고

Resources 패널의 전체 기능은 아래 링크를 통해 확인해볼 수 있다.

<https://developers.google.com/chrome-developer-tools/docs/resource-panel?hl=ko-KR>

7.1.3 Network 패널












Network 패널은 네트워크 성능 분석 도구로 어플리케이션에 포함된 각각의 자원들을 네트워크를 통해 전달 받은 성능을 기록하고, 자원의 Header 및 Response 를 검사한다.



그림에 보이듯 각각의 자원들을 전달 받은 Status, Type, Size, Time 등이 표시되며, Timeline 을 통해 HTTP 요청의 시작부터 종료시점까지의 자원 별 로딩에 소요된 시간을 그래프로 볼 수 있다.



Timeline 의 막대 그래프에 마우스를 올려놓은 모습이다. 자원을 로딩하는데 소요된 시간을 Connecting, Sending, Waiting, Receiving 으로 각각 나눠 표시된다.

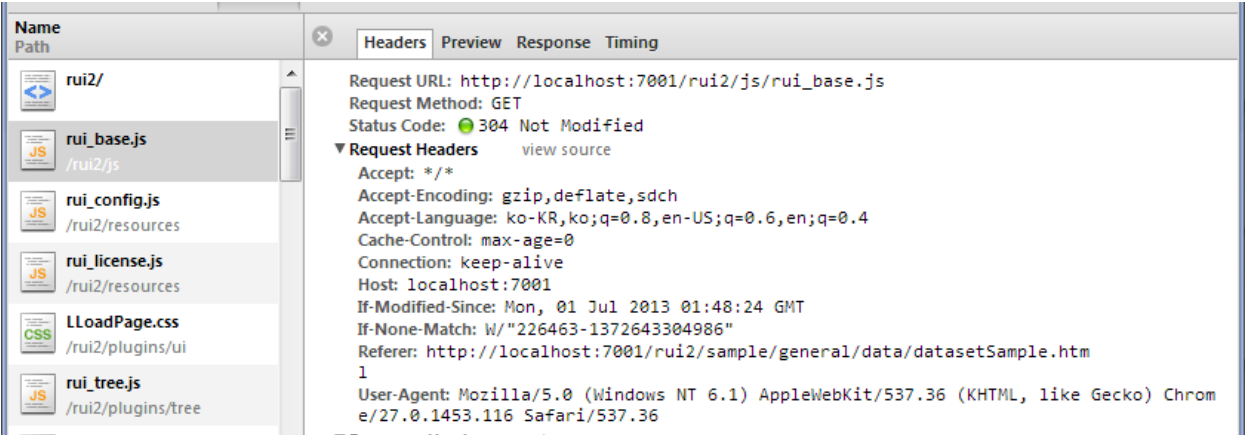
Name Path	Meth...	Status Text	Type	Initiator	Size Content	Time Latency	Timeline	124 ms	186 ms	248 ms	310 ms
 rui.css /rui2/resources	GET	304 Not Mo	text/css	datasetSam...	124 B 53.9 KB	10 ms 6 ms					
 rui_sample_import.css /rui2/resources	GET	304 Not Mo	text/css	rui_core.js:201	121 B 36 B	4 ms 4 ms					
 rui_sample_base.css /rui2/resources	GET	304 Not Mo	text/css	datasetSam...	123 B 2.5 KB	2 ms 2 ms					
3 / 12 requests 368 B / 21.6 KB transferred 310 ms (onload: 300 ms, DOMContentLoaded: 268 ms)											
     All Documents Stylesheets Images Scripts XHR Fonts WebSockets Other											

또한 하단의 Summary Bar 를 통해 로딩 된 자원의 개수, 크기, 시간 등을 집계하여 볼 수 있으며, Filter Button 들을 이용하여 자원을 종류별로 나누어서 확인할 수 있다.

7.1.3.1 Network 자원의 세부정보

Network 패널의 자원을 마우스로 클릭하여 선택하면 세부 정보를 검사 할 수 있다.

아래 그림들은 순서대로 로드 된 자원의 Header, Preview, Response, Timing 정보들을 표시한 것이다.



The screenshot shows the Network panel with the 'Headers' tab selected. The left sidebar lists resources including rui2/, rui_base.js, rui_config.js, rui_license.js, LLoadPage.css, rui_tree.js, and rui_core.js. The main area displays the headers for the selected resource, rui_base.js, with the following details:

- Request URL: http://localhost:7001/rui2/js/rui_base.js
- Request Method: GET
- Status Code: 304 Not Modified
- Request Headers:
 - Accept: */*
 - Accept-Encoding: gzip, deflate, sdch
 - Accept-Language: ko-KR, ko;q=0.8, en-US;q=0.6, en;q=0.4
 - Cache-Control: max-age=0
 - Connection: keep-alive
 - Host: localhost:7001
 - If-Modified-Since: Mon, 01 Jul 2013 01:48:24 GMT
 - If-None-Match: W/"226463-1372643304986"
 - Referer: http://localhost:7001/rui2/sample/general/data/datasetSample.htm
 - User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/27.0.1453.116 Safari/537.36

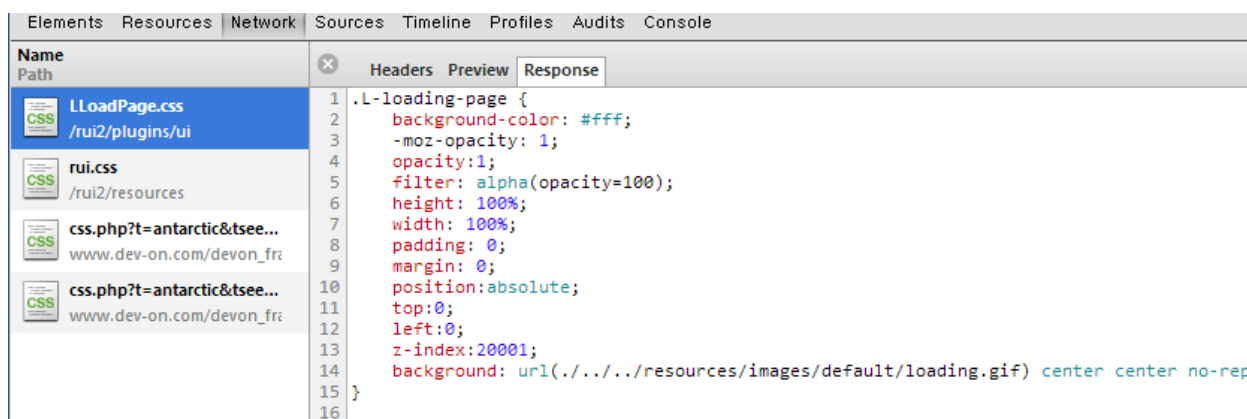
위 그림은 Request Header 정보이다.



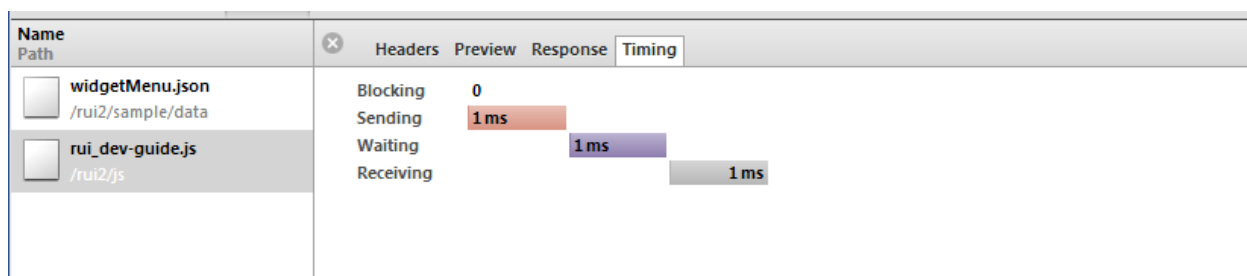
The screenshot shows the Network panel with the 'Preview' tab selected. The left sidebar lists resources including logo_RichUI_black.gif, topLogo.jpg, loading.gif, bg_devTab_on.gif, and bg_devTab_off.gif. The main area displays a preview of the selected resource, topLogo.jpg, with the following details:

- Dimensions: 162 x 60
- File size: 7.0 KB
- MIME type: image/jpeg
- URL: http://localhost:7001/rui2/resources/images/topLogo.jpg

위 그림은 이미지 파일 등을 미리 볼 수 있는 Preview 이다.



위 그림은 Response 이다. 전송 받은 파일의 전체 정보가 표시된다.



위 그림은 Network 패널의 Timeline 에 표시되던 Timing 정보와 동일하다.

7.1.3.2 Network 패널 참고

Network 패널의 전체 기능은 아래 링크를 통해 확인해볼 수 있다.


<https://developers.google.com/chrome-developer-tools/docs/network?hl=ko-KR>

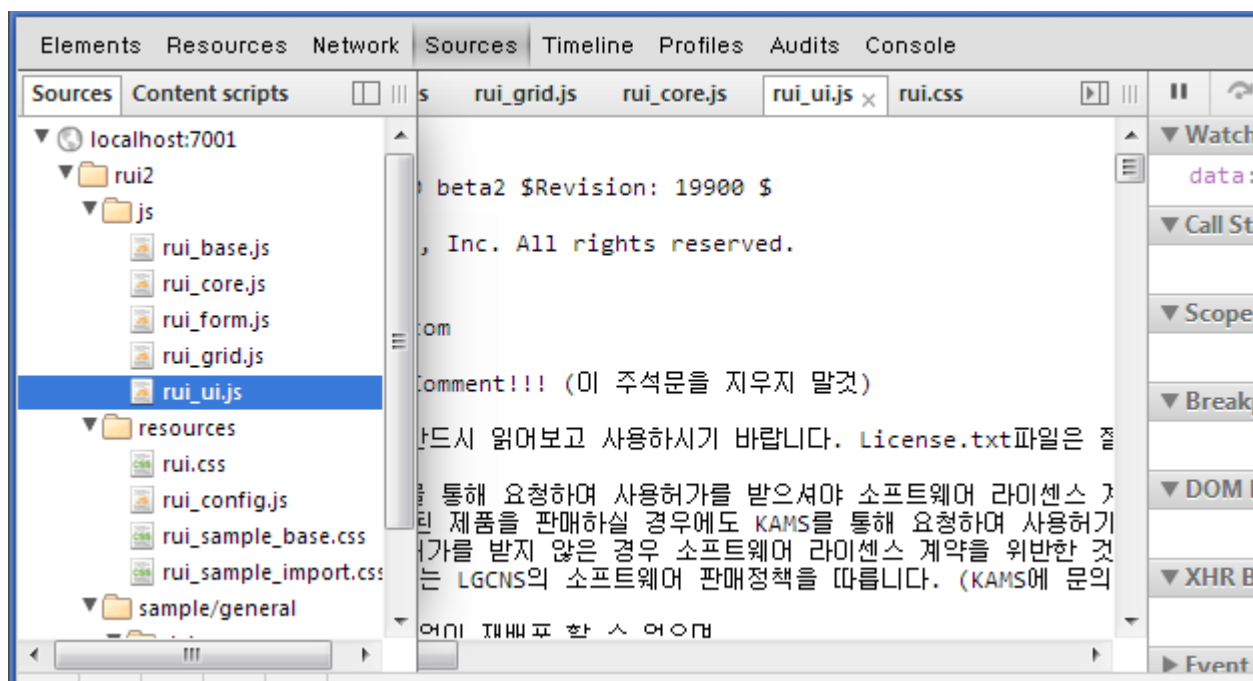
7.1.4 Sources 패널

Sources 패널은 어플리케이션의 소스를 검사하고 디버깅 할 수 있는 도구로 어플리케이션에 발생한 오류의 원인을 빠르게 찾을 수 있도록 강력한 도구들을 제공한다.



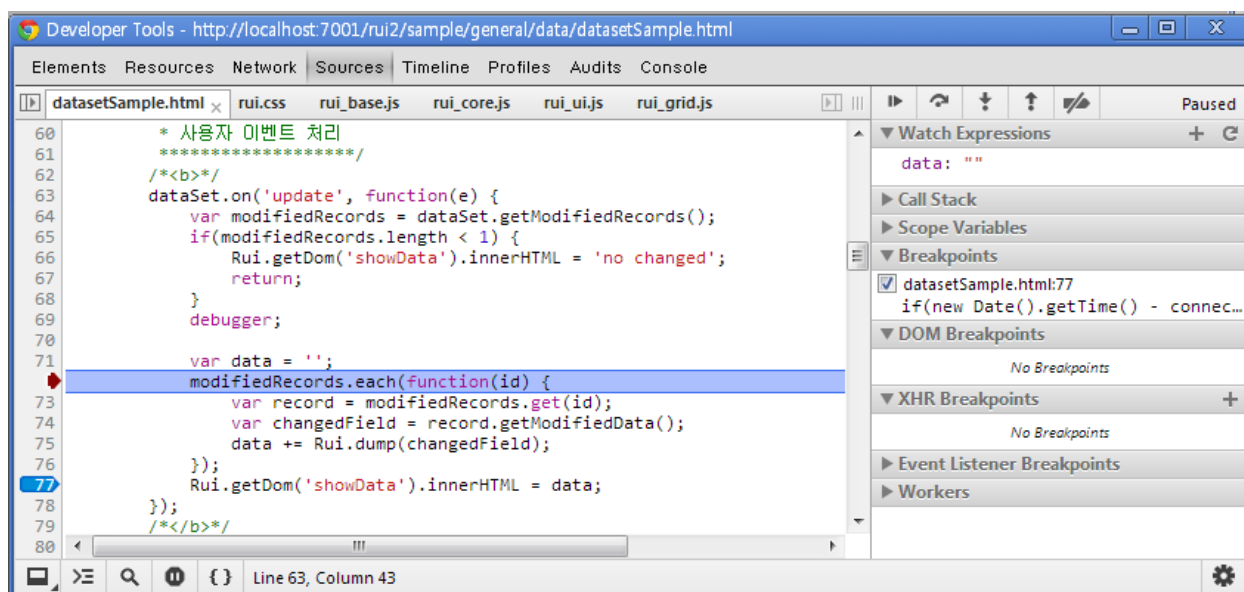
7.1.4.1 소스코드 검사

Sources 패널 좌측 상단의  버튼을 클릭하면 어플리케이션의 모든 코드 라이브러리들의 트리 목록이 펼쳐진다. 이 기능을 통해 필요한 파일을 검사할 수 있으며 Breakpoint 등을 지정하여 향후 디버깅이 가능하다.








7.1.4.2 Breakpoint 와 디버깅

Chrome 이 제공하는 개발자도구의 강력한 디버깅에 필요한 각종 기능과 방법을 소개한다.



위 그림은 어플리케이션을 디버깅하고 있는 모습이며 현재 datasetSample.html 파일의 72 번 라인에서 진행(Step)이 멈춰진 상태이다. 이렇게 진행을 멈추기 위해 Breakpoint 들을 지정 하는데 Breakpoint 는 그림과 같이 좌측 라인번호 지점을 클릭하여 지정할 수 있다. 또는 69 번 라인처럼 소스코드에 debugger 를 코딩 하여 지정할 수 도 있다.

진행이 멈춰진 후에는 우측 상단에 나열된  버튼들을 통해 단계적으로 진행할 수 있다.

	Resume script execution (F8) 스크립트의 남은 부분을 모두 진행한다. 단 다음 Breakpoint 를 만나면 멈춘다.
	Step over next function call. (F10) 다음 단계로 진행한다.
	Step into next function call (F11) 다음 단계로 진행한다. 단 메소드인 경우 메소드 안으로 들어간다.
	Step out of current function. (<Shift> + F11) 현재의 메소드를 모두 진행하고 나간다.
	Deactivate breakpoints. Toggle Breakpoint 를 모두 무시하고 진행한다.

7.1.4.3 디버깅을 위해 제공되는 정보

디버깅을 위해 진행이 멈춰진 상태에서는 변수의 값을 검사할 수 있으며, 진행중인 현재의 Stack Trace 를 확인 할 수 있다.

Watch Expressions	현재 scope 에 해당하는 범위에서 원하는 표현식을 추가하여 그 값을
-------------------	---

	검사할 수 있다.
Call Stack	현재 위치까지의 Stack Trace 가 역순으로 표시되며, 필요 시 위치를 이동한다.
Scope Variables	현재 scope 의 모든 값들이 표시되며, Local, Closure, Global(Window)등으로 나뉘어진다.
Breakpoints	지정된 Breakpoint 들의 목록이 표시되며, 해제 또는 비활성 할 수 있다.

7.1.4.4 Sources 패널 참고

Sources 패널로 디버깅 하는 방법의 부가 설명은 아래 링크를 통해 확인해볼 수 있다.

<https://developers.google.com/chrome-developer-tools/docs/javascript-debugging?hl=ko-KR>

7.1.5 Console 패널

Console 패널은 어플리케이션 오류 발생시 오류 내용과 위치를 출력하며, 어플리케이션에 포함된 개발 log 를 출력한다. XMLHttpRequest log 도 출력 가능하다. 그리고 필요에 따라서 표현식을 추가하여 검사할 수 있다.

7.1.5.1 Console 의 다양한 이용



위 그림의 Console 첫번째 라인은 javascript 파일을 찾지 못하였다는 오류가 발생한 것이 표시된 것으로 오류가 발생한 위치 정보가 우측에 표시된다.

두번째 라인은 어플리케이션에 console.log()를 이용하여 로그를 출력한 모습이다. 역시 우측에 로그가 출력된 위치정보가 표시된다.

세번째와 네번째 라인은 XMLHttpRequest log 이며, 이 로그는 XMLHttpRequest 기능을 활성화 시킬 경우 나타난다.

다섯번째와 여섯번째 라인은 Console 에 직접 표현식을 작성한 것으로 그 표현식의 결과가 하단에 즉시 출력되며 값을 검사할 수 있다.

7.1.5.2 Console 패널 참고

Console 패널의 이용방법은 아래 링크를 통해 보다 더 자세히 확인해볼 수 있다.

<https://developers.google.com/chrome-developer-tools/docs/console?hl=ko-KR>

7.1.6 기타 패널

지금까지 설명한 Elements 부터 Source 및 Console 가지의 패널 외에도 Timeline, Profiles, Audits 패널들이 제공되고 있으며, 이 기능들 또한 강력한 어플리케이션 분석도구들이다.

이러한 기능들에 대한 자세한 사용방법은 아래 링크를 통해 확인할 수 있다.

<https://developers.google.com/chrome-developer-tools>

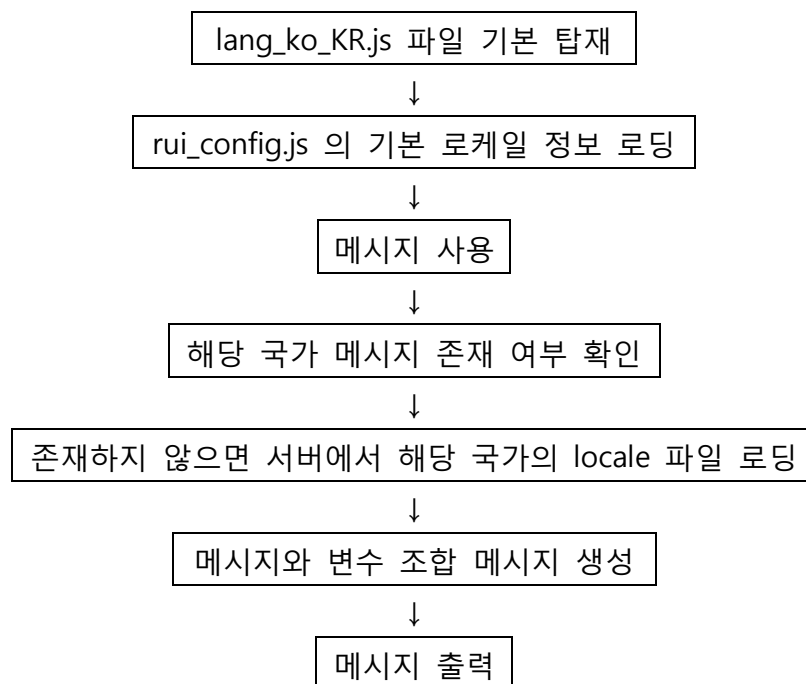
8. RichUI 다국어 처리

8.1 개요

DevOn RichUI 는 한국어와 영어, 2 개의 다국어 파일을 기본 지원한다. 2 개의 국가 외에 추가로 다국어를 지원하고자 한다면 국가별로 다국어 파일을 추가하여 다국어를 지원할 수 있다. 다국어는 단순한 문자 처리 이외에 날짜 처리, 금액 처리, 숫자 포맷을 지원한다. 다국어 파일은 현재 시스템 기본 언어 파일만 탑재하고 해당 국가의 언어를 사용할 경우 서버에서 메시지 파일을 읽어와서 탑재한다.

다국어 중 아랍어 관련 기능(RTL : Right to Left)은 지원하지 않는다.

8.1.1 다국어 지원 처리 흐름



8.1.2 설정 파일

RichUI 에서 다국어를 지원하기 위해 필요한 파일은 2 가지 파일로 설정된다.

/rui/resources/rui_config.js	현재 시스템의 기본 정보가 들어 있는 환경 파일
/rui/resources/locale/lang-XX_XX.js	국가별 다국어 처리 방식 및 메시지가 들어 있는 파일

rui_config.js

```
Rui.config.ConfigData = {
  core: {
    /*
     * 기본 언어 date, message등에 영향을 준다.
     */
    defaultLocale: 'ko_KR', ←시스템 기본 로케일 정보
  }
}
```

8.1.3 주의사항

Javascript 는 소스 코드의 양이 많아지면 성능이 심각하게 저하되므로 공통 메시지 등에 출력할 내용만 처리하고 Grid, Label 등에 들어가는 고정된 메시지들이나 서버에서 처리가 가능한 메시지들은 RichUI 의 메시지 처리 방법보다는 서버의 개발 방법으로 처리하는 게 성능 및 메모리 활용 측면에서 좋다. 즉, 메시지는 사용 용도에 따라 javascript 와 서버 프레임워크 측의 메시지 처리 방식을 적절히 혼용해서 사용해야 한다.

8.2 설정

8.2.1 디렉토리 및 파일 규칙

다국어 파일의 위치는 rui_config.js 에 기본 설정되어 있다.

```
message: {
  /*
   * 다국어 지원 message 파일 위치
   */
  localePath: '/resources/locale' ← 다국어 파일 위치
}
```

기본 디렉토리 : /rui/resources/locale

파일명 : lang- + <국가별 Locale 문자열> + .js

Locale 은 rui_config.js 에 등록된 defaultLocale 의 문자열을 그대로 인식한 파일로 처리된다.

8.2.2 Locale 파일 작성 방법

Locale 파일 형식은 Json Object 로 총 5 개의 속성으로 구성되며 JsonPath 를 이용하여 접근된다.

Locale	파일이 인식할 locale 문자열
Core	rui_base.js 와 연동되며, 날짜 변환 및 기본 변수로 적용된다.

Base	시스템 내부 메시지나 유효성 체크 관련 변수로 적용된다.
Ext	Rui 의 Plugin 에 적용된 다국어 메시지의 변수로 적용된다.
Message	프로젝트에서 사용할 수 있는 메시지의 변수로 적용된다.

Locale, core, base, ext 항목에는 프로젝트에서 임의로 메시지 코드를 추가할 수 없으며, 메시지의 내용만 수정이 가능하고 message 항목에 프로젝트에서 국가에 맞게 다국어를 추가하여 사용할 수 있다.

8.2.3 신규 locale 파일 작성 방법

기본으로 제공되는 locale-ko_KR.js 파일을 복사한 후 작성하고자 하는 국가의 locale 파일 이름으로 파일을 새로 만든다.

예) 일본 다국어 파일명 : locale-ja_JP.js

새로 작성한 파일 내용들을 해당 국가에 맞게 번역하여 저장한다.

8.2.4 메시지 작성 규칙

메시지는 키와 값으로 구성된다. 키 문자열의 작성 규칙은 앞 문자열이 msg 로 시작되어야 하며, 3 자리의 시퀀스 문자열을 포함한다. 값은 출력할 문자열과 변수에 따라 대체할 @ 문자로 구분되며, @는 여러 번 사용 가능하다.

msg003: "@자리수만큼 입력하십시오.", ←@는 대체할 변수	
↑	↑
키	값

8.2.5 Core 단계의 변수 설명

monthInYear	각 국가의 달 표시가 포함된 문자열 (2 자리로 채워야 한다.)
monthInYear1Char	각 국가의 달 표시가 없는 문자열 (2 자리로 채워야 한다.)
shortMonthInYear	각 국가의 달 표시가 포함된 문자열. (앞 숫자 0 은 제외한다.)
dayInWeek	각 국가의 달 표시가 없는 문자열 (앞 숫자 0 은 제외한다.)
shortDayInWeek	한 주의 요일 문자열
weekdays1Char	한 주의 요일 문자열
startWeekDay	한 주의 시작 요일의 숫자 변수 (0: 일요일, 1: 월요일~6:토요일)
localeMonths	수정하면 안 된다.

localeWeekdays	수정하면 안 된다.
dateDelimiter	수정하면 안 된다.
dateRangeDelimiter	수정하면 안 된다.
myLabelMonthSuffix	월 표시 문자열
myLabelYearSuffix	년 표시 문자열

8.2.6 LDateLocale 작성 방법

날짜 포맷을 변환 할 경우 내부적으로 LDateLocale 객체를 사용한다. 날짜를 사용자에게 출력할 경우 %Y-%m-%d 로 출력을 하면 다국어 처리를 할 수 없다. 그래서 일관성 있는 출력 방법들은 LDateLocale 에 적용된다.

8.2.7 LDateLocale 변수 설명

p(소문자)	['AM', 'PM'], ' 오전, 오후 대문자 변수
P(대문자)	['am', 'pm'], ' 오전, 오후 소문자 변수
q(소문자)	'%m%d%Y', ' YYYYMMDD 문자열의 다국어 변수
Q(대문자)	'%m%d%Y%H%M%S', ' YYYYMMDDHH24MISS 문자열의 다국어 변수
r(소문자)	'%I:%M:%S %p', ' 10:10:10 오후 문자열의 다국어 변수
x(소문자)	'%d/%m/%Y', ' YYYY/MM/DD 문자열의 다국어 변수 (구분자 있다)
X(대문자)	'%d/%m/%Y %T' ' YYYY/MM/DD HH24:MI:SS.SSS 문자열의 다국어 변수 (구분자 있다)

이 외의 변수는 API 에서 Rui.util.LDate 객체를 참조한다.

8.3 구현

8.3.1 다국어 설정값 변경 방법

RuchUI 에서는 모든 설정 정보는 LConfiguration 을 이용한다. LConfiguration 은 Rui.getConfig 메소드를 통해 얻어올 수 있다. LConfiguration 으로 아래와 같이 위에서 설명한 defaultLocale 정보를 변경하면 메시지 파일 및 LDateLocale 정보가 해당 locale 정보로 처리된다. set 메소드에서 앞 문자열은 rui_config.js 에 경로에 맞는 JsonPath 로 접근한다.

```
예)
Rui.getConfig().set('$core.defaultLocale', ['en_KR']);
```

```
예) 서버값으로 변경 방법
Rui.getConfig().set('$.core.defaultLocale', ['<%=locale%>']);
```

8.3.2 메시지 사용 방법

메시지를 사용하려면 LMessageManager 를 이용해서 값을 가져온다. LMessageManager 객체는 Rui.getMessageManager 메소드로 얻어올 수 있다.

아래 예제는 msg003 문자열에 해당되는 내용을 lang-XX_XX.js 파일에 찾아 변수 개수를 확인하여 호출 시 배열을 생성하여 변수를 전달한다.

```
예)
alert(Rui.getMessageManager().get('$.base.msg003', [10]));
                                ↑      ↑
                                키      변수

결과)
10 자리수만큼 입력하십시오.
```

8.3.3 그리드에서의 다국어 처리

그리드에서 날짜 다국어 처리는 LDataSet 의 field type 이 'date' 속성으로 설정되고 금액이나 숫자 다국어 처리는 field type 이 'number' 속성으로 설정한 후 그리드에서 renderer 로 구현된다.

```
예)
var dataSet = new Rui.data.LJsonDataSet({
    id: 'dataSet',
    fields: [
        { id: 'col8', type: 'number', defaultValue: 0 }, ← type 0 | number
        { id: 'date1', type: 'date', defaultValue: new Date() } ← type 0 | date
    ]
});

.....

var columnModel = new Rui.ui.grid.LColumnModel({
    columns: [
        { field: 'col8', renderer: 'money' }, ← 금액 renderer
        { field: 'date1', renderer: 'date' } ← 날짜 renderer
    ]
});
```



```
});
```

다국어 지원을 위한 기본 renderer 는 date, money, number, time 로 구성된다.

8.3.4 그리드에서의 사용자 renderer 처리

사용자 renderer 는 function 안에서 유틸리티 객체를 사용하여 처리할 수 있다.

```
예)
.....
var columnModel = new Rui.ui.grid.LColumnModel({
  columns: [
    { field: 'col8', renderer: function(val) {
      return Rui.util.LFormat.moneyFormat (val, 'en_US');
    }
  },
    { field: 'date1' , renderer: function(val) {
      Return Rui.util.LDate.format(val, { format: '%x'});
    }
  }
]
});
```

8.3.5 유틸리티를 이용한 값 변환 처리

숫자나 날짜의 경우 문자 포맷을 변환할 필요가 있을 경우 유틸리티 객체를 사용하여 변환할 수 있다. 유틸리티 객체로는 javascript 의 기본 prototype 과 Rui.util.LFormat 객체를 이용하여 변환 가능하다.

```
예)
prototype 사용 방법
'20130621'.format('%Q')
결과
new Date(2013, 5, 21) → Date 객체로 변환된다.

Rui.util.LFormat 의 숫자 변환 사용 방법
```

```
Rui.util.LFormat. numberFormat(10000);
```

결과

10,000

Rui.util.LFormat 의 금액 변환 사용 방법

```
Rui.util.LFormat. moneyFormat(10000);
```

결과

\$10,000

9. 버전관리

9.1 상위버전의 추가된 기능 사용

DevOn RichUI 1.x 또는 보다 하위의 버전을 사용중인 프로젝트에서 RichUI 2.0 에 추가된 기능을 사용하고자 할 경우에 RichUI 2.0 을 RichUI 버전관리 체계에 포함시켜 사용할 수 있다.

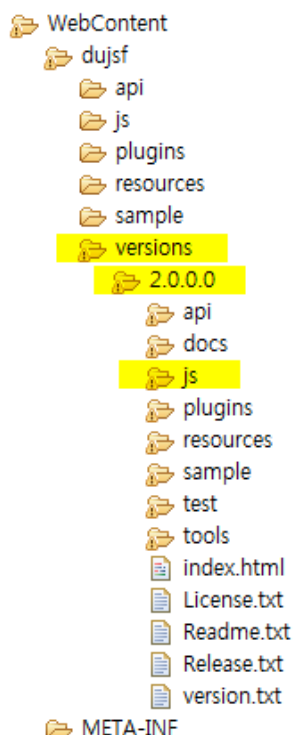
이런 사용 방법은 프로젝트 전체의 2.0 업그레이드가 아닌, 해당 기능이 필요한 일부 몇 개의 페이지만 이러한 방식으로 사용하기를 권고한다.

9.1.1 RichUI 1.x 환경에서 RichUI 2.0 사용

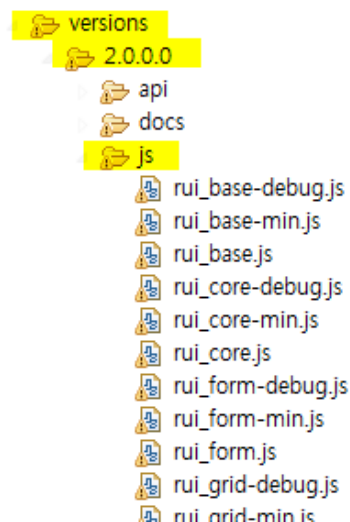
가령 RichUI 1.x 를 사용중인 프로젝트에서 RichUI 2.0 의 Grid 소계 기능을 이용하고자 한다면 다음과 같이 구성하여 사용할 수 있다.

9.1.1.1 RichUI 2.0 을 workspace 에 추가

그림과 같이 RichUI 1.x 가 설치된 workspace 의 dujsf 하위에 versions 폴더를 만들어 새로운 버전의 RichUI 를 설치한다.



versions 폴더 하위에는 설치하려는 RichUI 의 버전명으로 폴더를 만든 후 하위에 RichUI 2.0 을 설치한다



9.1.1.2 추가한 RichUI 2.0 의 라이브러리로 개발

프로젝트에서 사용하던 기존 RichUI 1.x 와 2.0 을 한 페이지에서 같이 사용할 수 없다. 즉 필요한 기능에 의해 RichUI 2.0 을 사용해야 한다면 해당 페이지만 RichUI 2.0 을 이용하여 개발하는 것이다.

예) RichUI 1.x으로 개발 한 페이지

```
<script type="text/javascript" src="/dujsf/js/du_base-debug.js"> </script>
<script type="text/javascript" src="/dujsf/js/du_core-debug.js"> </script>
<script type="text/javascript" src="/dujsf/js/du_ext_base-debug.js"> </script>
<script type="text/javascript" src="/dujsf/js/du_ext-debug.js"> </script>
```

이와 같은 형태로 RichUI 라이브러리를 추가했었다면

예) RichUI 2.0으로 개발 할 페이지

```
<script type="text/javascript" src="/dujsf/versions/2.0.0.0/js/rui_base-debug.js"> </script>
<script type="text/javascript" src="/dujsf/versions/2.0.0.0/js/rui_core-debug.js"> </script>
<script type="text/javascript" src="/dujsf/versions/2.0.0.0/js/rui_ui-debug.js"> </script>
<script type="text/javascript" src="/dujsf/versions/2.0.0.0/js/rui_form-debug.js"> </script>
<script type="text/javascript" src="/dujsf/versions/2.0.0.0/js/rui_grid-debug.js"> </script>
```

이와 같은 형태로 RichUI 라이브러리를 추가하여 2.0 의 Grid 소계 기능을 이용한다.

9.1.1.3 주의사항

여러 버전의 RichUI 를 한 개의 페이지에서 복합적으로 사용할 수 없다. 하나의 화면에서는 반드시 한 개의 버전만을 사용해야 한다.

프로젝트가 기능 요구사항에 따라 페이지 별로 다른 버전을 사용해야 하는 상황에서, 기본 버전이 아닌 다른 버전의 RichUI 를 사용하는 페이지가 점차적으로 많아질 수 있으며 이런 경우는 RichUI 의 버전 업그레이드를 고려해야 한다.

RichUI 의 버전 업그레이드 문의

RichUI 버전 업그레이드 문의	
DevOn	http://www.dev-on.com
e-Mail	devon@lgcns.com

10. 대용량 데이터 처리

10.1 대용량 데이터 로드

그리드 기능을 가지고 있는 웹표준 솔루션들은 데이터를 Load 할 때 javascript 에서 데이터 변환 및 사용이 자유로운 json 을 이용한다.

Json 처리 방식에서는 서버로 부터 Ajax 로 받아온 문자열(responseText)을 json 객체로 변환 할 때 브라우저 자체의 window.JSON 메소드를 이용하여 변환한다. 하지만 IE 6/7 버전의 경우 window.JSON 객체를 지원하지 않으므로 eval 메소드를 통해서 변환해야 하고 이때 Client 성능에 따라 json 객체로 변환할 때 성능 저하가 발생하고 메모리가 많이 증가되는 등, 반복적인 대량 데이터 처리를 할 경우 전반적으로 성능 저하가 심각하게 발생한다.

RichUI 는 대용량 데이터 처리 방식으로 Delimiter 처리 방식을 제공하고 있다. Delimiter 처리 방식은 브라우저에서 제공되는 window.JSON 변환 방식이 아닌 기본 javascript 의 split 메소드를 이용하여 직접 json 객체로 변환한다. 특히 로드할 때 모두 변환하지 않고 프로그램에서 당장 사용하지 않는 레코드들은 문자열로 보관하고 있다가 필요한 경우 변환하여 사용하므로, 초기 변환 성능을 높이고 메모리도 최소화하며 처리한다.

10.1.1 LJsonDataSet 과 LDelimiterDataSet 의 차이점

	LJsonDataSet	LDelimiterDataSet
성능	장점 -소량건 빠름(1,000 건 이하) 단점 -많은 양의 데이터는 PC 성능에 따라 심하게 성능저하가 발생됨.	장점 -대량건 처리시 빠름(10,000 건 이상) 단점 -초기 로드 시에 전체 데이터에 대해 변경이 발생하면 성능저하가 발생됨.
메모리	장점 -초기 로드 시에 변경이 많아도 메모리 증가에 영향이 적음. 단점 -IE 6,7,8 에서 Memory Leak 이 조금씩 발생함.	장점 -초기 로드 시 메모리가 적음 단점 -전체 데이터를 변경하면 메모리가 더 많이 증가함
사용성	장점 -필드 순서가 변경되어도 자동으로 맵핑됨	단점 -필드 순서를 개발자가 정확하게 맞춰서 개발해야 함. 서버 측에서 순서가 바뀔 경우 UI 도 바뀌야 함. (개발 복잡성 증가)
권장	일반적일 경우 권장	화면이 복잡하지 않고 대량 데이터

(30,000 건 이상)를 로드 할 경우 사용

10.1.2 LDelimiterDataSet 방식 구조

서버와 주고 받는 데이터의 문자열중 DataSet 및 레코드 등을 구분하는 구분 코드 값이 존재한다. 서버에서 데이터를 아래의 구조로 문자열을 생성하여 리턴 한다.

LDelimiterDataSet 의 기본값으로는 아래와 같이 정의되어 있다.

속성	구분자	설명
dataSetDelimiter	¶	데이터셋의 전체 구조의 반복 구분자
recordDelimiter	₩₩₩(엔터)	레코드의 반복 구분자
fieldDelimiter	₩	필드의 반복 구분자

config 값으로 대체 가능

데이터셋의 마지막은 메시지 문자열로 처리된다. 메시지가 없으면 ¶ 문자열을 반복한다.

첫번째 DataSet 이름 + ¶ + 필드값 1 + ₩ + 필드값 2 + ₩ + 필드값 3 + (리턴값) + ¶ + 데이터셋 메시지 + ¶ + 두번째 데이터셋 이름 + 필드값 1 + ₩ + 필드값 2 + ₩ + 필드값 3 + (리턴값) + ¶ + 데이터셋 메시지

예)

두개의 데이터셋을 읽어오는 데이터 예제

```
dataSet1¶0*r1001*Kaden Q. Delacruz    ← 엔터값은 레코드 구분자
0*r1001*Kaden Q. Delacruz¶return value and so on...¶dataSet2¶0*r1001*Kaden Q. Delacruz
0*r1001*Kaden Q. Delacruz¶return value and so on... ← 데이터셋 마지막은 데이터셋 메시지
```

10.1.3 LDelimiterDataSet Load 방법

```
var dataSet = new Rui.data.LDelimiterDataSet({ ← 생성자만 LDelimiterDataSet 으로 교체한다.
  id: 'dataSet',
  fields: [
    { id: 'col1' },
    { id: 'col2' },
    { id: 'col3' },
    { id: 'col4' },
    { id: 'col5' }
  ]
})
```

```
});

dataset.load({
    url: 'delimiterData.txt'
});
```

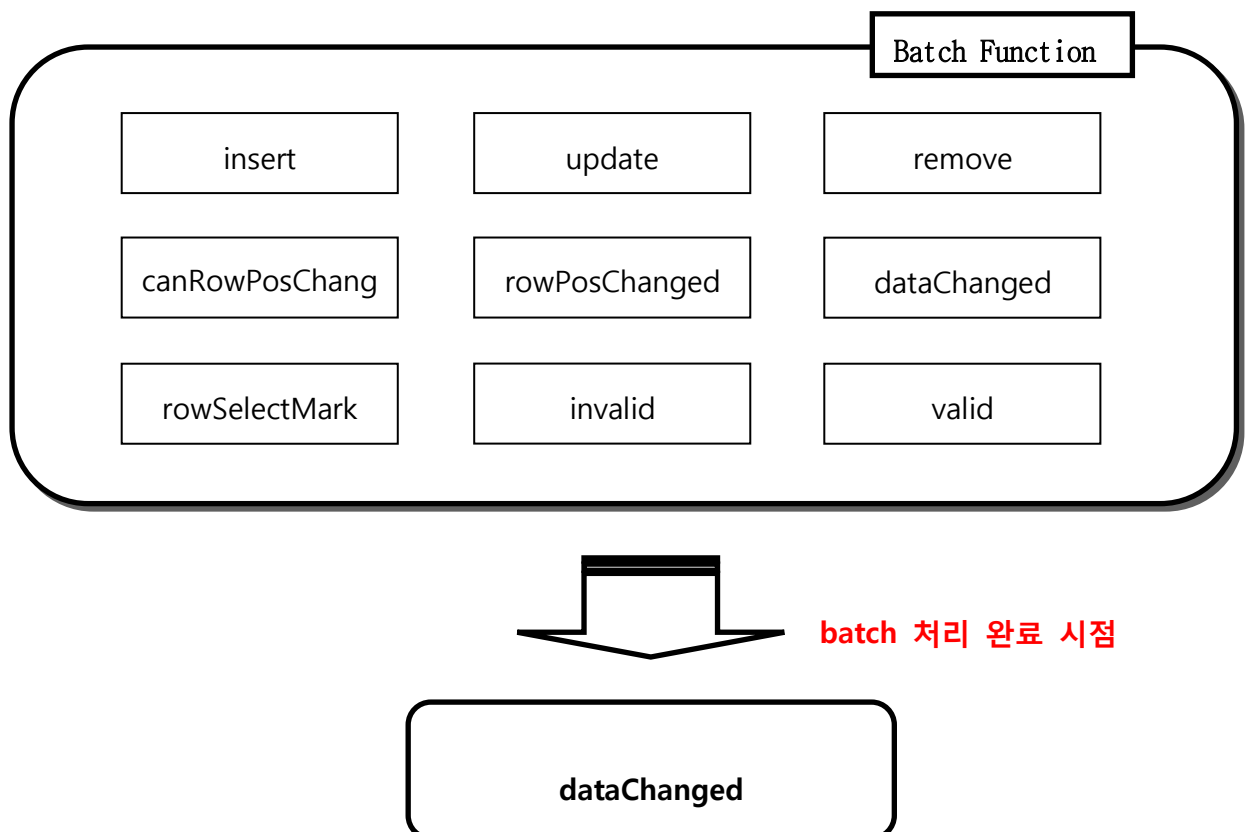
LJsonDataSet 구현 방식이 제공하는 모든 기능은 동일하게 사용할 수 있다.

10.2 대용량 데이터 변경

대량의 데이터를 DataSet 에서 반복 변경할 경우 이벤트가(add/update/remove/rowPosChanged/canRowPosChange) 발생하여 DataSet 과 연동되는 각 컴포넌트들이 이벤트 처리마다 렌더링을 시도하며 이에 성능이 저하된다.

따라서 RichUI 의 DataSet 은 이러한 경우 성능 저하 없이 처리 할 수 있도록 batch 기능을 제공한다. Batch 는 batch function 내부에서 수행되는 DataSet 변경 메소드들이 호출되어도 호출 때마다 이벤트를 발생시키지 않고 변경 작업이 일괄 완료되는 시점에서 dataChanged 이벤트만 발생시키며 이 때 DataSet 과 연동되는 모든 컴포넌트들이 단 한번만 재 렌더링을 수행하여 대량의 데이터 변경 시에도 빠른 처리 속도를 보장한다.

10.2.1 Batch 수행 시 무시되는 이벤트



10.2.2 Batch 메소드

10.2.2.1 설명

DataSet 의 대량건의 변경이 발생하여 성능 저하가 발생할 경우 이벤트를 발생시키지 않고 처리하는 메소드이다.

10.2.2.2 메소드 실행 파라미터

Javascript Function

10.2.3 Batch 처리 방법

10.2.3.1 신규 건의 데이터를 반복적으로 추가하는 예

```
dataSet.batch(function(){
    for(var i = 0 ; i < cnt; i++) {
        var r = dataSet.createRecord({
            col1: 'col1 값',
            col2: 'col2 값',
            col3: 'col3 값',
            col4: 'col4 값',
            col5: 'col5 값'
        });
        dataSet.add(r);
    }
});
```

10.2.3.2 첫번째 DataSet 을 두번째 데이터셋에 복사하는 예

```
dataSet.batch(function(){
    for(var i = 0, len = dataSet.getCount(); i < len; i++) {
        var r = dataSet.getAt(i).clone(); ← 레코드를 clone 한다.
        dataSet2.add(r);
    }
});
```