

**TMT-COMMON**

개발 가이드

V1.0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 구분 | 성명 | 일자 | 의 견 | 서명 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Document #: SMUIS\_II\_DE\_프레임워크 구성방안(분석)

**문서 이력**

| 버전 | 작성일 | 변경내용[[1]](#footnote-1) | 작성자 | 승인자 |
| --- | --- | --- | --- | --- |
| 1.0 | 2020-03-04 | 제정 |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**목 차**

[1. 개요 1](#_Toc49954758)

[1.1 목적 1](#_Toc49954759)

[1.2 적용 범위 1](#_Toc49954760)

[2. 화면 UI 코드 작성 규칙 1](#_Toc49954761)

[2.1 디자이너(UI 툴) 설정 1](#_Toc49954762)

[2.2 외부 스크립트 및 CSS 임포트 3](#_Toc49954763)

[2.3 공통 스크립트(javascript) 선언 4](#_Toc49954764)

[2.4 데이터셋(DataSet) 선언 4](#_Toc49954765)

[2.5 샘플화면 목록 8](#_Toc49954766)

[2.6 서브미션(Submission) 선언 9](#_Toc49954767)

[2.7 컨트롤 ID 지정 규칙 11](#_Toc49954768)

[2.8 컨트롤 필드 라벨 지정 12](#_Toc49954769)

[2.9 컨텐츠 그룹 규칙 13](#_Toc49954770)

[2.10 문자열 관련 처리 14](#_Toc49954771)

[2.11 Tab-Order 순서지정 14](#_Toc49954772)

[2.12 그리드(Grid) 설정 15](#_Toc49954773)

[2.13 그리드 디테일 컬럼 설정 18](#_Toc49954774)

[2.14 헤더 컴포넌트 임포트 19](#_Toc49954775)

[2.15 조회조건 접기/펼치기 20](#_Toc49954776)

[2.16 공통 작업버튼 임포트 21](#_Toc49954777)

[2.17 onBodyLoad() 이벤트 선언 23](#_Toc49954778)

[2.18 조회 이벤트 선언 23](#_Toc49954779)

[2.19 그리드 ROW 신규 24](#_Toc49954780)

[2.20 그리드 ROW 삭제 24](#_Toc49954781)

[2.21 CRUD 작업저장 처리 25](#_Toc49954782)

[2.22 프리폼 데이터 바인딩 25](#_Toc49954783)

[2.23 프리폼 CRUD 처리 27](#_Toc49954784)

[2.24 공통 서브미션 호출 서비스 28](#_Toc49954785)

[2.25 콤보박스 바인딩 29](#_Toc49954786)

[2.26 콤보박스 필터링 30](#_Toc49954787)

[2.27 인풋박스 컨트롤 속성 32](#_Toc49954788)

[2.28 넘버에디터 컨트롤 속성 32](#_Toc49954789)

[2.29 체크박스 컨트롤 속성 33](#_Toc49954790)

[2.30 메시지 처리 규칙 34](#_Toc49954791)

[2.31 공통코드 조회 35](#_Toc49954792)

[2.32 화면 커버(Cover) 처리 36](#_Toc49954793)

[2.33 유효성 체크 37](#_Toc49954794)

[2.34 엑셀 데이터 업로드(Import) 38](#_Toc49954795)

[2.35 엑셀 데이터 출력(Export) 42](#_Toc49954796)

[2.36 마지막 작업행 찾기 44](#_Toc49954797)

[2.37 그리드 틀고정 44](#_Toc49954798)

[2.38 그리드 셀병합(Suppress) 45](#_Toc49954799)

[2.39 그리드 소계/합계 46](#_Toc49954800)

[2.40 그리드 ROW 스타일 변경 49](#_Toc49954801)

[2.41 마스크/포맷 처리 49](#_Toc49954802)

[2.42 팝업 호출 53](#_Toc49954803)

[2.43 조회 조건영역 처리관련 54](#_Toc49954804)

[2.44 검색팝업 처리 규칙 55](#_Toc49954805)

[2.45 UDC 개발 56](#_Toc49954806)

[2.46 임베디드 앱 개발 58](#_Toc49954807)

[3. 코드 주석작성 방법 59](#_Toc49954808)

[3.1 UI (eXbuilder6) 59](#_Toc49954809)

[3.2 JAVA 59](#_Toc49954810)

[3.3 SQL 쿼리 61](#_Toc49954811)

[4. 서버단 코드 작성 63](#_Toc49954812)

[4.1 컨트롤러 클래스 작성 63](#_Toc49954813)

[4.2 서비스 작성 66](#_Toc49954814)

[4.3 쿼리 작성 68](#_Toc49954815)

[4.4 시큐어 코딩 70](#_Toc49954816)

[4.5 대용량 데이터 처리 76](#_Toc49954817)

[5. 공통모듈 활용방법 80](#_Toc49954818)

[5.1 첨부파일 업로드 80](#_Toc49954819)

[5.2 첨부파일 다운로드 84](#_Toc49954820)

[5.3 엑셀 업로드팝업 85](#_Toc49954821)

[5.4 차트(Chart) 사용 86](#_Toc49954822)

1. 개요
   1. 목적

본 문서는 개발하는 개발자들에게 개발에 대한 정형화된 개발 방법을 제시하여 개발을 보다 표준화및 용이하게 하고자 하는데 그 목적이 있다. 이 문서는 개발하면서 부딪힐 수 있는 문제들을 미리 도출하고 해결방안을 제시한다. 이 문서에게 다루어지지 않은 부분들은 다른 참고자료를 이용하기 바란다.

* 1. 적용 범위

프로젝트 수행 시 작성되는 소스코드에 대하여 적용한다. 단, 프로젝트 관리 목적으로 사용되는 일부 문서 및 소스코드는 예외로 한다.

1. 화면 UI 코드 작성 규칙
   1. 디자이너(UI 툴) 설정

UI 개발을 용이하게 하고자 추가 설정이 필요한 부분을 기술한다.

■ 화면 해상도(크기) 지정

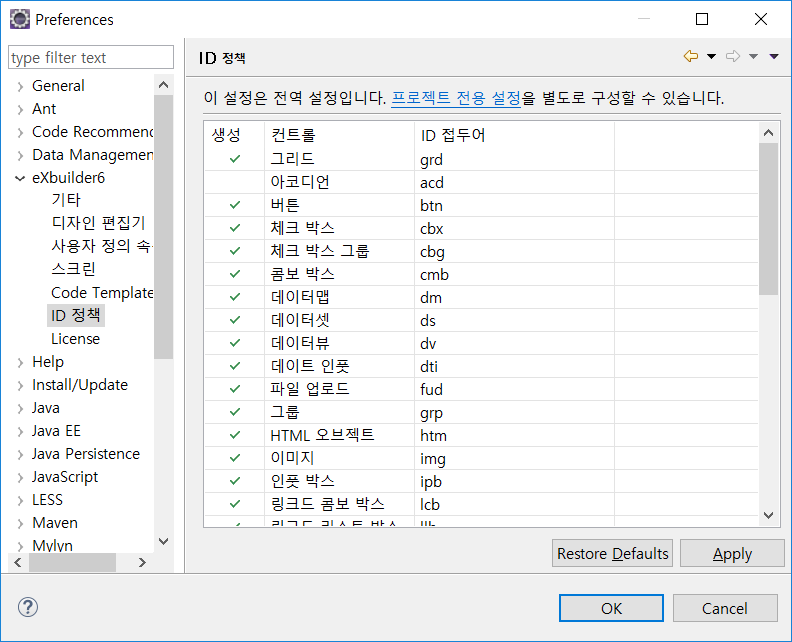
설정 경로: 이클립스 Window > Preferences > eXbuilder6 > 스크린

기본 해상도(화면 크기): 넓이(width): 1320px, 높이(hight): 680px

■ UI 컨트롤 ID 생성 규칙 지정

프로젝트 전역 설정인 ID 생성 규칙을 따른다.

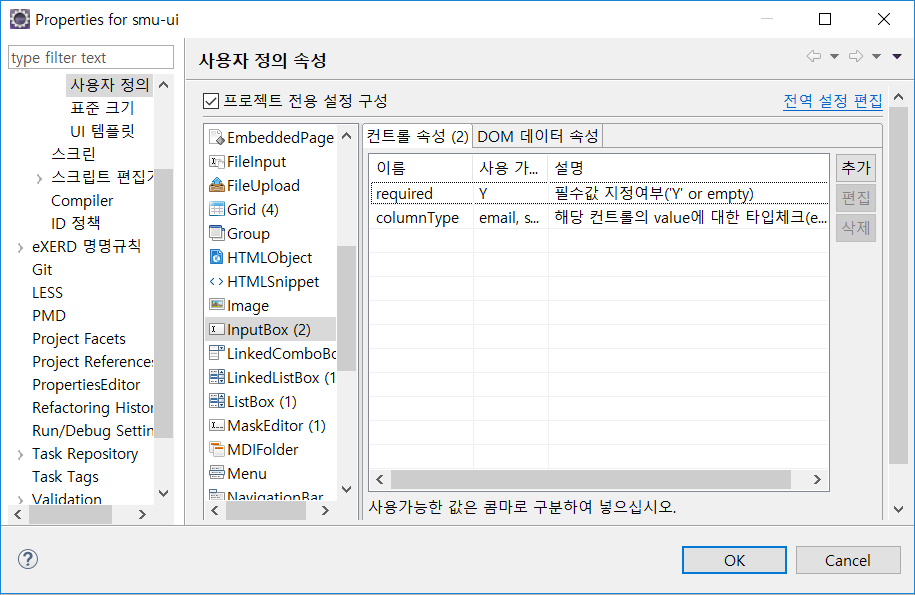
|  |
| --- |
| ID생성 정책 파일: /.settings/id-policy.xmi |

~~~~

■ UI 컨트롤 사용자 정의 속성 지정

프로젝트 전역 설정인 컨트롤 사용자 정의 속성 설정을 따른다.

|  |
| --- |
| ID생성 정책 파일: /.settings/custom-properties.xmi |



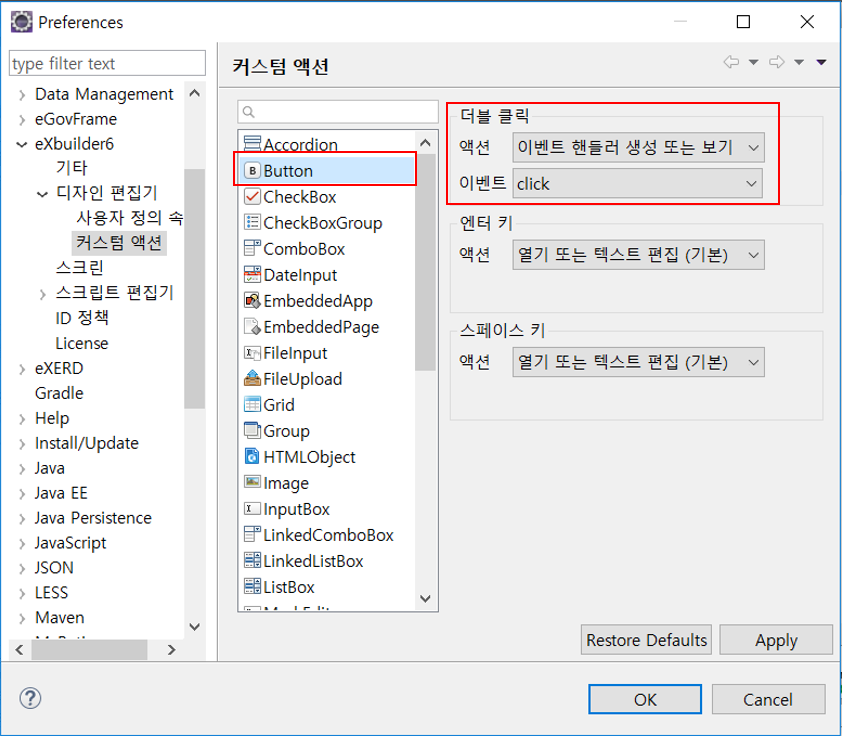
■ UI 컨트롤 커스텀 액션 지정

eXbuilder6 디자이너를 사용하여 UI 화면 개발시, 각 UI 컨트롤에 대해 키입력 또는 더블클릭시 기본 액션을 지정할 수가 있다. 예를 들어, 인풋박스 컨트롤을 선택한 후 Enter키를 누르면 해당 컨트롤에 등록된 이벤트 함수를 편집할 수 있는 스크립트 편집창으로 이동되게 할 수 있다.

여기서는 일반적으로 버튼 컨트롤의 경우에 click 이벤트 함수를 구현하는 경우가 많기 때문에 버튼 컨트롤에 한하여 마우스 더블클릭시 해당 버튼에 click 이벤트를 추가하거나 이미 등록된 이벤트가 있는 경우 해당 함수로 이동하도록 지정하면 개발시 편리하다.

① 이클립스 Window > Preferences > eXbuilder6 > 커스텀 액션 클릭

② 버튼 컨트롤 선택 후, 더블클릭시 액션과 이벤트를 지정한다.



■ 컨트롤 이동 단축키

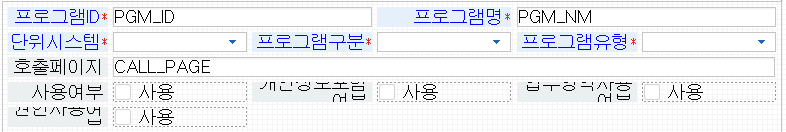
|  |  |
| --- | --- |
| 단축키 | 내용 |
| Ctrl + Alt + 방향키 | 10px씩 이동 |
| Ctrl + Alt + Shift + 방향키 | 1px씩 이동 |

\*\* 특정 PC에서는 Ctrl + Alt + 방향키 클릭시, 화면이 전화되는 경우가 있으며 해당 사항은 각 PC의 그래픽 옵션에 의한 것이다. 만약 화면이 전환되는 현상이 발생하면, 바탕화면 > 우측 마우스 클릭 > 그래픽옵션 > 바로가기 키 > 사용 안함으로 설정한다.

■ 디자이너 폰트 설정

eXbuilder6 디자이너상에서의 폰트를 설정할 수 있다.

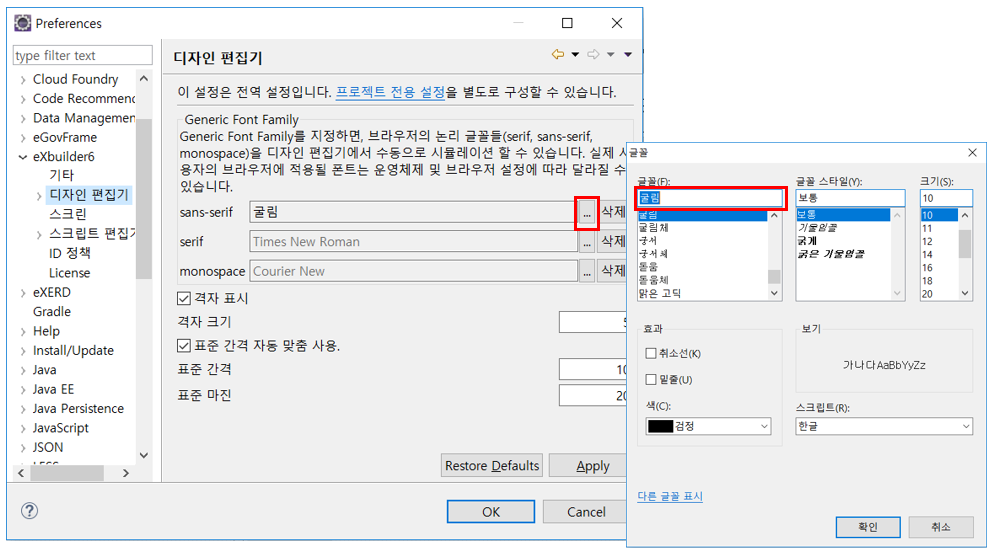
단, 디자이너에서 보여지는 폰트와 실제 해당 화면이 웹브라우저에서 실행됬을 때의 폰트는 다를 수 있다.



① 이클립스 상단의 Window > Preferences > eXbuilder6 > 디자인 편집기 선택

② sans-serif 글꼴에서 버튼 클릭하여, 글꼴 선택 팝업창 호출

③ 글꼴에 ‘굴림’ 입력 후, [확인] 버튼 클릭

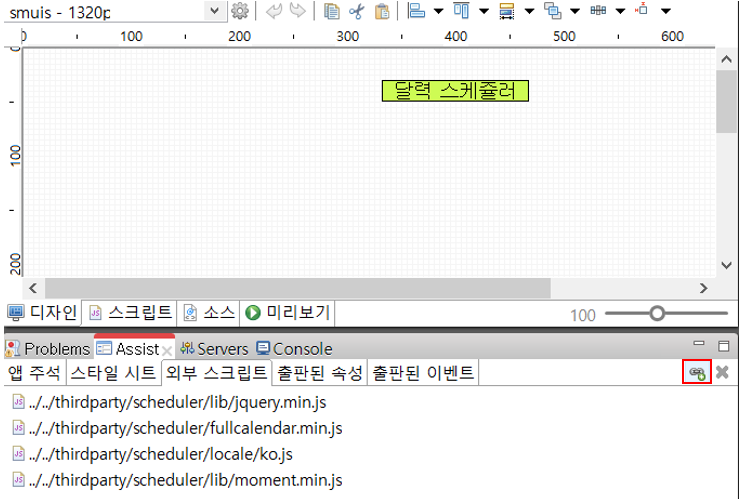


* 1. 외부 스크립트 및 CSS 임포트

기본적으로 통합정보시스템의 화면들에서는 디폴트 테마(Theme)를 지정함으로써 별도의 Javascript 및 CSS를 임포트할 필요는 없다.

단, 업무적으로 특별히 외부 스크립트 또는 CSS를 사용해야 되는 경우에는 임포트를 할 수 있다.

Assist탭 > 외부 스크립트 > 링크 이미지 버튼 클릭



* 1. 공통 스크립트(javascript) 선언

각 UI 화면에서 사용가능한 사이트 공통 스크립트를 선언한다. 공통 스크립트 선언은 각 화면 상단 부분에서 처리한다.

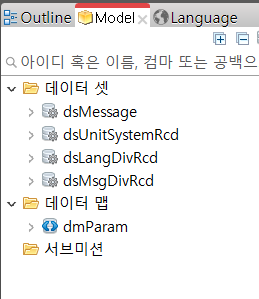
|  |
| --- |
| //공통 스크립트 임포트  var util = createCommonUtil(); |

|  |  |
| --- | --- |
| 파일명 | 기능 |
| common.module.js | 각 화면에 대한 메뉴정보, 사용자정보 및 필수적인 공통 함수들을 제공한다. |
| extension.module.js | 컨트롤들에 대한 값을 얻거나, 핸들링을 하기 위한 공통 API를 제공한다. |
| validation.module.js | 컨트롤에 대한 필수값 체크 등, 유효성 체크를 위한 스크립트이다. |
| util.module.js | 문자열, 숫자, Null 체크 등의 유틸 클래스가 있는 스크립트이다. |

* 1. 데이터셋(DataSet) 선언

각 컨트롤에 바인딩되는 데이터에 대한 명칭 및 구조를 선언한다.

데이터셋은 eXbuilder6 디자이너의 Model창에서 추가할 수 있다.

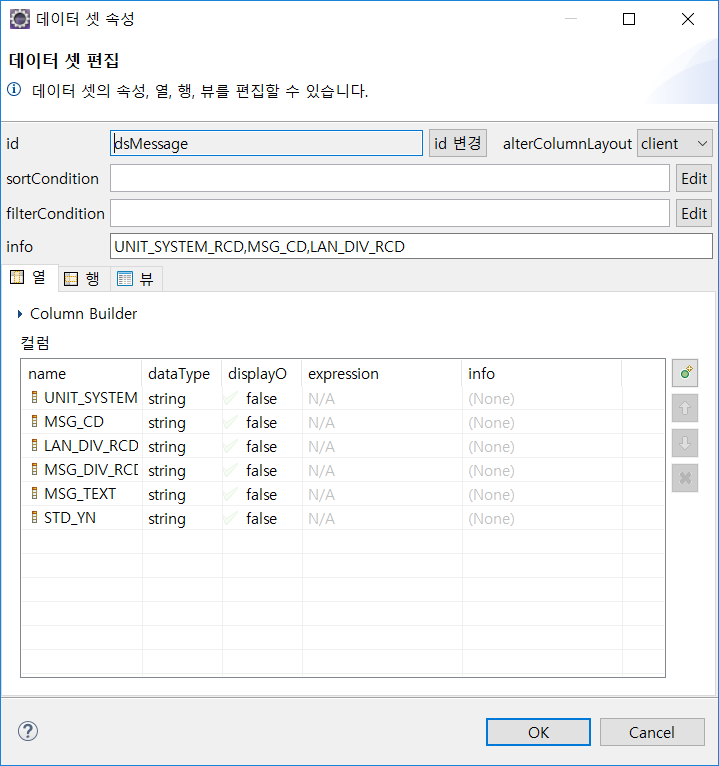


<데이터셋 종류>

|  |  |
| --- | --- |
| 구분 | 비고 |
| 데이터셋 | 주로 그리드를 통해 데이터를 표현하거나, 라디오버튼, 체크박스그룹, 콤보박스 등의 아이템 리스트에 바인딩해서 아이템 리스트를 표현한다.  즉, 목록성 데이터를 담기위해 사용한다. |
| 데이터맵 | Key/Value 쌍을 가지는 데이터, 컨트롤의 단일 속성과 바인딩되어 값을 제공하기 위해 사용된다.  주로 조회조건의 컨트롤에 바인딩하여 사용한다. |

■ 속성 지정

그리드에 바인딩되는 데이터셋인 경우에는 추가적으로 속성들을 지정할 수 있다.



|  |  |
| --- | --- |
| 속성 | 비고 |
| alterColumnLayout | build에 의해 데이터 구조가 재구성될 때 컬럼 구조 변경 기준을 설정한다.  기본값: **client**  client: 디자인 시점에 설정한 컬럼을 기준으로 데이터셋이 구성됨  server: 서버로부터 응답받은 데이터 구조(컬럼명, 컬럼갯수 등)를 기준으로 데이터셋이 구성됨  merge: 디자인 시점에 설정한 컬럼과 서버에서 응답받은 컬럼이 합쳐져서 데이터셋이 구성됨 |
| sortCondition | 데이터셋이 load될때의 기본적인 데이터 정렬 조건을 지정할 수 있다. (여러 개인 경우 콤마(,) 구분자 사용)  Ex) CD ASC : CD컬럼값으로 오름차순 정렬한다.  CD DESC, CD\_NM: CD 컬럼값으로 내림차순 후 CD\_NM으로 오름차순으로 정렬한다. |
| filterCondition | 데이터셋이 load될때의 기본적인 데이터 필터 조건을 지정할 수 있다.  Ex) CD == ‘001’  CD == ‘001’ && CD == ‘002’ |
| **info** | 데이터 그리드에 대한 데이터 저장시, PK 컬럼의 필수값 입력등의 유효성 체크 및 DB에 대한 CUD 작업시 PK값 식별 등을 위해, 해당 그리드에 바인딩되는 데이터셋의 PK 컬럼을 콤마(,) 구분자로 지정한다.  Ex) 1개인 경우: CD  2개인 경우: MST\_CD,CD |

■ 컬럼 지정

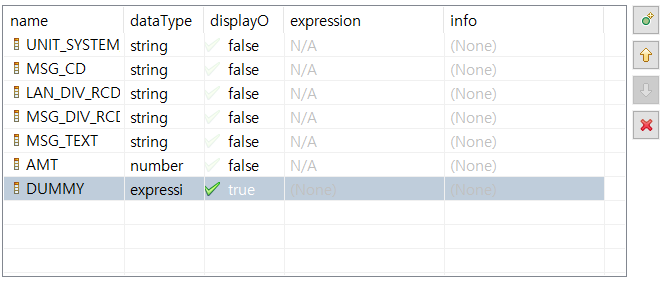
데이터셋의 컬럼을 정의할 때 주의사항은 아래와 같다.

- 데이터베이스에서 조회한 데이터들을 담기 위한 데이터셋인 경우에는 컬럼의 name을 대문자를 사용한다.

- 기본적으로는 string 타입을 사용하며, 숫자 타입의 컬럼인 경우에 한하여 number 유형을 선언하여 사용한다.

> dataType은 매칭되는 DB 테이블의 컬럼의 유형과 다를 수 있으며, 화면에서의 해당 데이터 유형에 맞게 지정해야 함

> 숫자컬럼 정렬시, dataType이 number로 지정되어 있어야 올바르게 내림차순/오름차순 정렬됨



■ 컬럼 빌더(Colum Builder) 사용하기

데이터셋/맵의 컬럼을 정의하는데 있어 eXbuilder6 디자이너는 개발 편의성을 제공한다.

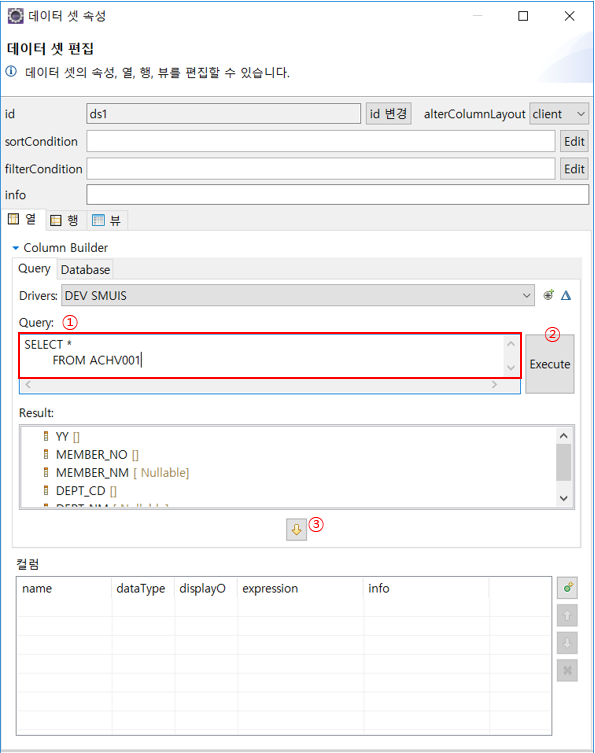
컬럼 빌더 기능을 사용하여, DB 쿼리를 실행함으로써 해당 테이블에 대한 컬럼 정보를 얻어와서 eXbuilder6의 데이터셋의 컬럼을 생성할 수 있다.

사용방법은 아래와 같다.

- Query 창에 실행할 쿼리를 입력한 후, [Execute] 버튼을 클릭한다.

- Result 부분에 쿼리 실행결과에 대한 컬럼 목록이 조회된다.

- 데이터셋의 컬럼으로 생성할 컬럼 목록을 마우스로 선택한 후, [↓] 버튼을 클릭한다.



\*\* 컬럼 빌더에 대한 설정 방법은 SMUIS\_II\_DE\_공통(개발환경설정)\_V1.0.docx 문서를 참고한다.

■ 데이터셋 필터링

특정 조건을 지정함으로써 데이터셋 내의 행(Row)을 필터링할 수 있다. 필터링하는 몇가지 예시는 아래와 같다. 해당 필터링은 데이터셋 API의 setFilter(), setFilterExpr()를 사용하거나, 그룹/풋터에서 getConditionalSum() 그룹함수를 이용한 합계 등의 처리에 사용할 수 있다.

- A컬럼의 값이 100초과인 row만 필터링

|  |
| --- |
| A > 100 |

- A컬럼의 값이 10이상이고, B컬럼의 값이 20이하인 row만 필터링

|  |
| --- |
| A >= 10 && B <= 20 |

- C컬럼의 값에 '가'가 포함되는 row만 필터링

|  |
| --- |
| C \*= '가' |

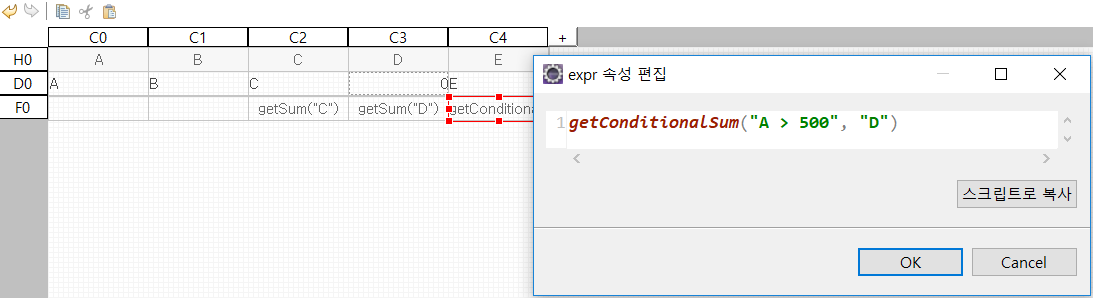
- C컬럼의 값이 '홍'으로 시작되는 row만 필터링

|  |
| --- |
| C ^= '홍' |

- row의 상태가 신규상태인 row만 필터링

|  |
| --- |
| getStateString() == 'I' |

<풋터 사용예시>



■ Expression 컬럼 지정

데이터셋 컬럼에는 서버에서 응답한 데이터들에 없는 추가적인 컬럼을 지정할 수 있다.

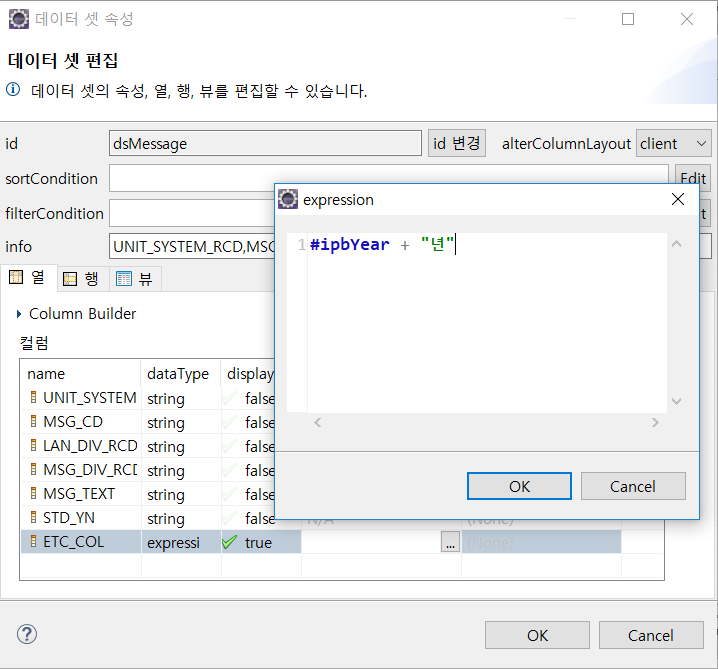
즉, expression 표현식을 통해 데이터들의 조합 및 해당 화면에 정의된 컨트롤들의 값을 ID참조를 통해 데이터를 가공하여 그리드에 표시할 수 있다.

① 데이터셋 선언 팝업창에서 컬럼을 추가한 후, datatype 속성을 expression으로 지정한다.

② expression 항목에서 해당 컬럼에 보여줄 데이터들을 표현식을 통해 지정한다.

* 데이터셋 내의 다른 컬럼 지정시 컬럼명을 명시하면 됨
* 화면내의 컨트롤값 사용하고 하는 경우에 ID 참조를 통한 지정

Ex) #ipbYear.value + ‘년도’



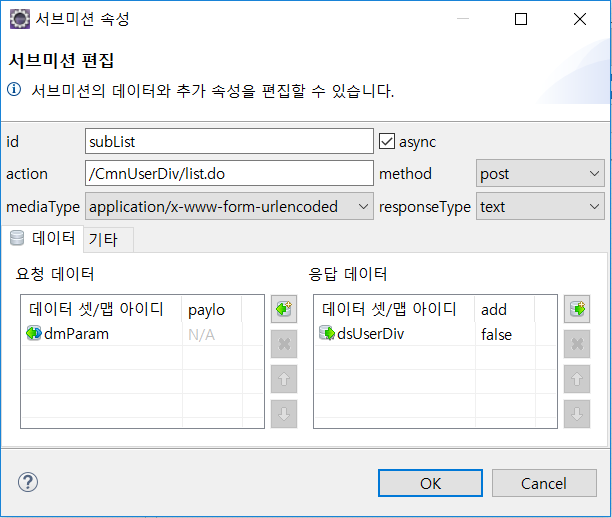
* 1. 샘플화면 목록

화면 패턴별 개발 참고를 위한 템플릿 화면 목록은 아래와 같다.

|  |  |
| --- | --- |
| 유형 | 파일명 |
| One 그리드 | 공통 > 메뉴/업무역할 > 프로그램관리  파일명: cmnCPgm.clx |
| Two 그리드 | 공통 > 코드관리 > 공통코드관리  파일명: cmnCCode.clx |
| 그리드-폼 | 공통 > 연동 > 이메일발송문관리  파일명: cmnCMailMsg.clx |
| 팝업 | 파일명: cmnPCode.clx |
| 검색 UDC | 파일명: udcCmnPCode.clx |

* 1. 서브미션(Submission) 선언

서버에 요청을 보내고, 응답 데이터를 받기 위한 구조 및 프로토콜들을 선언한다.



id: 서브미션 아이디(sub + 액션명)

**action**: 요청URL(규칙: /로 시작)

**async**: 동기화/비동기화 모드(기본: false-비동기화)

**method**: HTTP 요청모드(기본: post)

**mediaType**: 요청 프로토콜(기본: application/x-www-form-urlencoded, 파일업로드인 경우: multipart/form-data)

**responseType**: 응답데이터 유형(기본: text, 파일다운로드인 경우: filedownload)

**요청 데이터:** 서버측에 보낼 요청 파라메터(데이터셋, 데이터맵 아이디 지정)

**응답 데이터:** 서버측에서 응답한 데이터를 받을 데이터셋 또는 데이터맵 아이디 지정

|  |  |  |
| --- | --- | --- |
| 구분 | 비고 |  |
| 요청 데이터 | 데이터셋/맵 아이디 | 요청 데이터로 보낼 데이터셋/맵의 아이디를 지정한다. |
| payload | 요청 데이터에 실어 보낼 데이터 범위를 지정한다.  해당값은 데이터셋인 경우에만 지정 가능하다.  기본값: **modified**  modified: 해당 데이터셋 내의 데이터에서 변경된 행 데이터만 요청 데이터에 포함된다.  all: 변경사항과 관계없이 해당 데이터셋의 모든 데이터행이 요청 데이터에 포함된다. |
| 응답 데이터 | 데이터셋/맵 아이디 | 서버로부터 보내오는 응답 데이터를 받을 데이터셋/맵의 아이디를 지정한다. |
| Alias | 서버에서 받은 응답데이터에 대한 Alias 명을 의미한다.  기본적으로는 서버에서 보내온 데이터 명칭(데이터셋, 데이터맵)과 클라이언트에서 받을 데이터맵/데이터셋 명칭이 동일해야하지만, Alias 속성을 통해 A라는 데이터를 B라는 명칭으로 받을 수 있게된다. |
| add | 서버에서 받은 데이터를 어떤 형태로 데이터셋에 추가할 것인지 여부를 지정한다.  기본값: **false**  true인 경우: 현재 클라이언트에 구성된 데이터셋의 데이터에 추가적으로 서버에서 받은 데이터를 추가한다. (예를 들어, 서브미션 호출 전에 해당 데이터셋에 2개의 행 데이터가 있었고, 서브미션 호출 후 서버에서 2개의 행을 응답받았을 때 총 4개의 행 데이터가 생성된다.)  false인 경우: 현재 클라이언트에 구성된 데이터셋 내의 데이터를 무시하고 서버에서 받은 데이터를 재구성한다. (예를 들어, 서브미션 호출 전에 해당 데이터셋에 2개의 행 데이터가 있었고, 서브미션 호출 후 서버에서 2개의 행을 응답받았을 때 기존 데이터는 리셋(Reset)되고 2개의 행 데이터만 생성된다.) |

< 웹서버에 대한 요청 방법 >

일반적으로 웹어플리케이션의 화면 페이지에서 웹서버에 요청하는 방식에는 크게 2가지 통신방법이 존재한다.

◆ **동기방식**: 사용자가 페이지의 어떤 이벤트를 발생하여 웹서버에 요청하면 웹서버에서 데이터를 응답할 때까지 사용자는 아무런 일을 할 수가 없다. 즉, 사용자는 응답이 온 이후에야 다음 액션을 취할수가 있다.

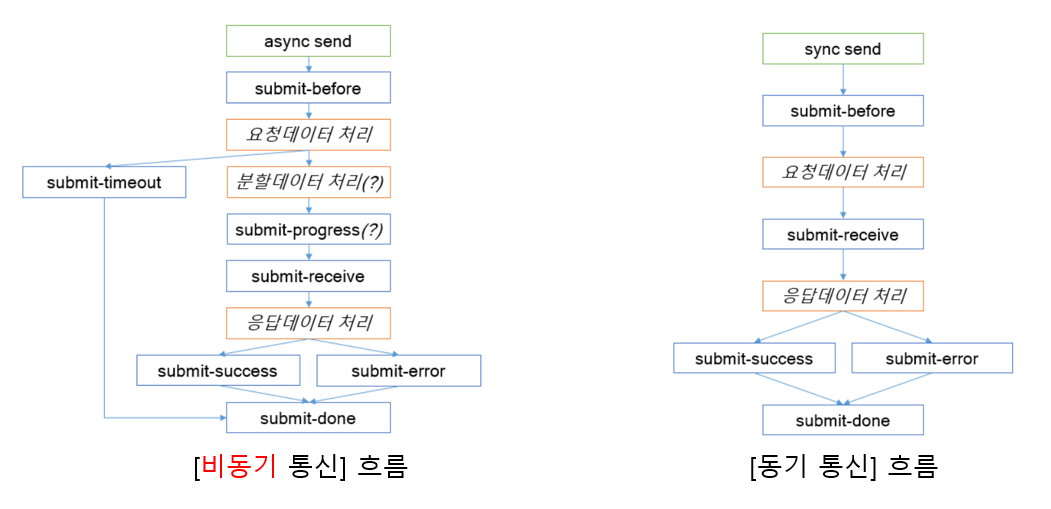
◆ **비동기방식**: 사용자가 페이지의 어떤 이벤트를 발생하여 웹서버에 요청하면 웹서버에서 데이터를 반환하기 전에도 사용자가 다른 이벤트를 취할 수가 있다.

eXbuilder6 UI 프레임워크에서도 웹서버에 대해 동기 또는 비동기 형태로 요청을 보낼 수 있으며, 기본적으로 비동기 방식을 사용한다. 동기/비동기 방식 설정은 서브미션 설정창의 async 옵션으로 지정 가능하다.

|  |  |
| --- | --- |
| **비동기 방식**  **(디폴트)** | * 서브미션 수행이 완료된 이후에 콜백 메소드가 호출된다. 따라서, 서브미션 성공 이후에 처리할 작업 내용이 있는 경우, 반드시 콜백 메소드 내에 기술해야 한다. * **서브미션 수행도중에 프로그래스(Progress) 이미지가 나타난다**.   <사용처>  기본적으로 사용해야 되는 서브미션 호출 방식이며, 사용자의 액션에 의해 발생하는 서브미션인 경우에는 반드시 비동기 방식으로 호출해야 한다.  ex)그리드 목록 조회 및 저장 등의 서브미션에 사용 |
| 동기 방식 | * 서브미션 수행이 완료된 이후에, 그 이후의 스크립트가 수행된다. 따라서, 서브미션 성공 이후에 처리할 작업내용이 있는 경우, 하단에 작업내용을 기술한다. * 서브미션 수행도중에 프로그래스(Progress) 이미지가 나타나지 않는다. |

■ 서브미션의 이벤트 발생순서

간략한 비동기/동기 서브미션 호출시의 이벤트 발생순서는 아래와 같다.



* 1. 컨트롤 ID 지정 규칙

화면에 그려지는 컨트롤들에 대한 ID 규칙은 소스 네이밍룰을 따른다.

프로젝트 내의 소스상에서의 조회조건 영역 및 프리폼에 들어가는 기본 컨트롤들에 대한 ID 지정 규칙은 아래의 룰을 따라 지정한다.

|  |  |
| --- | --- |
| 영역 | 내용 |
| 그리드 | - 그리드 디테일(Detail) 컬럼 항목으로 들어가는 컨트롤들(InputBox, DateInput, NumberEditor 등등)은 별도로 ID를 지정하지 않고 자동 생성된 ID를 사용한다. |
| 조회조건 | - 조회조건 내의 라벨에 대해서는 ID를 지정하지 않는다.  - 조회조건 내의 비필수 항목에 대한 컨트롤에 대해서는 ID를 별도로 지정하지 않는다.  - 조회조건 내의 필수 항목에 대한 컨트롤에 대해서는 ID를 의미있는 명칭으로 별도 지정하며, 조회되는 데이터의 DB 컬럼명과 매핑되는 경우에는 해당 컬럼명을 기준으로 ID를 지정한다.  - 단, 비필수 항목이 아래의 사항에 해당하는 경우에는 별도로 ID를 의미있게 지정해야 한다.  ①이벤트가 달려야 하는 경우  ②해당 컨트롤을 가지고 별도 핸들링해야 되는 경우  ③입력값에 대한 유효성 체크 대상이 되는 경우 |
| 프리폼 | - 프리폼 내의 라벨에 대해서는 ID를 지정하지 않는다.  - 프리폼 내의 비필수 항목에 대한 컨트롤에 대해서는 ID를 별도로 지정하지 않는다.  - 프리폼 내의 필수 항목에 대한 컨트롤에 대해서는 ID를 의미있는 명칭으로 별도 지정하며, 조회되는 데이터의 DB 컬럼명과 매핑되는 경우에는 해당 컬럼명을 기준으로 ID를 지정한다.  - 단, 비필수 항목이 아래의 사항에 해당하는 경우에는 별도로 ID를 의미있게 지정해야 한다.  ①이벤트가 달려야 하는 경우  ②해당 컨트롤을 가지고 별도 핸들링해야 되는 경우  ③입력값에 대한 유효성 체크 대상이 되는 경우 |

* 1. 컨트롤 필드 라벨 지정

조회조건 및 폼 영역에서의 유효성 체크(Validation Check)를 위하여 해당 영역 내의 필수입력 및 유효성 체크 컨트롤에 대한 논리적인 라벨(Label)명을 지정한다.

조회조건 및 프리폼 내의 컨트롤에 대한 필드 라벨 지정은 아래의 규칙을 따른다.

|  |
| --- |
| - 필수 입력항목에 대한 컨트롤에 대해서는 라벨 필드를 반드시 지정한다.  - 기타 유효성 체크 대상이 되는 컨트롤에 대해서는 라벨 필드를 지정한다.  - 그 외의 컨트롤에 대해서는 별도록 라벨 필드를 지정하지 않아도 된다. |

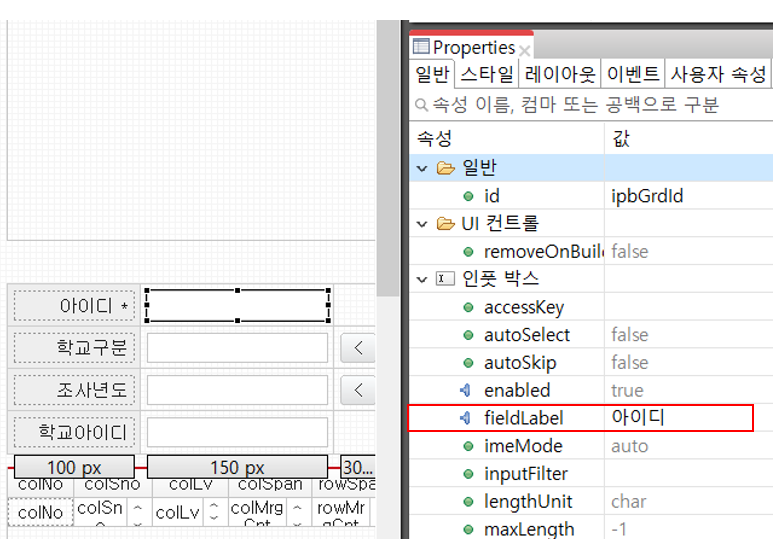
그리드(Grid) 컨트롤에 대한 filedLabel은 데이터 타이틀 및 엑셀 다운로드시 해당 엑셀 파일명으로 사용된다.

|  |
| --- |
| 속성명: fieldLabel |

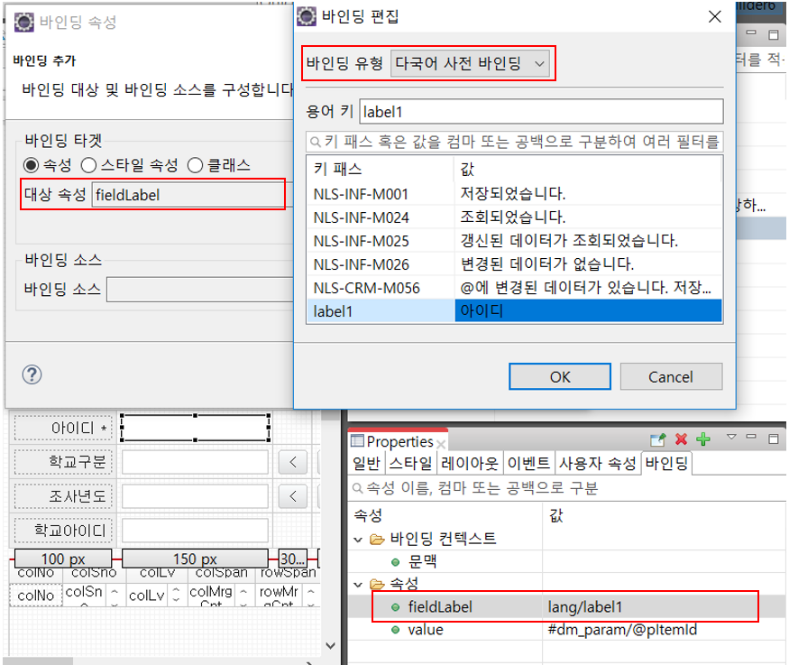
<지정방법>

① 컨트롤을 마우스로 선택한다.

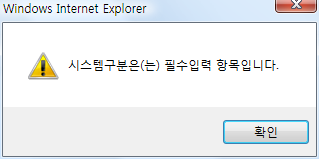
② Properties창에서 해당 컨트롤의 fieldLabel의 속성값에 라벨명을 지정한다.



③ 만약 다국어를 지정해야 하는 경우에는 바인딩탭에서 해당 fieldLabel 속성에 다국어를 지정하면 된다.



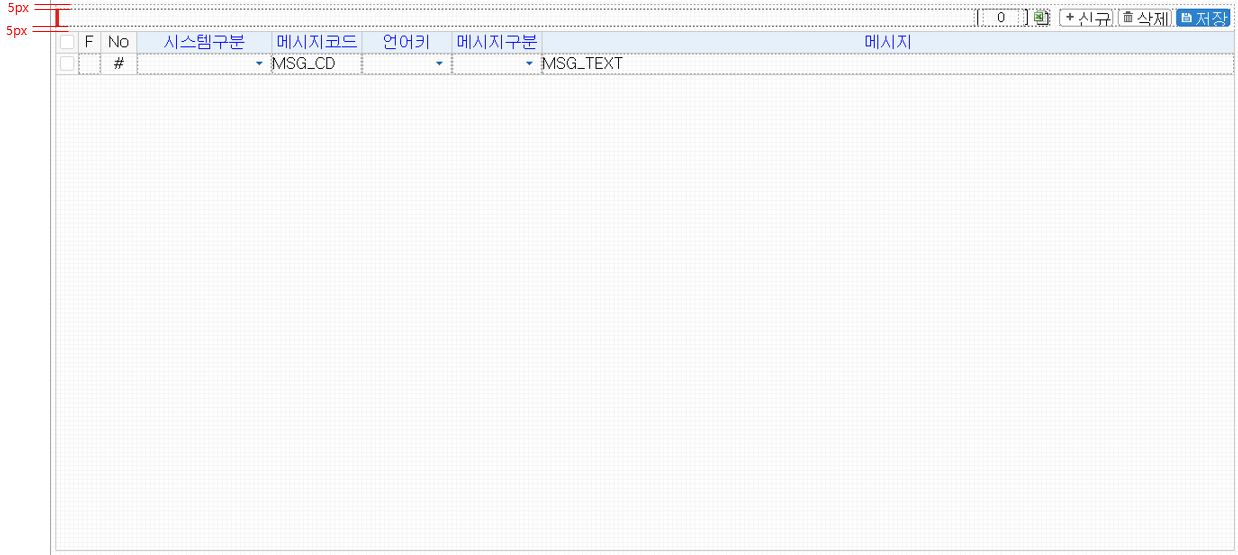
조회조건 및 폼 영역 내의 라벨 컨트롤에 대한 ID를 이와 같이 규정하는 이유는 해당 영역에 대한 유효성 체크시, 해당 컨트롤에 대한 라벨명을 자동으로 보여주기 위한 용도이다.



* 1. 컨텐츠 그룹 규칙

화면 내의 그리드/프리폼 컨텐츠를 감싸는 그룹 컨트롤을 지정하는 규칙에 대해 기술한다.

기본적으로 화면내의 컨텐츠는 변경사항 체크 및 CRUD 작업 핸들링 처리를 위해 그룹 컨트롤로 감싼다.



* 그룹 컨트롤과 그리드/프리폼 컨텐츠간의 여백은 상(5px), 하(0px), 좌(0px), 우(0px)이다.
* 기본적으로 그룹을 감싸는 규칙은 CRUD가 발생하는 컨텐츠 기준으로 감싼다.
* 탭(Tab) 컨트롤 자체에는 그룹컨트롤을 감싸지 않는다.
  1. 문자열 관련 처리

인풋계열(InputBox, TextArea 등) 컨트롤에 대해서는 사용자가 입력 가능한 최대 문자열 길이를 설정하게 되는데, DB2에서는 VARCHAR 타입에 대해서 byte만 제공한다. 따라서, 입력 가능한 길이를 제한해야 하는 인풋계열 컨트롤에 대해서는 아래의 규칙을 따른다.

- 컨트롤의 lengthUnit 속성을 ‘utf8’로 지정한다.

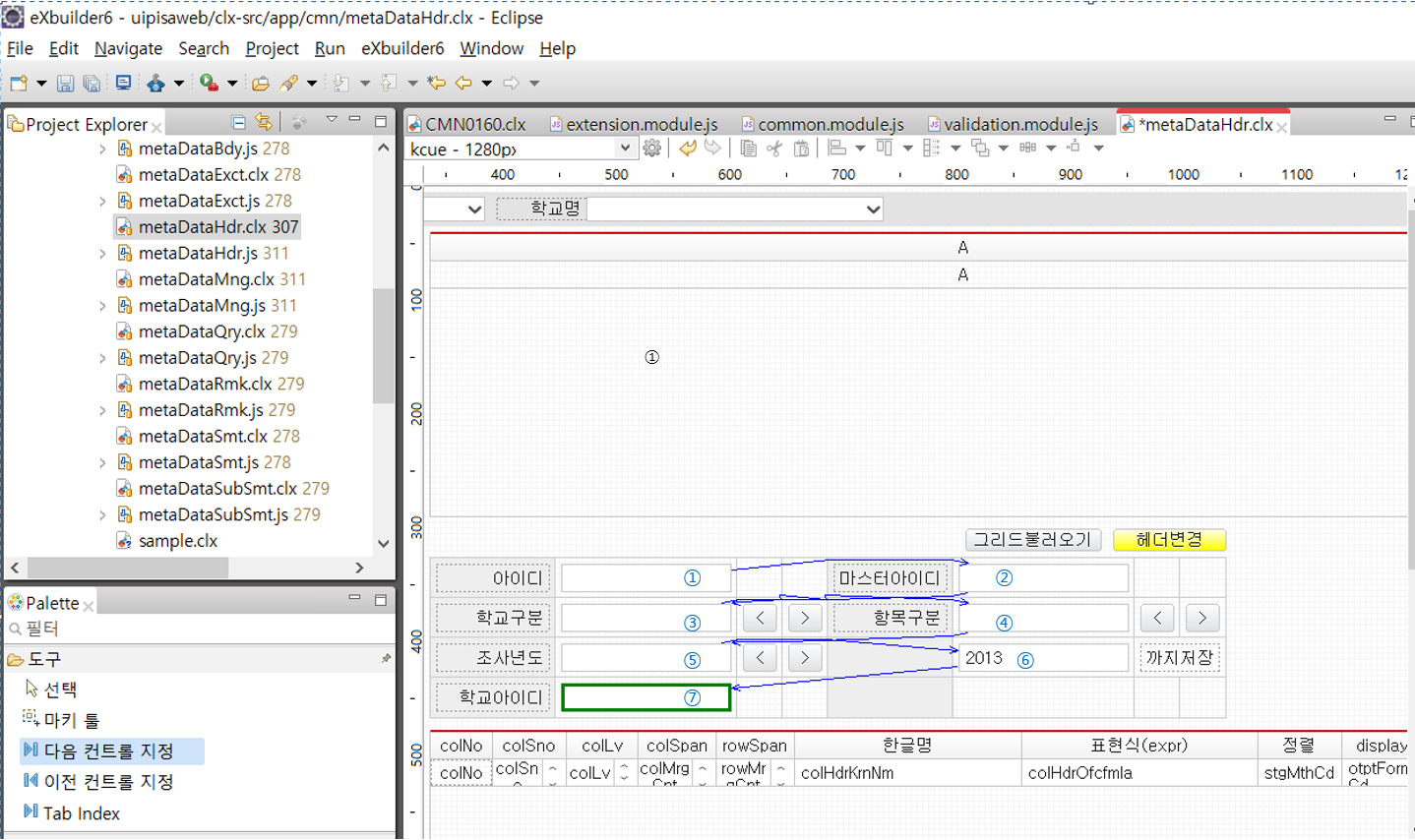
- 컨트롤의 maxLength값을 매핑되는 DB 테이블 컬럼의 길이만큼 지정한다.



* 1. Tab-Order 순서지정

조회조건 영역 및 프리폼 내의 컨트롤들에 대해서는 Tab키 이동에 따른 다음으로 이동할 컨트롤을 지정한다.

적용 대상: 조회조건, 프리폼



① 팔렛트(Palett)뷰에서 다음 컨트롤 지정을 클릭한다.

② 조회조건 입력 컨트롤(InputBox, DateInput, ComboBox 등)에 대하여 탭-오더 지정

③ 기본적인 탭 오더 순서는 아래와 같다.

- 좌에서 우로 지정

- 상에서 하로 지정

- 비활성화된(disabled)된 컨트롤에 대해서는 지정 불필요

* 1. 그리드(Grid) 설정

목록성 데이터에 대한 데이터 그리드(Grid) 컨트롤의 설정값들 정의하는 방법에 대해 기술한다. 해당 속성값(Attributes)들은 Properties 뷰에서 설정 가능하다.

|  |  |
| --- | --- |
| 속성 | 내용 |
| **id** | 그리드 컨트롤 ID 지정, 네이밍룰에 따라 ‘grd’ prefix로 시작하여 ID명을 기술한다.  Ex) grdCode, grdCodeDtl |
| **autoFit** | 그리드의 지정된 너비에 따라서 컬럼 사이즈를 자동으로 조절할지 여부를 지정한다.  기본값: **all**  해당 값이 ‘none’인 경우에는 디자인 시점에 지정한 컬럼 사이즈로 고정된다.  특정 컬럼의 사이즈만 고정시키고자 하는 경우에는 그리드 컨트롤 내의 각 컬럼별 autoFit 설정을 false로 개별적으로 지정하면 된다.  \*\* 인덱스 컬럼 및 그리드 행 선택용 컬럼은 사이즈가 고정되도록 autoFit=false로 지정한다. |
| **clickMode** | 그리드 내의 각 셀(Cell)을 마우스로 클릭시 선택모드를 나타낸다.  기본값: select  select: 마우스로 셀 클릭시, 해당 셀 컬럼이 선택이 된다. 입력을 위해서는 마우스로 한번 더 클릭을 해야 한다.  edit: 마우스로 셀 클릭시 해당 컬럼이 입력 가능한 경우, 입력 모드로 된다. |
| **columnMovable** | 그리드 컬럼이 마우스로 위치가 이동할 수 있는지 여부를 지정한다.  기본값: false |
| **columnResizable** | 그리드 컬럼의 크기가 마우스로 조절될 수 있는지 여부를 지정한다.  기본값: false |
| **dataSetId** | 해당 그리드에 바인딩되는 데이터셋 아이디를 지정한다.  Ex) dsMessage |
| **fieldLabel** | 해당 그리드의 타이틀 명칭을 지정한다.  \*\* 해당 명칭은 그리드 타이틀 영역(UDC)에 보여지는 명칭 및 그리드 데이터를 엑셀로 Export시 엑셀 파일명에 사용된다. |
| **readOnly** | 해당 그리드를 편집 불가능한 조회용으로 지정되게 한다.  기본값: **false** |
| **selectionMulti** | 다중 ROW 선택 가능여부를 지정한다.  기본값: single |
| **selectionUnit** | 마우스로 그리드 선택시 선택 모드를 지정한다.  기본값: row  row: 행단위 선택  column: 컬럼단위 선택  cell: 개별 셋단위 선택 |
| **showDeletedRow** | 그리드에서 데이터 행을 삭제할 경우에, 삭제된 행을 사용자에게 보여줄지 여부를 지정한다.  기본값: **true**  \*\* 해당값이 false인 경우에 삭제된 행은 그리드상에서 보여지지 않고 숨겨진 형태로 간직된다. |
| <사용자 정의속성>  bindDataFormId | 그리드-프리폼 화면에서 해당 그리드에 매핑되는 프리폼ID를 지정한다. 이는 프리폼에 대한 값 변경시(삭제 등)에 그리드를 Refesh및 PK 컬럼에 대한 활성/비활성 처리 등에 사용된다. |
| <사용자 정의속성>  columnMoveFix | 그리드의 컬럼은 기본적으로 이동 가능하며, 특정한 경우에 한하여 컬럼 이동을 막을 경우에 ‘Y’로 설정한다. |
| <사용자 정의속성>  columnResizeFix | 그리드의 컬럼은 기본적으로 사용자가 늘리거나/줄일 수 있으며, 특정한 경우에 한하여 컬럼 Resize를 막을 경우에 ‘Y’로 설정한다. |
| <사용자 정의속성>  columnSortFix | 그리드의 컬럼은 기본적으로 클릭시 데이터 정렬이 가능하며, 특정한 경우에 한하여 헤더 클릭시 정렬을 막을 경우에 ‘Y’로 설정한다. |

■ 그리드 헤더(Header) 컬럼 설정

그리드 내의 헤더 컬럼들에 대한 기본 설정에 대해 기술한다.

<컬럼 크기 및 리사이징 관련>

|  |  |
| --- | --- |
| 속성 | 내용 |
| autoFit | 그리드 표시공간에 여유가 있는 경우 자동으로 해당 컬럼 사이즈를 확장할지 여부  false: 고정 사이즈 |
| width | 컬럼 사이즈(OOpx 형태로 지정)  반드시 px로 지정해야 됨 |

|  |  |
| --- | --- |
| 속성 | 내용 |
| **columnType** | 헤더 컬럼의 유형을 지정한다.  기본값: normal  \*\* 그리드 행 전체 선택용 체크박스인 경우에는 해당 속성을 checkbox로 지정한다. |
| **targetColumnName** | 헤더 클릭시 소트(Sort)할 데이터셋 컬럼명칭을 지정한다.  \*\*지정되어 있지 않은경우, 공통 그리드 초기화시 자동 지정됨 |
| **text** | 헤더 컬럼의 명칭을 지정한다.  \*\* 만약 헤더명에 다국어를 지정해야 되는 경우에는 바인딩탭에서 text 속성에 다국어를 지정한다.  \*\* 만약 헤더명에 표현식(Expression)을 사용해 동적으로 명칭을 변경하고자 하는 경우에도 바인딩탭에서 text 속성에 표현식을 지정하면 된다. |
| **visible** | 해당 컬럼을 숨길지 여부를 지정한다.  기본값: true |

■ 그리드 디테일(Detail) 컬럼 설정

그리드 내의 디테일 컬럼들에 대한 기본 설정에 대해 기술한다.

디테일에는 입력용 InputBox, ComboBox 등의 컨트롤을 추가하여 사용한다. 따라서, 기본적인 그리드 셀 속성외의 속성값들은 각 컨트롤 유형의 속성(format, maxLength, enable, readOnly, required 등등)에서 지정하면 된다.

[컨트롤 추가 규칙]

그리드의 랜더링 속도 향상을 위하여, 편집불가한(ReadOnly)한 데이터는 기본적으로 추가적인 컨트롤을 사용하지 않으며, 편집 가능한 컨트롤에 한하여 각 데이터 유형에 맞게 InputBox, ComboBox 등의 컨트롤을 추가하여 사용한다.

|  |  |
| --- | --- |
| 속성 | 내용 |
| **columnName** | 해당 컬럼에 바인딩되는 데이터셋의 노드명을 지정한다.  Ex) CD\_NM, PGM\_NM |
| **columnType** | 컬럼의 유형을 지정한다.  기본값: normal  rowindex: 그리드 행의 인덱스를 표시하고자 하는 경우에 사용  checkbox: 그리드 행 선택용 체크박스 사용시 지정 |
| **suppressible** | 바로 직전 로우의 값과 일치하는 셀은 텍스트를 숨기도록 할지 여부를 설정한다.  기본값: false |
| **suppressRef** | suppressible(셀 텍스트 숨김) 효과에 영향을 줄 다른 컬럼 인덱스를 설정한다. 예를들어 B컬럼이 앞 컬럼인 A컬럼의 값에 따라 suppress를 하고자 하는 경우에 A컬럼의 인덱스를 지정하면 된다.  \*\* 디테일의 컬럼 인덱스는 0부터 시작한다. |

* 1. 그리드 디테일 컬럼 설정

그리드의 디테일 영역에서는 바인딩탭에서 속성 및 스타일들에 대한 추가적인 작업을 할 수 있으며, 더 복잡한 데이터 핸들링을 위해 기본 컨트롤을 추가하여 사용할 수 있다.

① 그리드를 더블클릭하여 편집창 모드로 들어간다.

② 팔렛트뷰에서 디테일행의 컬럼에 데이터 형식에 맞게 컨트롤을 추가한다.

디테일 컬럼의 기본 컨트롤 유형은 Output이다.



* 단순 읽기 전용 컬럼에는 추가적인 컨트롤을 사용하지 않는다.(디폴트 Output)

(단, 금액을 표시하고자 하는 경우에는 별도의 Output 컨트롤을 추가하여 포맷을 지정하여 사용한다.)

* 텍스트 입력 가능한 컬럼에는 InputBox 컨트롤을 추가하며, 해당 데이터가 저장되는 DB의 컬럼사이즈에 맞게 maxLength값을 지정한다.
* 코드값에서 선택하는 컬럼에는 ComboBox 컨트롤을 추가하며, 해당 ComboBox에 보여질 데이터셋을 콤보박스에 바인딩한다.
* 금액을 입력하는 컬럼에는 NumberEditor 컨트롤을 추가하며, Spin버튼 숨김처리 후 포맷을 지정하여 사용한다. (단, 필요한 경우에 한하여 Spin버튼을 보이게 처리할 수도 있다.)
* NumberEditor를 통한 금액 표시는 천단위 콤마(,)로 표시되게끔 포맷을 지정한다. 자세한 사항은 개발가이드의 마스크/포맷처리 부분을 참고한다.
* 날짜를 입력하는 컬럼에는 DateInput 컨트롤을 추가하여 사용한다.
* 기타 데이터 타입에 적절한 입력컨트롤을 추가하여 사용한다.

■ 바인딩 표현식

디테일 컬럼에서도 컨트롤을 추가하여, 디테일에 보여지는 데이터를 가공할 수 있다.

아래는 추가적으로 Output 컨트롤을 이용해 표현식(Expression)으로 데이터값을 가공하여 보여주는 방법을 예시로 보여준다.

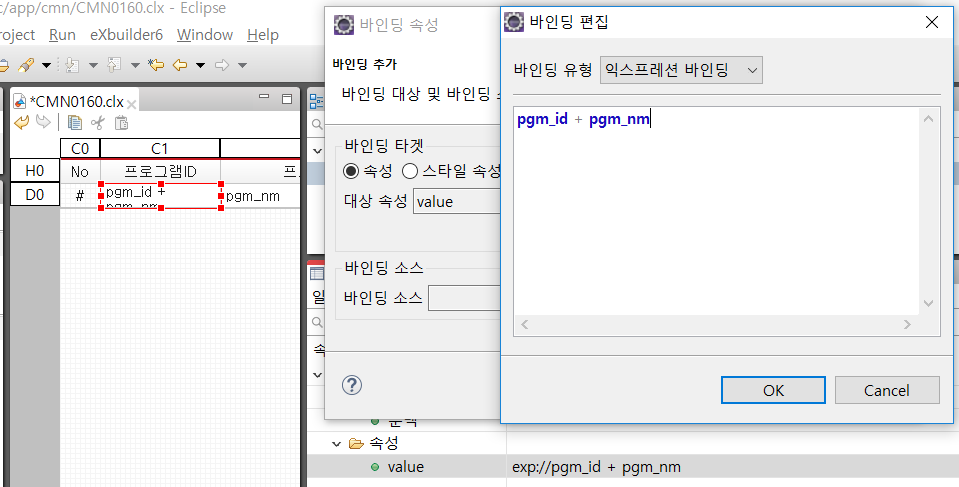
① Output 컨트롤을 추가한다.

② 추가한 컨트롤을 선택후, 바인딩탭에서 [+]버튼을 클릭한다.

③ 바인딩 속성창에서 대상속성을 value로 선택하고 바인딩 소스를 클릭한다.

④ 바인딩 편집창에서 유형을 익스프레션 바인딩으로 선택한 후, 표현식을 작성한다.

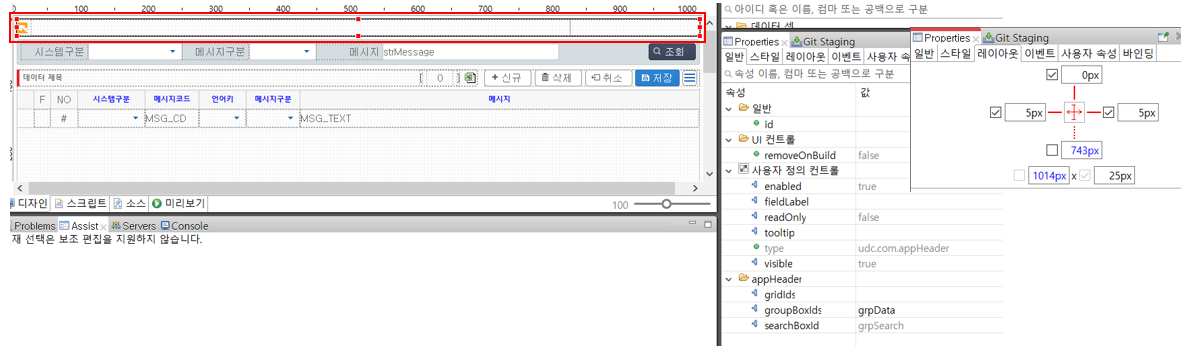
(표현식에서는 데이터셋 컬럼뿐만 아니라, 해당 화면내의 컨트롤의 값을 ID참조로 얻어올 수도 있으며, getSum(), getRowCount() 등과 같은 그룹함수도 사용 가능하다.)



* 1. 헤더 컴포넌트 임포트

각 화면별 사용자 권한 및 컨트롤 기능 활성화를 위해 헤더 UDC(udc.com.appHeader)를 화면 상단에 추가한다.

① 화면 상단에 udc.com.appHeader 컨트롤을 추가한다.

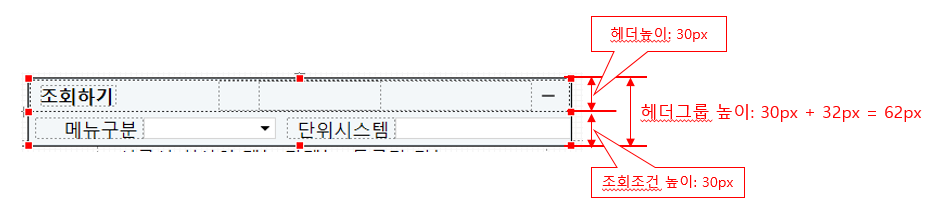


② 화면 컨트롤을 위한 추가적인 정보를 지정한다.

|  |  |
| --- | --- |
| 속성 | 내용 |
| gridIds | 그리드에 공통적인 기능을 추가하기 위한 초기화 작업시, 대상 그리드 컨트롤ID 목록을 지정한다.  만약, 해당 값이 지정되어 있지 않은 경우에는 화면 내의 모든 그리드가 자동으로 대상이 된다. |
| searchBoxId | 조회조건 변경시 화면 Clear 매커니즘을 사용하는 경우, 조회조건 영역 그룹 컨트롤 ID값을 지정한다.  화면내에 조회조건이 여러 개인 경우에는 콤마(,) 구분자로 조회조건 그룹ID를 여러 개 지정한다.  \*\* 만약, 해당 값이 지정되어 있지 않은 경우에는 조회조건 그룹 컨트롤의 아이디가 grpSearch가 대상이 된다. |
| groupBoxIds | 조회조건 변경시 화면 Clear 매커니즘을 사용하는 경우, 조회조건 변경시 Clear 대상 데이터 영역 그룹 컨트롤ID값 또는 탭 컨트롤ID값을 지정한다.  \*\* 만약, 해당 값이 지정되어 있지 않은 경우에는 데이터 그룹 컨트롤의 아이디가 grpData가 대상이 된다. |
| expandCtrlIds | 조회조건이 접혔을 때, 기본적으로는 해당 화면내의 최상위 데이터 그룹을 자동으로 Top을 변경하여 리사이징 처리해준다.  하지만, 화면이 특이하거나 복잡한 경우 자동으로 리사이징 처리가 안되는 경우에는 리사이징되어야 하는 대상 데이터 그룹 컨트롤ID를 지정한다. |

* 1. 조회조건 접기/펼치기

통합정보시스템 UI화면내의 조회조건은 기본적으로 접기/펼치기 기능이 들어가 있다. 이러한 구현을 위해서 헤더 영역과 조회조건 영역을 아래와 같이 하나의 헤더그룹(Group)으로 감싸 처리한다.



① 어플리케이션 헤더와 조회조건을 하나의 그룹 컨트롤로 감싼다.

② 헤더 그룹 컨트롤의 ID는 grpSearchAra로 한다.

③ 헤더 그룹(Group)은 Vertical-Layout으로 지정한다. (spacing: 0, scrollable: false)

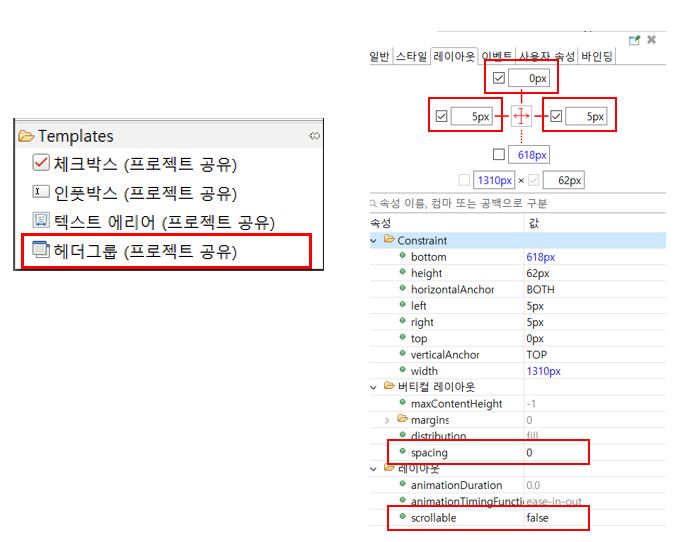
④ 헤더 그룹(Group)의 높이는 헤더 높이 + 조회조건 높이로 맞춘다.

⑤ 조회조건이 없는 경우에도 헤더를 헤더그룹(Group) 컨트롤로 감싼다.

■ 작업순서

- 헤더와 조회조건을 그룹(Group)으로 감싼다.

- 헤더 그룹은 이클립스 팔렛트의 Templates에서 헤더그룹(프로젝트 공유) 템플릿을 사용하여 추가한다.



- 헤더 그룹(Group) 안에 헤더 UDC와 조회조건을 넣는다.

* 1. 공통 작업버튼 임포트

화면상에 데이터 추가/삭제/저장 및 엑셀출력 등과 같은 기능을 처리하기 위해서는 각 UI화면에 공통 작업버튼 UDC(사용자 정의 컨트롤)을 추가한다.

<공통 작업버튼>

|  |
| --- |
|  |

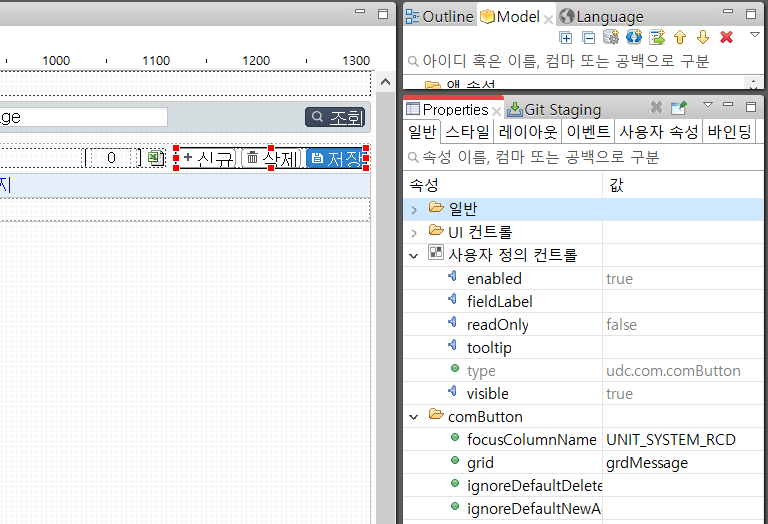
해당 작업버튼 UDC 컨트롤에는 각 기능 버튼들이 클릭되었을 때 발생하는 이벤트들을 각 화면에서 추가하여야 한다.

해당 버튼들은 사용자의 권한에 따라, 각 버튼이 활성화 또는 비활성화 된다.

|  |  |  |
| --- | --- | --- |
| 기능 버튼 | 이벤트명 | 비고 |
| 신규 | insert | 그리드에 신규 행을 추가한다. |
| 삭제 | delete | 그리드에서 선택된 행을 삭제한다. |
| 저장 | save | 그리드의 변경사항을 저장한다. |
| 신규전 이벤트 | beforeInsert | 신규버튼 클릭전 호출 이벤트(신규 행 추가전에 체크할 비지니스 로직이 있는 경우 사용)  \*\* false를 반환하는 경우 신규 행 추가로직 동작 안함 |
| 삭제전 이벤트 | beforeDelete | 삭제버튼 클릭전 호출 이벤트(행 삭제전에 체크할 비지니스 로직이 있는 경우 사용)  \*\* false를 반환하는 경우 행 삭제로직 동작 안함 |

■ 적용방법

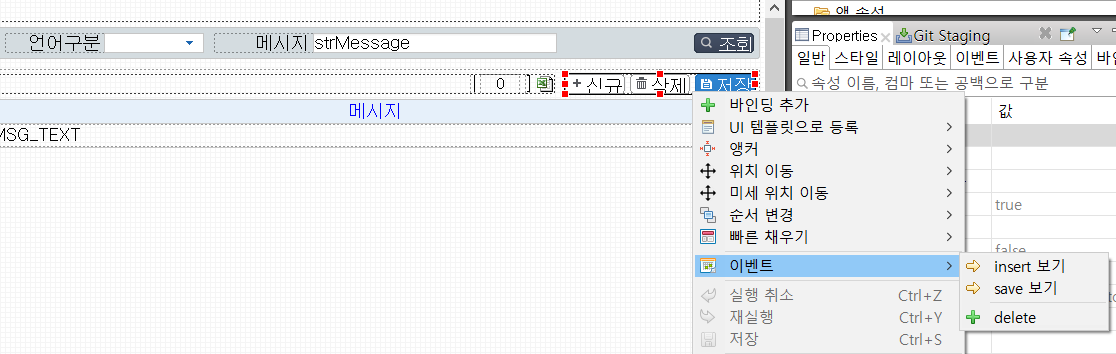
① 디자이너의 팔렛트 뷰의 UDC 패널에서 udc.com.comButton 컨트롤을 화면에 추가한다.



|  |  |
| --- | --- |
| 속성 | 내용 |
| **grid** | 신규/삭제/저장 대상 그리드(Grid) 컨트롤ID 또는 프리폼 그룹(Group) 컨트롤ID를 지정한다. |
| **focusColumnName** | [신규] 버튼 클릭하여 행추가 후, 포커싱할 컬럼 노드명을 지정한다.  Ex) CD |
| ignoreDefaultNewAction | [신규] 버튼 클릭시 디폴트로 대상 그리드(Grid) 컨트롤에 1행을 자동으로 추가한다.  해당 값이 true인 경우에는 디폴트로 행을 추가하지 않는다. 이는 행 추가전에 개별적으로 유효성 체크 및 추가적인 검증 액션을 취해야 하는 경우를 위한 옵션이며, 이와 같은 경우에는 해당 화면에서 직접 대상 그리드에 행을 추가하는 로직을 직접 구현해야 한다. |
| ignoreDefaultDeleteAction | [삭제] 버튼 클릭시 디폴트로 대상 그리드(Grid) 컨트롤에서 체크/선택된 행을 삭제한다.  해당 값이 true인 경우에는 디폴트로 액션을 수행하지 않는다. 이는 행 삭제전에 개별적으로 유효성 체크 및 추가적인 검증 액션을 취해야 하는 경우를 위한 옵션이며, 이와 같은 경우에는 해당 화면에서 직접 대상 그리드에 선택된 행을 삭제하는 로직을 직접 구현해야 한다. |
| forceDelete | [삭제] 버튼 스타일을 변경한다.  [삭제] 버튼 클릭시, 곧바로 서버에 요청하여 삭제 처리를 하는 경우에 지정한다. |
| buttonNewLabel | 신규 버튼 라벨 명칭 |
| buttonDelLabel | 삭제 버튼 라벨 명칭 |
| buttonSaveLabel | 저장 버튼 라벨 명칭 |
| visibleSaveButton | 저장버튼 보이기 여부(true/false) |
| enableMultiForm | 프리폼 다건 신규/삭제 가능여부(true/false) |
| insertMasterPkColumns | 신규시 마스터 데이터셋의 PK 컬럼 자동 추가여부(true/false) |

② 신규/삭제/저장 버튼 클릭시 추가적으로 구현해야 하는 로직이 있는 경우, 해당 컨트롤에서 출판한 이벤트를 추가하여 사용한다.(신규, 삭제, 버튼의 경우에는 디폴트 액션이 구현되어 있으며, 저장 버튼의 경우에는 반드시 각 화면단에서 재정의하여 구현해야 한다.)

③ 이벤트 메뉴에서 추가하고자 하는 이벤트를 선택하여 추가한다.



* 1. onBodyLoad() 이벤트 선언

화면이 로딩된 이후에 처리할 작업들을 기술하기 위하여, onBodyLoad() 이벤트를 추가한다.

onBodyLoad() 이벤트 내에서 처리할 작업들은 업무별 로직에 따라 다르겠지만 기본적으로 아래와 같다.

① 화면내 공통코드가 필요한 경우, 공통코드 조회하기 위한 서브미션 호출

|  |
| --- |
| /\*  \* Body에서 load 이벤트 발생 시 호출.  \* 앱이 최초 구성된후 최초 랜더링 직후에 발생하는 이벤트 입니다.  \*/  function onBodyLoad(/\* cpr.events.CEvent \*/ e){  //코드조회 서브미션 호출  util.Submit.send(app, "subOnload", null, function(pbSuccess){  if(pbSuccess) {  util.SelectCtl.selectItem(app, "cmbUnitSystemRcd", 0);  util.SelectCtl.selectItem(app, "cmbMsgDivRcd", 0);  }  });  } |

* 1. 조회 이벤트 선언

데이터 조회를 위해 조회조건 영역내의 [조회] 버튼 클릭시의 이벤트를 선언한다.

[조회] 버튼을 클릭했을 때의 작업순서는 아래와 같다.

① 요청 파라메터 셋팅

② 조회조건 유효성 체크(필수 입력값 체크 및 조건간 유효성 체크)

③ 서브미션 호출

④ 조회된 데이터를 보여주기 위하여, 그리드 또는 폼 redraw 및 후처리

|  |
| --- |
| /\*  \* "조회" 버튼에서 click 이벤트 발생 시 호출.  \* 사용자가 컨트롤을 클릭할 때 발생하는 이벤트.  \*/  function onBtnSearchClick(/\* cpr.events.CMouseEvent \*/ e){  //1. 데이터 변경사항 체크  if (util.Grid.isModified(app, "grdMessage", "CRM")) {  return false;  }    //2. 조회조건 유효성 체크(필수입력 등)  if(!util.validate("grpSearch")) return false;  //3. 데이터 조회  doList();  };  function doList(psStaus) {  //서브미션 파라메터 셋팅 및 서브미션 호출  util.Submit.send(app, "subList", null, function(pbSuccess){  if(pbSuccess) {  if(psStatus == "save"){  //갱신된 데이터가 조회되었습니다.  util.Msg.notify("INF-M005");  }else{  //조회되었습니다.  util.Msg.notify("INF-M001");  }  }  });  }; |

* 1. 그리드 ROW 신규

데이터 그리드에 신규 아이템을 추가하기 위해서 [신규]버튼을 클릭한 경우에 공통 유틸 클래스를 통하여 신규 ROW를 추가한다. 공통 작업버튼을 사용하는 경우 [신규] 버튼 클릭시 디폴트로 지정된 대상 그리드에 자동으로 1행을 추가해준다. 따라서, 디폴트 행추가 이후에 추가적으로 수행해야하는 로직이 없는 경우에는 insert 이벤트를 개별 화면단에서 재정의하여 사용할 필요는 없다.

단, 행 추가 이후에 추가된 행에 대한 디폴트 값을 셋팅해주는 등의 로직을 수행하여야 하는 경우에는 공통 작업버튼(comButton)의 insert 이벤트를 재정의한다.

|  |
| --- |
| /\*  \* 사용자 정의 컨트롤에서 insert 이벤트 발생 시 호출.  \*/  function onComButtonInsert(/\* cpr.event.CUIEvent \*/ e){  // 조회조건에 셋팅된 내역이 있으면 신규시 자동셋팅한다.(시스템구분, 언어키, 메시지구분)  util.Grid.setCellValue("grdMessage", "UNIT\_SYSTEM\_RCD", app.lookup("cmbUnitSystemRcd").value, e.rowIndex);// 시스템구분  util.Grid.setCellValue("grdMessage", "MSG\_DIV\_RCD", app.lookup("cmbMsgDivRcd").value, e.rowIndex);// 메세지구분  }; |

* 1. 그리드 ROW 삭제

[삭제] 버튼 클릭시, 선택된 ROW에 대하여 삭제처리를 위해 공통 유틸클래스를 호출한다.

삭제 로직에 대한 작업순서는 아래와 같다. 그리드에 대한 아이템 삭제는 화면상에서만 삭제된 ROW라는 것을 표시만 할 뿐이며, 실제 삭제 작업은 [저장] 버튼을 클릭했을 때 서브미션을 통해 실제 DB에서 삭제된다.

공통 작업버튼을 사용하는 경우 [삭제] 버튼 클릭시 디폴트로 지정된 대상 그리드의 선택된 행을 삭제 상태로 변경한다. 따라서, 디폴트 행 삭제 전 또는 삭제 이후에 추가적으로 수행해야하는 로직이 없는 경우에는 delete 이벤트를 개별 화면단에서 재정의하여 사용할 필요는 없다.

① 삭제할 선택된 데이터가 있는지 체크한다.

② 삭제할 데이터가 없으면, ‘삭제할 데이터가 없습니다.’ 메시지를 보여준다.

③ 삭제를 수행한다.

|  |
| --- |
| /\*  \* 사용자 정의 컨트롤에서 Delete 이벤트 발생 시 호출.  \*/  function onComButtonDelete(/\* cpr.event.CUIEvent \*/ e){  //util.Grid.deleteRow(app, "grdMessage");  }; |

* 1. CRUD 작업저장 처리

화면상에서 사용자 액션에 의해 추가/변경/삭제된 데이터를 서버에 저장하기 위한 작업을 처리한다.

기본적인 데이터 저장 작업순서는 아래와 같다.

① 저장할 데이터가 있는지 확인하기 위하여, 그리드 또는 폼컨트롤의 바인딩된 데이터셋의 변경사항을 체크한다.

② 저장할 데이터에 대한 필수값 및 해당 값의 유효성을 체크한다.

③ 저장 서브미션을 호출한다.

④ 저장한 데이터를 재조회한다.

⑤ ‘갱신된 데이터가 조회되었습니다.’ 메시지를 보여준다.

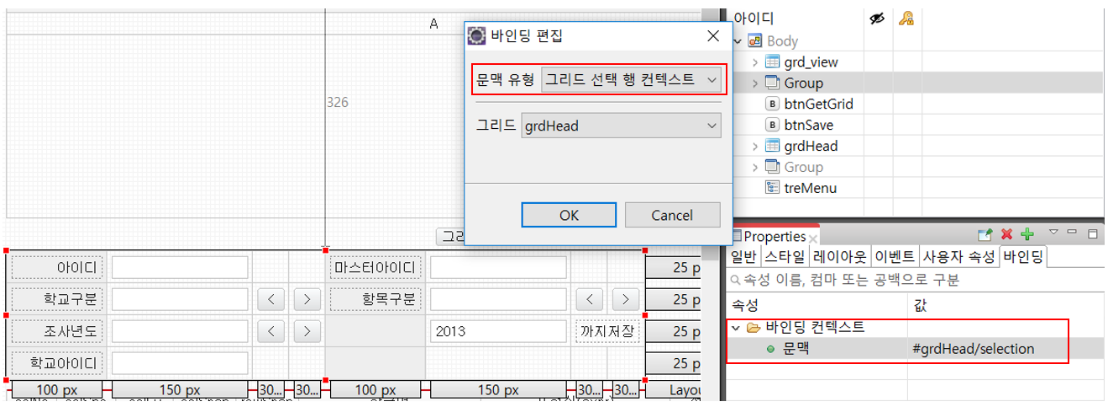
|  |
| --- |
| /\*  \* 사용자 정의 컨트롤에서 save 이벤트 발생 시 호출.  \*/  function onGrid\_IDS\_BtnSave(/\* cpr.event.CUIEvent \*/ e){  doSave();  };  function doSave(){  //1. 변경사항 체크  if(!util.Grid.isModified("grxMst", "MSG")) return;    //2. 그리드에서 필수값 체크 등 유효성 체크 수행  if(!util.validate("grxMst")) return;    //3. 서브미션 파라메터 셋팅 및 서브미션 호출  util.Submit.send(app, "subSave", null, function(pbSuccess){  if(pbSuccess) {  //4. 데이터 재조회  doList("save");  }  });  } |

* 1. 프리폼 데이터 바인딩

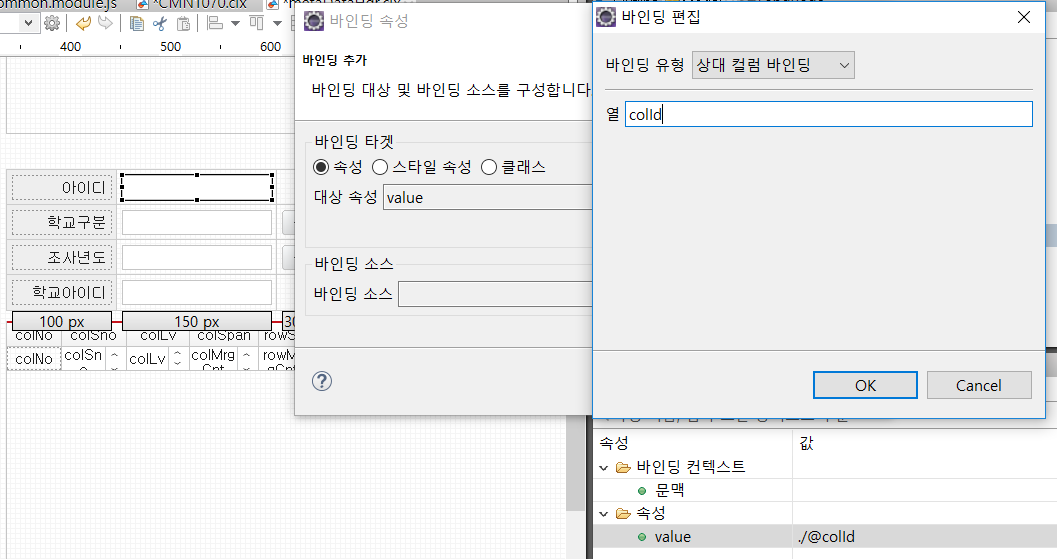
그리드에서 선택된 행의 데이터를 프리폼에 바인딩하는 방법은 아래와 같다.

① 그룹 컨트롤(프리폼)을 마우스로 선택 후, 바인딩탭에서 [+]버튼을 클릭하여 바인딩 컨텍스트 문맥 데이터를 지정한다.

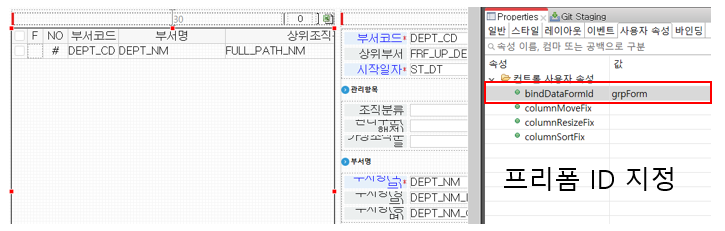
② 바인딩 편집창에서 문맥유형을 ‘그리드 선택 행 컨텍스트’로 선택하고, 하단에 콤보에서 그리드를 선택한다.



③ 프리폼 내의 각 컨트롤을 선택한 후, 바인딩탭에서 [+] 버튼을 클릭하여 해당 컨트롤의 value값에 데이터셋의 상대컬럼 바인딩을 지정한다. 해당값은 바인딩되는 데이터셋을 지정 후 마우스 드래그앤 드랍으로 컨트롤에 value값을 매핑할 수 있다.



④ 그리드 행에 바인딩되는 프리폼인 경우에는 반드시, 관련된 해당 그리드 컨트롤의 사용자 정의 속성인 bindDataFormId에 프리폼 그룹 컨트롤의 ID를 지정해주어야 한다. 바인딩되는 프리폼이 여러 개인 경우에는 콤마(,)로 구분해서 입력한다. 이는 프리폼의 PK 컬럼 항목 활성/비활성 자동 처리 등의 추가적인 공통처리를 하는데 사용된다.



* 1. 프리폼 CRUD 처리

프리폼의 경우에는 단건 저장을 기본으로 한다.

* 신규/삭제/저장 버튼은 공통 UDC 작업 버튼을 사용한다.
* 삭제시 곧바로 해당 데이터를 삭제하는 서브미션을 호출한다.
* 공통 UDC의 forceDelete 속성을 ‘true’로 지정하여 [삭제] 버튼의 스타일을 변경한다.

■ 프리폼 신규

기본적인 프리폼 신규 작업순서는 아래와 같다.

① 프리폼에 변경사항이 있는지 유/무를 체크한다.

② 프리폼 데이터를 모두 Reset 처리한다.

③ 프리폼에 바인딩된 데이터셋에 신규 행을 추가한다.

공통버튼 UDC를 통해 신규행 추가를 수행한다. 만약, 신규 후에 디폴트 처리를 해야하는 경우에는 insert 콜백함수에서 처리를 수행한다.

|  |
| --- |
| /\*  \* 사용자 정의 컨트롤에서 insert 이벤트 발생 시 호출.  \*/  function ***onComButtonInsert***(/\* cpr.event.CUIEvent \*/ e){  // 조회조건에 셋팅된 내역이 있으면 신규시 자동셋팅한다.  util.FreeForm.setValue(app, "grpForm", "UNIT\_SYS\_RCD", app.lookup("cmbUnitSystemRcd").value);// 시스템구분  util.FreeForm.setValue(app, "grpForm", "PGM\_DIV\_RCD", "CMN059.0004");// 프로그램구분  util.FreeForm.setValue(app, "grpForm", "USE\_YN", "Y");//사용여부  } |

■ 프리폼 삭제

기본적인 프리폼 삭제 작업순서는 아래와 같다.

① 삭제할 데이터가 있는지 체크하며, 데이터가 없으면 ‘삭제할 데이터가 없습니다.’ 메시지를 보여준다.

② ‘삭제하시겠습니까?’ 메시지를 보여준다.

③ 신규 행을 삭제하는 경우에는 데이터를 Reset시키고, 신규행이 아니면 데이터를 삭제 상태로 변경한다.

④ 삭제된 데이터를 실제 삭제하기 위한 저장 서브미션을 호출한다.

|  |
| --- |
| **<예시 1>**  /\*  \* 사용자 정의 컨트롤에서 delete 이벤트 발생 시 호출.  \*/  function ***onComButtonDelete***(/\* cpr.event.CUIEvent \*/ e){  //삭제처리  doSave();  }  **<예시 2>**  --------- 직접 프리폼 데이터 삭제시(삭제 전 별도 체크로직 수행 등) ----------  function ***onComButtonDelete***(/\* cpr.event.CUIEvent \*/ e){  var vsActnStsRcd = util.FreeForm.getValue(app, "grpForm", "ACTN\_STAT\_RCD");//조치상태코드  var vsActnCntn = util.FreeForm.getValue(app, "grpForm", "ACTN\_CTNT");//조치내용  //조치 상태값 및 조치내용 체크  if((!ValueUtil.isNull(vsActnStsRcd) && vsActnStsRcd != "CMN302.0001") || !ValueUtil.isNull(vsActnCntn)){  //조치중 또는 조치완료된 건은 삭제가 불가합니다.  util.Msg.alert("INF-M018");  return false;  }    //삭제 후 저장 처리  if(util.FreeForm.deleteRow(app, "grpForm", "CRM")){  //저장  doSave();  }  } |

■ 프리폼 저장

기본적인 데이터 저장 작업순서는 아래와 같다.

① 저장할 데이터가 있는지 확인하기 위하여, 프리폼에 바인딩된 데이터셋의 변경사항을 체크한다.

② 저장할 데이터에 대한 필수값 및 해당 값의 유효성을 체크한다.

③ 저장 서브미션을 호출한다.

④ 저장한 데이터를 재조회한다.

⑤ ‘갱신된 데이터가 조회되었습니다.’ 메시지를 보여준다.

|  |
| --- |
| /\*  \* 사용자 정의 컨트롤에서 save 이벤트 발생 시 호출.  \*/  function ***onComButtonSave***(/\* cpr.event.CUIEvent \*/ e){  doSave();  }  /\*\*  \* 데이터를 저장한다.  \*/  function doSave(){  // 프리폼의 변경사항 유/무를 반환  if (!util.FreeForm.isModified(app, "grpForm", "MSG")) return false;    // 프리폼 유효성 검증  if(!util.validate(app, "grpForm")) return false;    //저장 서브미션 호출  util.Submit.send(app, "subSave", null, function(pbSuccess){  if(pbSuccess) {  doList("save");  }  });  } |

* 1. 공통 서브미션 호출 서비스

데이터 조회용(조회용 팝업 포함) 화면 및 출력(보고서) 화면의 경우에 한하여, 별도의 서버단 코딩 없이 UI화면과 쿼리(SQL)만으로 데이터를 조회하는 서비스를 사용한다.

단, 비즈니스 로직이 들어가야 하는 경우에는 조회용 화면이더라도 반드시 서버단 컨트롤러 (Controller)를 작성해야 한다.

파일 유형이 S(조회), R(출력), P(팝업)인 경우에 적용한다. 3개의 서비스는 모두 한번에 선언 가능하다.

예시) 공통코드 및 날짜 가져오기

|  |
| --- |
| var voParams = [["dsCampusRcd", "CMN001", "", "Y", "Y"],  ["dsOrgClsRcd", "CMN005", "", "Y", "Y"],  {"stime":["dmTime", "strToday", "YYYY-MM-DD"]}];  util.Submit.send(app, "subOnload\_tran", voParams, function(pbSuccess){  if(pbSuccess){  util.SelectCtl.selectItem(app, "cmbUnitSystemRcd", 0);  }  }); |

■ 공통코드 호출

화면에 필요한 공통코드의 경우에는 아래와 같이 사용할 수 있다. 여러 공통코드를 조회하는 경우에는 요청 파라메터를 배열로 선언하면 된다.

|  |
| --- |
| //초기 파라메터 셋팅  var voParams = [["dsUnitSystemRcd", "CMN003", "", "", ""]];    //공통코드 조회  util.Submit.send(app, "subOnload\_tran", voParams, function(pbSuccess){  if(pbSuccess){  util.SelectCtl.selectItem(app, "cmbUnitSystemRcd", 0);  }  }); |

* 첫번째 인자: 응답받은 공통코드를 받을 데이터셋 ID
* 두번째 인자: 조회할 공통코드 구분(CD\_CLS)
* 세번째 인자: 조회할 코드(CD)
* 네번째 인자: 사용여부 (Y/)
* 다섯번째 인자: 권한코드 여부

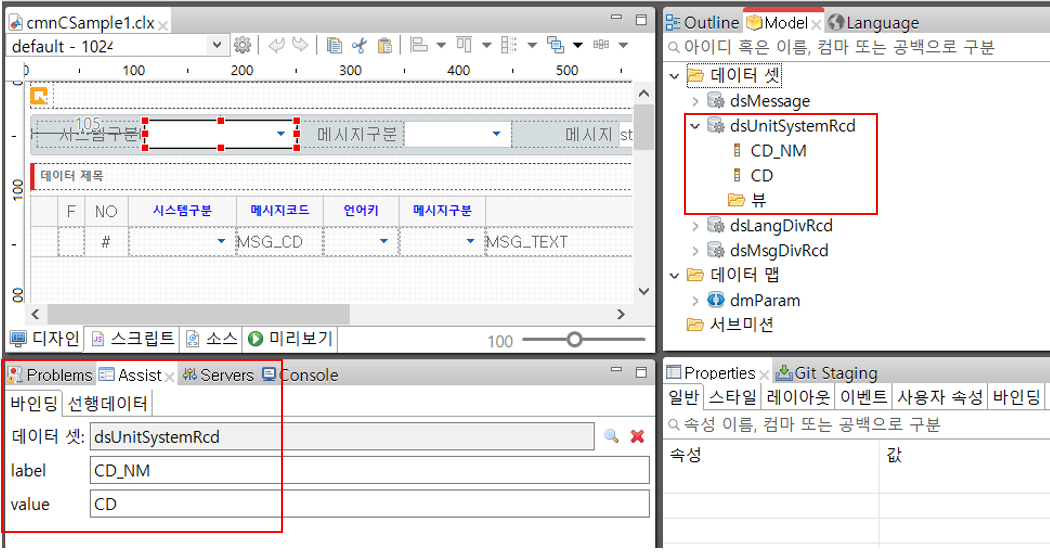
■ 현재일자 호출

화면에 필요한 현재일자의 경우에는 아래와 같이 사용할 수 있다

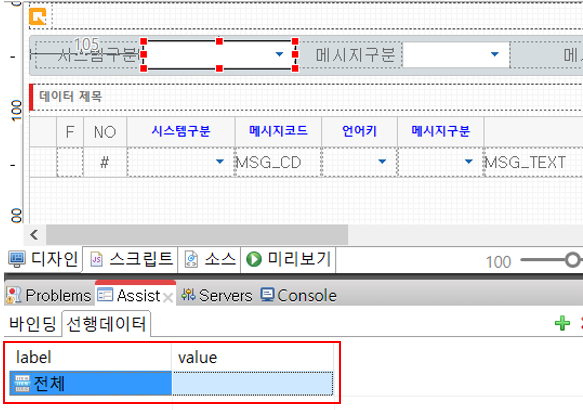
|  |
| --- |
| //초기 파라메터 셋팅  var voParams = [{"stime":["dmTime", "strToday", "YYYY-MM-DD"]}];    //공통코드 조회  util.Submit.send(app, "subOnload\_tran", voParams, function(pbSuccess){  if(pbSuccess){  util.SelectCtl.selectItem(app, "cmbUnitSystemRcd", 0);  }  }); |

* 구분자: stime
* 배열 첫번째 인자: 응답받을 데이터를 저장할 데이터맵 명
* 배열 두번재 인자: 응답 데이터 노드명
* 배열 세번째 인자: 날짜 포맷
  1. 콤보박스 바인딩

콤보박스의 데이터를 위해 별도의 데이터셋을 선언한다.



콤보박스 컨트롤에는 해당 컨트롤이 필수값을 갖는지 여부에 따라 Prompt-Item이라는 별도 더미(Dummy) 아이템을 설정한다. 해당 Prompt-Item은 Assist뷰의 선행데이터 탭에서 추가 가능하다.





* 필수값 아닌경우: ‘**전체**’ Prompt-Item을 갖는다.
* 필수값인 경우: ‘**선택**‘ Prompt-Item을 갖는다.
* 빈값 선택용: ‘ ‘ Prompt\_item을 갖는다.

(Ex) 리피트 콤보컬럼에서 NULL값 설정할 경우)

* 1. 콤보박스 필터링

■ 다중 콤보박스 필터링

일반적으로 다중 콤보박스에서의 필터링은 해당 콤보박스의 selection-change 이벤트 내에서 처리한다.

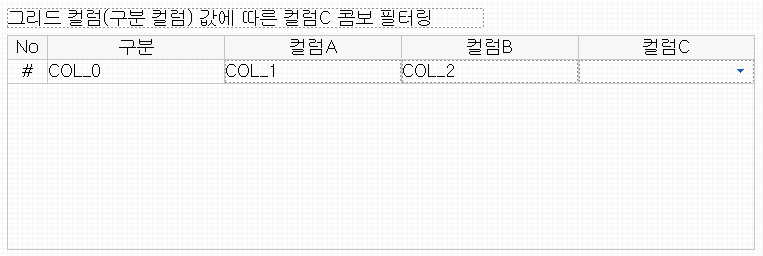
<UI 예시>



|  |
| --- |
| /\*  \* 콤보 박스에서 selection-change 이벤트 발생 시 호출.  \* ComboBox Item을 선택하여 선택된 값이 저장된 후에 발생하는 이벤트.  \*/  function ***onCmb4SelectionChange***(/\* cpr.events.CSelectionEvent \*/ e){  /\*\*  \* @type cpr.controls.ComboBox  \*/  var vcCombo = e.control;    var vcTargetCombo = app.lookup("cmb5");  //선택된 값  var vsNewVal = e.newSelection[0].value;  //필터링 수행  if(vsNewVal == ""){ //'전체' 선택시  vcTargetCombo.dataSet.clearFilter();  }else{ //특정 아이템 선택시  vcTargetCombo.dataSet.setFilter("NAT =='"+vsNewVal+"'");  }    //필터링된 콤보의 첫번째 아이템 선택  util.SelectCtl.selectItem(app, "cmb5", 0);  } |

■ 그리드 다중 콤보박스 필터링

그리드 컨트롤 내에서 특정 조건에 의해 콤보박스 데이터가 필터링되어야 하는 경우에는 해당 콤보박스의 **open** 이벤트에서 필터링을 수행한다.



open 메소드에서 콤보박스 필터링

|  |
| --- |
| /\*  \* 콤보 박스에서 open 이벤트 발생 시 호출.  \* 리스트박스를 열때 발생하는 이벤트.  \*/  function ***onCmb3Open***(/\* cpr.events.CUIEvent \*/ e){  /\*\*  \* @type cpr.controls.ComboBox  \*/  var cmb3 = e.control;  var grd = app.lookup("grd2");  var row = grd.getSelectedRow();  var val = row.getValue("COL\_0");    if(val == "A"){  cmb3.setFilter("value == '' || value == '10' || value == '20'");  }else{  cmb3.setFilter("value == '' || value == '30'");  }  } |

* 1. 인풋박스 컨트롤 속성

텍스트를 입력하기 위한 용도로 InputBox 컨트롤을 사용한다. 인풋박스 컨트롤은 자주 사용되는 컨트롤로 추가적인 유용한 속성값은 아래와 같다.

|  |  |
| --- | --- |
| 속성 | 비고 |
| imeMode | IME에 대한 한영 변환설정  단, IE, Firefox 브라우저에서만 지원됨 |
| lengthUnit | 입력 문자열의 길이 체크 단위 (\*utf8로 지정 필요) |
| maxLength | 입력 가능한 문자열의 최대 길이 |
| placeholder | 설명 또는 예상된 값에 대한 문구(힌트) |
| secret | 입력방식의 패스워드 설정여부  Ex) 12345 입력시 🡪 •••••로 표시됨 |
| inputFilter | 입력에서 받은 character를 필터링하는 정규식(regexp)를 지정할 수 있다.  Ex) ^[0-9]\*$: 0-9까지의 숫자만 입력받고자 하는 경우  ^[0-9\-]\*$: 0-9 숫자와 하이픈(-) 문자만 입력받고자 하는 경우  ^[a-zA-Z]\*$: 영문 소문자/대문자만 입력받고자 하는 경우 |

<예시>

|  |
| --- |
|  |

* 1. 넘버에디터 컨트롤 속성

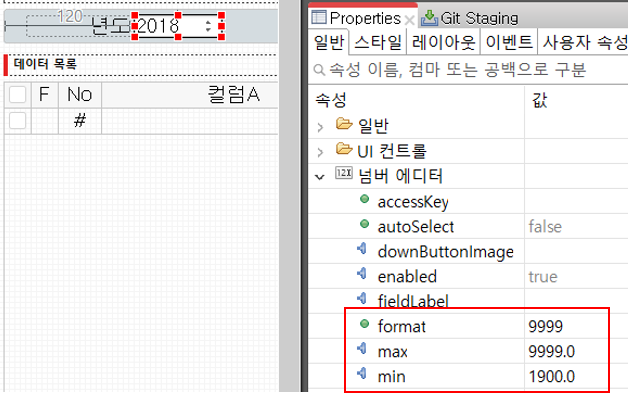
UI 화면에서 금액 등 숫자를 입력하는데 사용된다.

넘버에디터 사용시 포맷 속성 지정은 개발가이드 문서의 마스크/포맷 처리 부분을 참고한다.

■ 년도 표시용

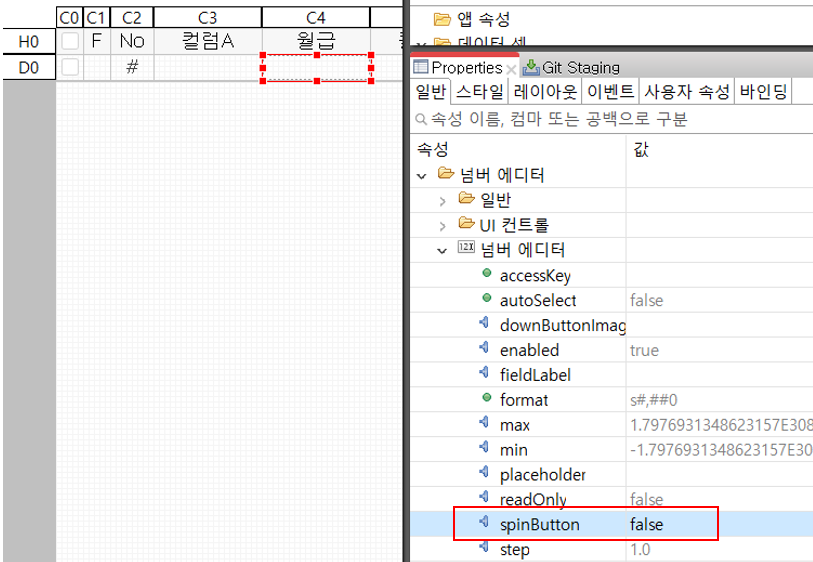
조회조건 등 영역에서 년도 표시용으로 넘버에디터 컨트롤을 사용하는 경우에는 최대/최소값을 명시적으로 지정한다.

* 4자리 숫자만 입력되도록 format 속성을 ‘9999’로 지정
* 최대 년도는 ‘9999’로 지정
* 최소 년도는 ‘1900’으로 지정



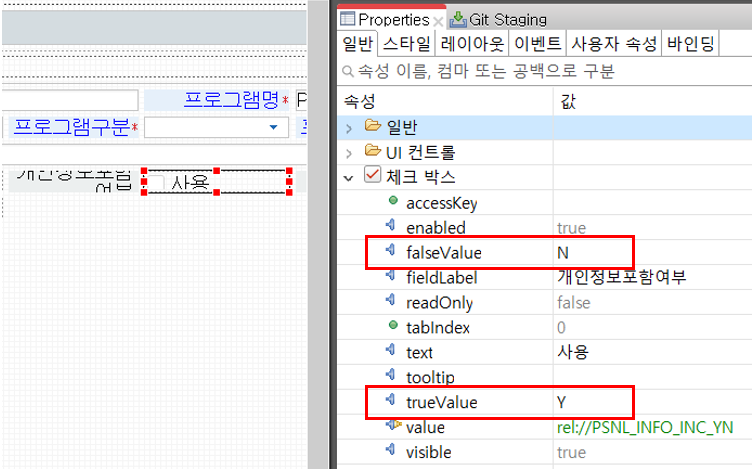
■ 그리드 컬럼에 금액 및 숫자 입력시

넘버에디터를 그리드 컬럼에 사용하는 경우에는 해당 컨트롤의 스핀버튼(spinButton)이 나타나지 않도록 속성값을 false로 지정하여 사용한다. 단, 필요한 경우에 한하여 스핀버튼을 보이도록 할 수 있다.



* 1. 체크박스 컨트롤 속성

UI 화면에서 선택 여부를 나타내는 경우에 CheckBox 컨트롤 또는 CheckBox Group을 사용한다. 프로젝트 공통적으로 체크박스 컨트롤의 선택/비선택의 값은 [Y/N]을 사용한다.

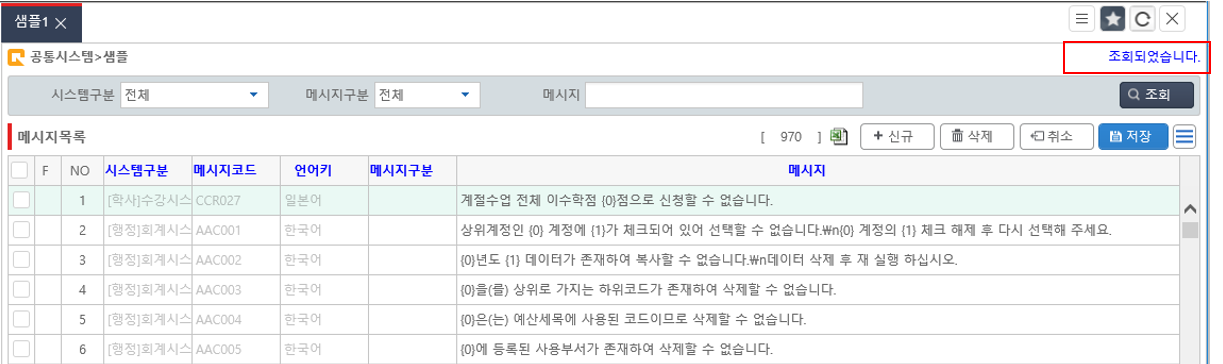


* 1. 메시지 처리 규칙

화면UI에서 업무 메시지 또는 오류 메시지를 보여주는 방법은 2가지 패턴이 존재한다.

■ 알림성 메시지

각 화면의 헤더영역에 메시지를 디스플레이 해주는 형태이다. 일반적으로 ‘저장되었습니다.’, ‘조회되었습니다’와 같은 공통적인 사용자 액션에 대한 응답 메시지를 보여주는데 사용된다.



|  |
| --- |
| function doList(psStatus){  //조회조건 유효성 체크  if(!util.validate("grpSearch")) return false;    //조회 서브미션 호출  util.Submit.send(app, "subList", null, function(pbSuccess){  if(pbSuccess){  if(psStatus == "save"){  //갱신된 데이터가 조회되었습니다.  util.Msg.**notify**(app,"INF-M005");  }else{  //조회되었습니다.  util.Msg.**notify**(app,"INF-M001");  }  }  });  } |

※ 저장 처리 메시지 관련

마스터-디테일 관계의 화면에서 마스터 데이터 저장 후에 ‘갱신된 데이터가 조회되었습니다.’ 메시지를 보여준다. 하지만, 마스터 데이터 재조회 후에 디테일 데이터가 조회됨으로써 ‘갱신된 데이터가 조회되었습니다.’ 메시지 대신 ‘조회되었습니다.’ 메시지가 보여지는 현상을 방지하기 위해서

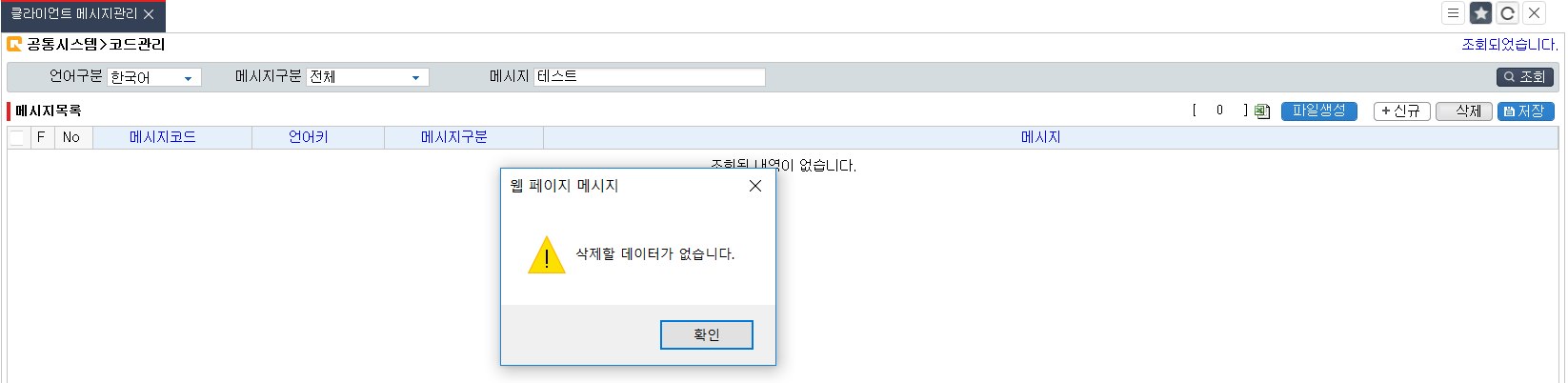
이전 메시지를 유지 시키주기 위한 옵션을 추가적으로 설정해주어야 한다.

|  |
| --- |
| //마스터 조회  //조회 서브미션 호출  util.Submit.send(app, "subList", null, function(pbSuccess){  if(pbSuccess){  //마스터 저장 후 데이터 재조회시 처리  if(psStatus == "save"){  //갱신된 데이터가 조회되었습니다.  util.Msg.**notify**(app,"INF-M005", null, true);  }else{  //조회되었습니다.  util.Msg.**notify**(app,"INF-M001");  }  }  }); |

■ 화면단 업무별 알림/경고/오류 메시지

그 외의 업무단에서 보여주고자 하는 메시지는 자바스크립트의 alert API를 사용하여 메시지 박스를 띄우는 형태이다.

화면단 메시지는 공통시스템 > 코드관리 > 클라이언트메시지관리 화면에 등록하여 사용



|  |
| --- |
| function onComButtonDelete(/\* cpr.event.CUIEvent \*/ e){  var vcGrd = app.lookup("grdMessage");  if(vcGrd.rowCount < 1){  //삭제할 데이터가 없습니다.  util.Msg.**alert**("INF-M007");  return;  }  } |

|  |  |
| --- | --- |
| 메시지 유형 | 호출 메소드 |
| util.Msg.notify | 헤더 영역에 메시지를 보여준다. |
| util.Msg.alert | 웹브라우저 Alert 메시지를 보여준다. |
| util.Msg.confirm | 예/아니오 Confirm 메시지를 보여준다. |

■ 서버단 업무별 알림/경고/오류 메시지

서버단 서비스(Service) 또는 컨트롤러(Controller)에서 업무단 로직 처리시 오류 및 경고 메시지를 반환하고자 하는 경우에는 아래와 같이 처리한다.

서버단 메시지는 공통시스템 > 코드관리 > 메시지관리 화면에 등록하여 사용

|  |
| --- |
| **try** {  **this**.loginService.login(request, principal, credential, strDefaultLocale);  } **catch**(FailedLoginException e) {  //사용자정보가 일치하지 않습니다. \n관리자에게 문의하세요.  **throw** **new** AppWorksException("CMN003.CMN@CMN005", Alert.***ERROR***);  } **catch**(Exception e) {  **throw** **new** AppWorksException("CMN003.CMN@CMN005", Alert.***ERROR***);  } |

* 메시지 구조: 단위시스템코드(10자리) + @ + 메시지코드
  1. 공통코드 조회

화면상에 필요한 공통코드 목록은 각 업무단에서 개별적으로 조회(Select)하지 않고, 제공되는 공통코드 조회 서비스(Service) API를 사용한다.

예시)

|  |
| --- |
| @RequestMapping("/onLoad.do")  **public** View onLoad(HttpServletRequest request, HttpServletResponse response, RequestData requestData) **throws** Exception {  String strLocale = requestData.getAppLocale();//로케일 정보  //단위시스템 코드  requestData.setResponse("dsUnitSystemRcd", cmnCodeService.selectCmnCodeList("CMN003", strLocale));  //언어구분 코드  requestData.setResponse("dsLangDivRcd", cmnCodeService.selectCmnCodeList("CMN006", strLocale));  //메시지구분 코드  requestData.setResponse("dsMsgDivRcd", cmnCodeService.selectCmnCodeList("CMN037", strLocale));    **return** requestData.getDefaultView();  } |

* 로케일 정보(strLocale): 다국어 처리를 위한 Locale 정보

|  |  |
| --- | --- |
| 사용 구분 | 내용 |
| 일반 코드 목록 조회시 | selectCmnCodeList, selectCmnCode 등 |
| 권한 코드 목록 조회시 | **selectCmnAuthCodeList** |

* 권한코드: 업무영역 코드로 관린되는 공통코드로, 사용자 권한에 의해 조회되어야 하는 공통코드를 의미한다.
  1. 화면 커버(Cover) 처리

업무 중에 특정 기간이 아닌 경우에 화면 처리를 못하도록 한다거나, 기타 업무 체크 로직에 따라 화면 자체를 비활성화 처리해야 되는 경우에 화면 전체에 대해 막을 씌워 사용자의 입력을 못하도록 할 수 있다.

■ 처리 방법

화면에 막을 씌우는 처리는 2가지 방법이 있다.

1) 첫째, 서브미션 호출시 5번째 인자로 (true) 옵션을 주는 경우에 서버단에서 오류 메시지(오류 또는 업무 메시지)가 리턴되는 경우에 자동으로 화면 전체에 대해 막을 씌운다.

2) onBodyLoad 이벤트 내에서 업무 로직에 따라, 공통 유틸인 util.coverPage() 메소드를 호출하여 막을 씌울 수 있다.

|  |
| --- |
| /\*  \* Body에서 load 이벤트 발생 시 호출.  \* 앱이 최초 구성된후 최초 랜더링 직후에 발생하는 이벤트 입니다.  \*/  function onBodyLoad(/\* cpr.events.CEvent \*/ e){  //공통코드 서브미션 호출  util.Submit.send(app, "subOnload", null, function(pbSuccess){  if(pbSuccess) {  util.SelectCtl.selectItem(app, "cmbUnitSystemRcd", 0);  util.SelectCtl.selectItem(app, "cmbMsgDivRcd", 0);  }  }, true);  } |

* 1. 유효성 체크

프로그램 화면의 각 항목에 들어가는 사용자 입력값들에 대한 유효성 체크 매커니즘 및 방법론을 기술한다.

■ 체크대상

1) 일반 컨트롤: 조회조건 및 프리폼 내의 컨트롤들에 대한 입력값 유효성을 체크한다.

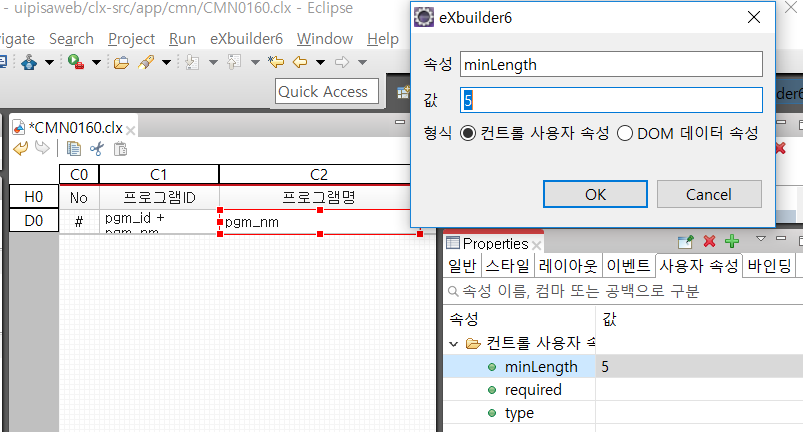
2) 그리드 컨트롤: 그리드 내의 각 컬럼 항목들에 대한 입력값 유효성을 체크한다.

■ 유효성 체크 메커니즘

1) 일반 컨트롤: 체크할 Validation의 유형은 각 Control의 사용자정의 속성(userattr) 값으로 판단한다.

2) 그리드: 체크할 Validation의 유형은 각 Grid 디테일 컬럼의 사용자정의 속성(userattr) 값으로 판단한다.

입력 컨트롤(InputBox, NumberEditor, ComboBox 등등)에는 기본적으로 디자이너 전역설정에서 설정한 required 및 columnType, compare이라는 속성을 추가되어 있다. 이 외의 속성이 필요한 경우에는 해당 컨트롤의 Properties뷰의 사용자속성탭에서 [+] 버튼을 클릭하여 추가한다.



■ 유효성 체크 목록

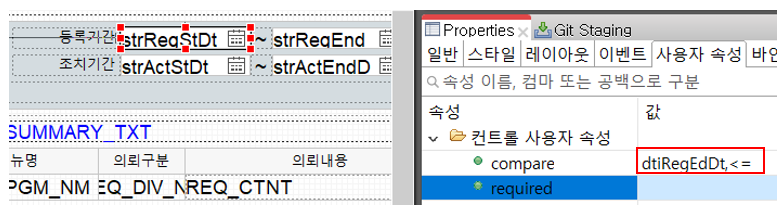
|  |  |  |
| --- | --- | --- |
| 속성명 | 적용값 | 설명 |
| required | Y | 필수 입력 체크 |
| xorRequired | 그리드일경우 columnname, 그룹 및 일반컨트롤일 경우 id | 지정된 컬럼중 하나이상 필수 입력 체크 |
| columnType | email  bizno  ssn  tel  phone  url | 이메일  사업자등록번호  주민등록번호  전화번호  핸드폰번호  URL주소 |
| minlength | Number | 최소 길이 지정 |
| fixlength | Number | 자릿수 고정 |
| compare | 비교할 대상 컨트롤 또는 컬럼명  (컨트롤ID/명+’,’+비교연산자) | ditRegEndDt,<=  ED\_DT,<=  ditRegStDt,>= |

■ 유효성 체크 스크립트 호출

데이터 조회 및 저장시 공통스크립트를 호출하여 입력 컨트롤에 대한 데이터 유효성을 체크한다.

|  |
| --- |
| <데이터 저장시>  /\*  \* 사용자 정의 컨트롤에서 save 이벤트 발생 시 호출.  \*/  function onGrid\_IDS\_BtnSave(/\* cpr.event.CUIEvent \*/ e){  //2. 그리드에서 필수값 체크 등 유효성 체크 수행  if(!**util.validate**("grxMst")) return;  }; |

■ 시작일자~ 종료일자 비교



* 등록기간 시작일자 컨트롤에 등록 종료일자 compare 사용자 속성 추가하여 유효성 체크
  1. 엑셀 데이터 업로드(Import)

일반적인 로직 체크가 없는 엑셀 업로드인 경우에는 공통에서 제공하는 엑셀 업로드 팝업을 사용할 수 있다. 하지만, 엑셀 데이터를 업로드하여 읽어들이면서 로직 체크 및 유효성 체크 등을 해야 하는 경우에는 직접 엑셀 파일을 서버에 업로드하여 처리한다.

<처리 흐름>

① 엑셀 파일을 서버에 업로드한다.

② 업로드된 엑셀을 셀단위로 읽는다.

③ 해당 데이터에 대한 업무 로직을 수행한다. (유효성 체크, 로직 체크, 데이터 저장)

④ 오류 내역이 있는 경우에는 오류 내역 정보를 사용자에게 보여주기 위해, 오류 내역에 대한 엑셀 파일을 임시 디렉토리에 생성한다.

⑤ 엑셀 데이터 업로드에 대한 데이터 처리 건수를 보여준다.

⑥ 오류 내역건이 있으면, 사용자에게 오류 내역 엑셀 파일을 다운로드시킨다.

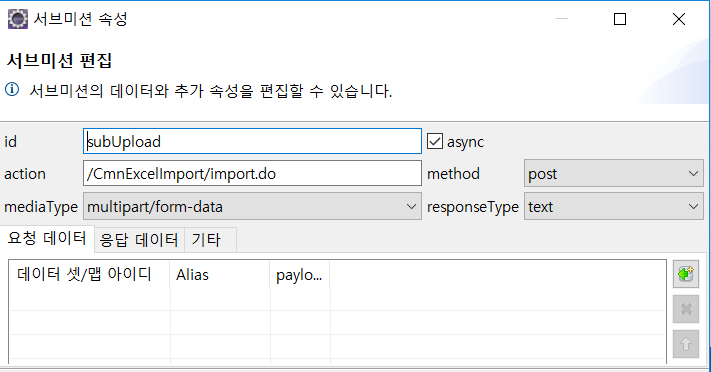
■ 작업 순서

1) 화면단에 엑셀 파일을 업로드하기 위한 공통 UDC(udcCmnPFileUpload)을 추가한다.

- 해당 UDC의 fileSelection 이벤트를 구현한다.

- 사용자가 파일 선택 완료시, 업로드 서브미션을 호출한다.

- 업로드 서브미션은 각 화면단에서 정의하여 사용한다.



|  |
| --- |
| function onUdcCmnPFileFileSelection(/\* cpr.events.CUIEvent \*/ e){  var submit = app.lookup("subUpload");  var voFile = null;    //파일 선택정보  for(var i=0, len=e.files.length; i<len; i++){  var voFile = e.files[i];  submit.addFileParameter(voFile.name, voFile);  }    //파일 업로드  util.Submit.send(app, "subUpload", null, function(pbSuccess){  if(pbSuccess){  var voDmResUpload = app.lookup("dmResUpload");  var vsSaveFileName = voDmResUpload.getValue("strSaveFileNm");  var vsFileName = voDmResUpload.getValue("strFileNm");  if(!ValueUtil.isNull(vsFileName)){  app.lookup("udcCmnFile").doDownloadTemp(vsFileName, vsSaveFileName);  }  }  });  } |

2) 업로드 요청을 처리하는 컨트롤러(Controller)에서 엑셀 데이터를 처리하기 위한 서비스를 호출한다.

|  |
| --- |
| /\*\*  \* 엑셀을 읽어들인다.  \* **@param** request  \* **@param** response  \* **@throws** Exception  \*/  @RequestMapping("import.do")  **public** View upload(HttpServletRequest request, HttpServletResponse response, RequestData requestData) **throws** Exception {    xxxService.uploadExcel(request, response, requestData);    **return** requestData.getDefaultView();  } |

3) 엑셀 데이터를 읽어들어, 데이터를 처리한다.(ServiceImpl 단에서 처리)

- ExcelImporter 객체를 생성하여, 엑셀 데이터를 읽어들인다.

(엑셀 데이터는 각 ROW별 ExcelVO 객체에 담겨있음)

- 행(ROW)만큼 돌면서, 비즈니스 로직 체크/유효성 체크 등을 수행한 후, 정상적인 데이터이면 해당 데이터를 데이터베이스 테이블에 INSERT 또는 UPDATE 처리한다.

- 데이터에 오류가 있는 경우, 해당 내역을 오류 목록에 넣는다.

- 오류 내역이 존재하는 경우, 오류 내역에 대한 임시 엑셀 파일을 생성한다.

※ 주의사항: ServiceImpl단에서 데이터 처리시try { … } catch() {…}로 예외(Exception)을 처리하는 경우에는 에러가 발생했을 때, 데이터 전체에 대한 롤백처리가 되지 않는다. 즉, 각각 처리된 데이터마다 Auto-commit이라고 생각하면 된다. 따라서, 만약 전체 데이터에 대해 하나의 트랜잭션으로 처리되어야 하는 경우에는 예외(Exception)가 ServiceImpl 위 단으로 전파되도록 try/catch문을 사용하지 않던가, catch문에서 예외를 받아서 다시 throw로 던진다.

|  |
| --- |
| **public** **void** uploadExcel(HttpServletRequest request, HttpServletResponse response, RequestData requestData) **throws** Exception {  //1. 엑셀 임포트 객체 생성  ExcelImporter excelImporter = **new** ExcelImporter();    //2. 엑셀 데이터 읽음  List<ExcelVO> dataList = excelImporter.getCellDataList(request, response, requestData);    // 데이터 오류 목록  List<ExcelVO> listErr = **new** ArrayList<ExcelVO>();    Map<String, Object> mapParam = **new** HashMap<String, Object>();  String strAprvNo = "";  **boolean** bChk = **true**;  **int** iSucssCnt = 0;  **for**(ExcelVO data : dataList){  **try** {  mapParam.clear();    strAprvNo = StringUtil.*fixNull*(data.getCELL0());    // 업무 비지니스 로직 체크 - 승인번호 중복 체크  bChk = xxxService.checkAacCashReceiptAprvNo(strAprvNo, **false**);  **if**(!bChk) {  **throw** **new** AppWorksException("이미 존재하는 승인번호 입니다. [승인번호 : "+ strAprvNo +"]");  }    // 승인번호  mapParam.put("APRV\_NO", strAprvNo);  // 회계단위  mapParam.put("ACCT\_UNIT\_RCD", data.getCELL1());    // 업무 데이터 저장  xxxService.insertAacCashReceipt(mapParam);    iSucssCnt++;  } **catch** (AppWorksException e){  // 오류 발생시, 오류 메시지 저장  data.setERRMSG(e.getMessage());  listErr.add(data);  } **catch** (Exception e) {  // 오류 발생시, 오류 메시지 저장  data.setERRMSG(e.getMessage());  listErr.add(data);  }  }    Map<String, Object> mapData = **new** HashMap<String, Object>();  // 오류내역이 존재할 경우  **if**(listErr.size() > 0) {  //엑셀 파일 헤더 정보 셋팅  LinkedHashMap<String, String> mapHeader = **new** LinkedHashMap<String, String>();  mapHeader.put("ERR", "오류 메시지");  mapHeader.put("1", "매입일시");  mapHeader.put("2", "가맹점명");  mapHeader.put("3", "가맹점 사업자번호");  mapHeader.put("4", "공급가액");  mapHeader.put("5", "세액");  mapHeader.put("6", "봉사료");  mapHeader.put("7", "합계");  mapHeader.put("8", "승인번호");  mapHeader.put("9", "사용자");  mapHeader.put("10", "가맹점 유형");  mapHeader.put("11", "거래구분");  mapHeader.put("12", "공제여부 결정");  mapHeader.put("13", "비고");    //오류 내역 임시파일 저장  Map<String, String> fileInfo = excelImporter.createExcelErrFile(request, response, "현금영수증\_오류내역", mapHeader, listErr);    //파일정보  mapData.put("strFileNm", fileInfo.get("FILE\_NM"));  mapData.put("strSaveFileNm", fileInfo.get("SAVE\_FILE\_NM"));  } **else** {  //파일정보  mapData.put("strFileNm", "");  mapData.put("strSaveFileNm", "");  }  mapData.put("strCnt", iSucssCnt);    requestData.setResponse("dmResUpload", mapData);  } |

4) 오류내역이 존재하는 경우, 클라이언트에 오류내역 다운로드 시킨다.

|  |
| --- |
| function onUdcCmnPFileFileSelection(/\* cpr.events.CUIEvent \*/ e){  var submit = app.lookup("subUpload");  var voFile = null;    //파일 선택정보  for(var i=0, len=e.files.length; i<len; i++){  var voFile = e.files[i];  submit.addFileParameter(voFile.name, voFile);  }    //파일 업로드  util.Submit.send(app, "subUpload", null, function(pbSuccess){  if(pbSuccess){  //1. 저장 건수를 보여주거나, 기타 업무처리 작업 수행    var voDmResUpload = app.lookup("dmResUpload");  var vsSaveFileName = voDmResUpload.getValue("strSaveFileNm");  var vsFileName = voDmResUpload.getValue("strFileNm");    //오류 내역 파일이 존재하면, 다운로드한다.  if(!ValueUtil.isNull(vsFileName)){  app.lookup("udcCmnFile").doDownloadTemp(vsFileName, vsSaveFileName);  }  }  });  } |

* 1. 엑셀 데이터 출력(Export)

■ 화면에서 엑셀 다운로드

화면상에 있는 그리드의 데이터를 엑셀로 출력할 수 있는 기능을 제공한다.

기본적인 화면상의 그리드(Grid)에 대한 엑셀 출력은 공통 UDC인 commTitle 컨트롤의 기능으로 제공된다. 그 외에 별도로 화면상의 그리드 데이터를 엑셀파일로 다운로드 받고자 하는 경우에는 공통 유틸 스크립트 API를 호출하여 출력한다.

<작업순서>

① 각 화면 상단에서 공통 스크립트를 임포트한다.

|  |
| --- |
| //공통 스크립트 임포트  var util = createCommonUtil(app); |

② util 클래스에 있는 exportExcel() 함수를 호출한다.

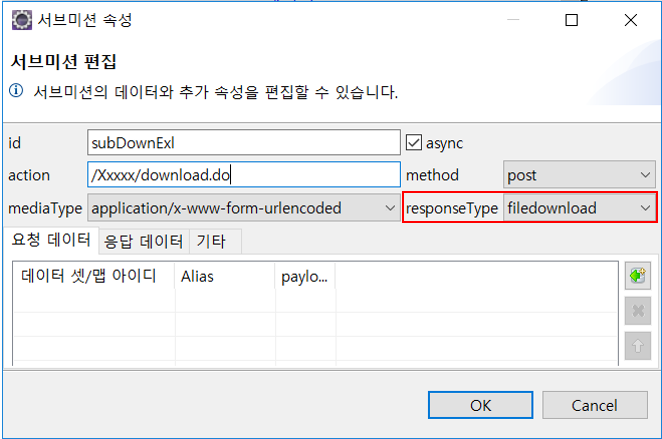
|  |
| --- |
| //엑셀 Export  util.Grid.exportData(app, "grd1", "코드목록"); |

* psExcludeColumns: 제외할 컬럼이 있는 경우 “COL\_1, COL\_2”와 같은 형태로 지정
* psFileType: 파일유형(xls, xlsx, cvs 중 선택… 디폴트는 xlsx임)

■ SQL 조회된 데이터 엑셀 다운로드

1) 서버상에서 SQL로 조회한 결과를 엑셀 파일로 다운로드 할 수 있다. 엑셀 출력 결과를 곧바로 HttpServletReponse 객체의 OutputStream으로 내보내게 된다.

※ 주의사항: 엑셀 다운로드를 하는 경우에 반드시 해당 컨트롤러(Controller) 클래스는 null을 반환해야 함, 또한 해당 다운로드 요청을 보내는 UI단 서브미션의 reponseType을 filedownload로 지정해야 한다.



|  |
| --- |
| @RequestMapping("/downloadExcel.do")  **public** View downloadExcel(HttpServletRequest request, HttpServletResponse response, RequestData requestData) **throws** Exception {  //요청 파라메터 셋팅  Map<String, String> mapParam = **new** HashMap<String, String>();  mapParam.put("UNIT\_SYS\_RCD", requestData.getString("strUnitSystemRcd"));  mapParam.put("MSG\_DIV\_RCD", requestData.getString("strMessageDivCd"));  mapParam.put("MSG\_TEXT", requestData.getString("strMessage"));    //엑셀 헤더 정보  LinkedHashMap<String, String> mapHeader = **new** LinkedHashMap<String, String>();  mapHeader.put("CD", "코드");  mapHeader.put("CD\_NM", "코드명");    //출력 데이터  List listData = xxxService.selectDataList(mapParam);    //엑셀 출력 -> 다운로드  DataExportUtil.exportExcel(request, response, "테스트.xlsx", mapHeader, listData);    //반환값은 null임  **return** **null**;  } |

* strFileName: 다운로드 엑셀 파일명(.xls 또는 .xlsx)
* mapHeader: 헤더 정보
* listData: 출력 데이터

2) 서버에서 임시로 엑셀 파일을 생성 후, 후처리하는 경우에는 DataExportUtil 클래스의 createExcel() 메소드를 호출하여 생성할 수 있다. 단, 해당 엑셀 파일을 생성한 이후, 후처리 작업이 끝나면 반드시 해당 임시 파일을 삭제해 주어야 한다.

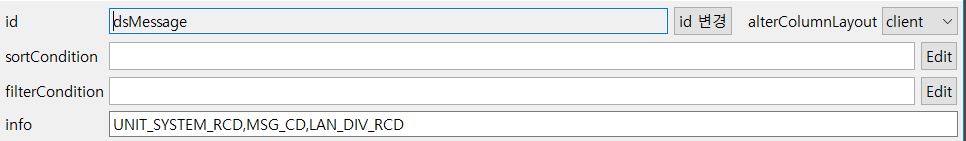
임시 파일은 서버 어플리케이션의 Export 경로(컨텍스트 루트 하위의 /export 경로에 생김)

|  |
| --- |
| //엑셀 헤더 정보  LinkedHashMap<String, String> mapHeader = **new** LinkedHashMap<String, String>();  mapHeader.put("CD", "코드");  mapHeader.put("CD\_NM", "코드명");    //출력 데이터  List listData = xxxService.selectDataList(mapParam);    //임시 경로에 엑셀 생성  Map fileInfo = DataExportUtil.createExcel(request, response, "테스트.xlsx", mapHeader, listData);  if(fileInfo != null){  File file = new File(fileInfo.get("FILE\_PATH");  //파일에 대한 후처리 작업(읽기/쓰기)    //파일에 대한 후처리 작업이 끝나면… 반드시 임시 파일 삭제처리  if(file.exists()){  file.delete();  }  } |

* 1. 마지막 작업행 찾기

CRUD용 그리드의 경우 행추가, 변경 등과 같이 작업 저장 후 데이터 재조회시 작업 마지막에 선택된 로우를 유지한다.

* 선택된 로우(Row) 유지를 위해 데이터셋의 Info에 설정한 PK 컬럼값을 사용한다.



- 일련번호와 같이 신규 데이터에 대한 PK 컬럼의 값이 서버단에서 생성되는 경우 트랜잭션 종료후 화면으로 생성된 Key값을 반환한다.

|  |
| --- |
| @RequestMapping("/save.do")  **public** View save(HttpServletRequest request, HttpServletResponse response, RequestData requestData) **throws** Exception {  //데이터 저장 및 PK 컬럼 반환  String strSerialNo = cmnMessageService.saveCmnMessage(requestData);    //화면단에서 저장된 행(ROW)를 찾기 위한 FindKey 반환  Map<String, Object> message = **new** HashMap<String, Object>();  message.put("strFindRowKey", "SERIAL\_NO =='"+strSerialNo+"'");  requestData.setMetadata(**true**, message);    **return** requestData.getDefaultView();  } |

- 키컬럼이 여러 개인경우 “&&”를 이용하여 구분한다.

|  |
| --- |
| message.put("strFindRowKey", "SERIAL\_NO == '"+strSerialNo+"' && SEQ == '"+strSeq)+ "'"; |

- 데이터셋이 여러 개인 경우에는 아래의 규칙을 따라 Find Key를 반환한다.(단, 데이터셋이 1개인 경우에는 데이터셋명을 명시하지 않아도 됨)

|  |
| --- |
| [데이터셋명1] + 구분자(:) + Find Key1 + 구분자(|) + [데이터셋명2] + 구분자(:) + Find Key2  예시) dsMst:SERIAL\_NO == ‘AAA’  dsMst:SERIAL\_NO == ‘AAA’ | dsDtl:SEQ == ‘BBB’ |

* 1. 그리드 틀고정

그리드 컬럼에 대한 틀고정은 상/하 및 왼쪽/오른쪽 고정이 가능하다.

틀고정과 관련된 그리드 속성은 아래와 같다.

|  |  |
| --- | --- |
| 속성 | 비고 |
| leftSplit | 왼쪽으로 고정할 컬럼 수를 지정한다.  leftSplitWidth가 지정되어 있지 않으면, 넓이는 자동 계산됨 |
| leftSplitWidth | 왼쪽으로 고정될 영역의 넓이  미 설정 시 현재 설정된 split cell width 를 계산하여 크기를 차지함 (split cell 중 autoFit 에 해당되면, 기본 100px 로 계산됨)  \*\* 필요시 지정 |
| rightSplit | 오른쪽으로 고정할 컬럼 수를 지정한다.  rightSplitWidth가 지정되어 있지 않으면, 넓이는 자동 계산됨 |
| rightSplitWidth | 오른쪽으로 고정될 영역의 넓이  미 설정 시 현재 설정된 split cell width 를 계산하여 크기를 차지함 (split cell 중 autoFit 에 해당되면, 기본 100px 로 계산됨)  \*\* 필요시 지정 |
| topSplitHeight | 디테일 상단에 고정할 영역의 기본 높이를 지정한다.  \*\* 그룹핑(소계)이 지정된 그리드에서는 지원되지 않는다. |
| bottomSplitHeight | 디테일 하단에 고정할 영역의 기본 높이를 지정한다.  \*\* 그룹핑(소계)이 지정된 그리드에서는 지원되지 않는다. |

<왼쪽 틀고정 예시화면>



* 1. 그리드 셀병합(Suppress)

그리드(Grid)에서 세로 셀에 대해 세로 병합 기능 제공한다. Suppress를 지정한다고 해서 실제 데이터가 병합되지는 않으며, 단순히 디스플레이시에만 해당 ROW가 병합되어 보여지게 된다.

그리드의 디테일(Detail) 컬럼에는 Suppress 기능과 관련된 2가지 속성을 제공한다.

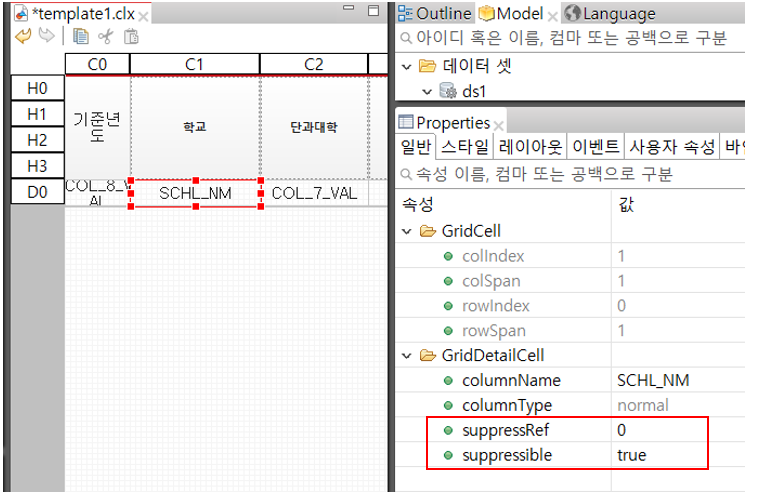
|  |  |
| --- | --- |
| 속성 | 비고 |
| suppressible | 바로 직전 컬럼의 값과 일치하는 셀은 텍스트를 숨기도록 할지 여부를 설정한다. |
| suppressRef | suppressible(셀 텍스트 숨김) 효과에 영향을 줄 다른 컬럼 인덱스를 설정한다. 예를들어 B컬럼이 앞 컬럼인 A컬럼의 값에 따라 suppress를 하고자 하는 경우에 A컬럼의 인덱스를 지정하면 된다.  \*\* 디테일의 컬럼 인덱스는 0부터 시작한다. |

<예시>

기준년도, 학교, 단과대학, 학과(전공), 구분 컬럼에 대해 세로 셀병합을 하고자 하는 경우 아래와 같이 설정한다.

① suppressible 속성을 true로 설정한다.

② 만약 해당 컬럼이 앞선 컬럼에 따라 병합되어야 하는 경우에는 suppressRef에 앞선 컬럼의 인덱스를 지정한다.





* 1. 그리드 소계/합계

그리드 내에 그룹 소계를 추가하거나, 합계 행을 추가하는 방법을 기술한다.

■ 합계

렌더링되는 그리드 데이터들의 합계를 표시하기 위해서는 별도의 풋터(Footer)행을 추가하여 사용한다.

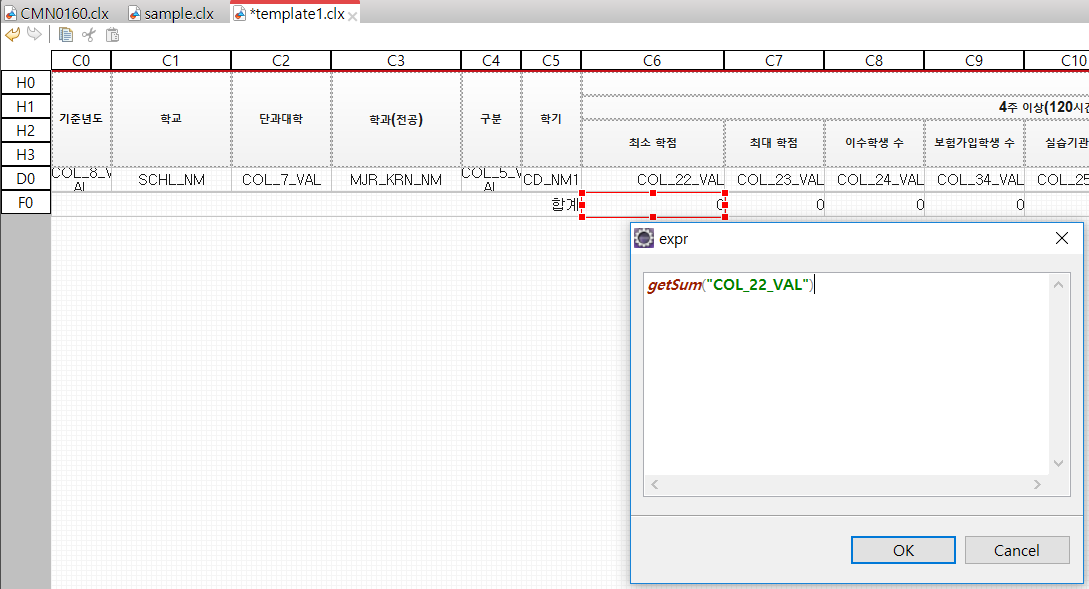
① 그리드를 더블클릭하여 편집창으로 들어감

② 좌측 회색부분(H0, D0 선언부분)에서 마우스 우측버튼 클릭하여 푸터 추가 메뉴 클릭



③ 풋터 행 추가후, 각 컬럼 선택 후 더블 클릭하여 표현식(Expression) 입력창을 호출한다.

④ 해당 합계 컬럼에 보여줄 값을 그룹핑 함수를 사용하여 표시한다. (해당 편집창에서 Ctrl+스페이지 키보드 누르면 사용가능한 변수 및 함수 목록들이 보여진다.)



* 그룹함수의 인자값으로 들어가는 변수들은 쌍따옴표(“) 또는 따옴표(‘)로 감싸져야 한다.
* 해당 화면에 있는 컨트롤의 값을 참고하고자 하는 경우에는 ID 참조를 통해 해당 값을 가져올 수 있다.

|  |
| --- |
| 조회조건의 년도(ipbYear 컨트롤)의 값을 가져와서 사용하는 경우  🡺 #ipbYear.value + ‘년도 합계’ 형태로 사용 가능 |

대표적인 그룹함수는 아래와 같다.

|  |  |
| --- | --- |
| 그룹 함수 | 비고 |
| getSum() | 입력 받은 컬럼명의 데이터의 합계를 반환합니다. |
| getMax() | 입력 받은 컬럼명의 데이터 중 최대값을 반환합니다. |
| getMin() | 입력 받은 컬럼명의 데이터 중 최소값을 반환합니다. |
| getAvg() | 입력 받은 컬럼명의 데이터의 평균을 반환합니다. |
| getConditionalSum() | 조건에 맞는 Row중 입력 받은 컬럼명의 데이터의 합계를 반환합니다. |
| getConditionalMax() | 조건에 맞는 Row중 입력 받은 컬럼명의 데이터의 최대값을 반환합니다. |
| getConditionalAvg() | 조건에 맞는 Row중 입력 받은 컬럼명의 데이터의 평균을 반환합니다. |
| getRowCount() | 현재 Row 수를 반환합니다. |

표현식에서 사용 가능한 형식에 대한 자세한 설명은 eXBuilder6의 Help를 참고한다.

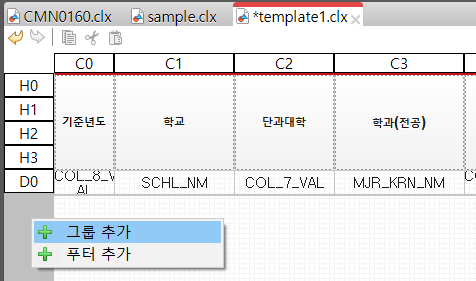
참고) 이클립스 Help > Help Contents > eXbuilder6 > 앱개발 > 컨트롤 > 표현식

■ 소계

렌더링되는 그리드 데이터들의 그룹별 소계를 표시하기 위해서는 별도의 그룹(RowGroup)행을 추가하여 사용한다.

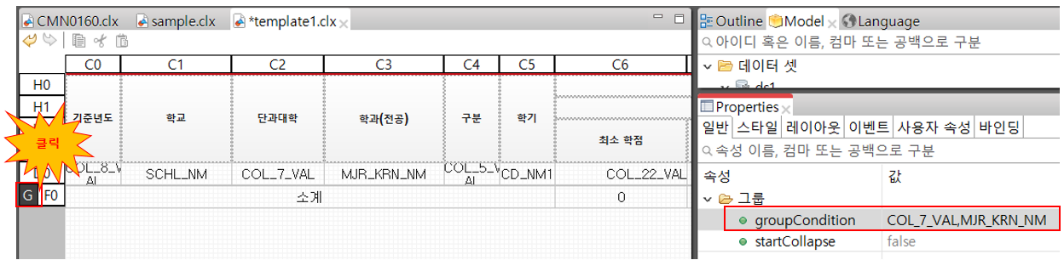
① 그리드를 더블클릭하여 편집창으로 들어감

② 좌측 회색부분(H0, D0 선언부분)에서 마우스 우측버튼 클릭하여 그룹 추가 메뉴 클릭



③ 그룹 행 추가후, 각 컬럼 선택 후 더블 클릭하여 표현식(Expression) 입력창을 호출한다.

④ 그룹 행에서 [G] 영역을 클릭하여 해당 속성의 groupCondition값에 그룹 조건을 지정한다.



* 조건이 여러 개인 경우에는 콤마(,) 구분자로 컬럼명을 지정함

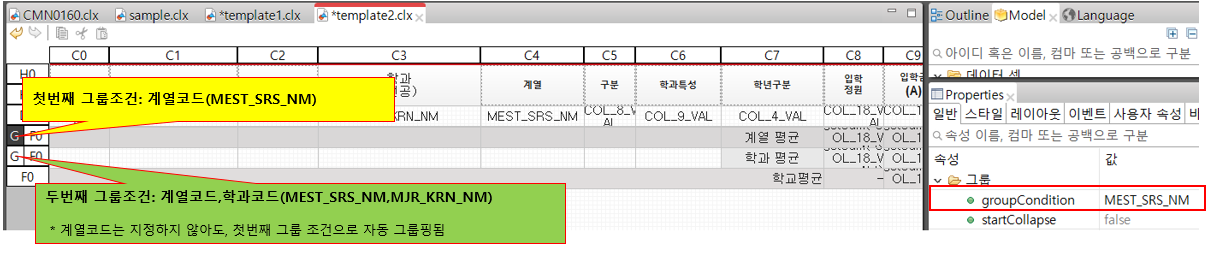
⑤ 해당 그룹별 소계 컬럼에 보여줄 값을 그룹핑 함수를 사용하여 표시한다. 자세한 사항은 상단의 합계처리 부분을 참고한다. (해당 편집창에서 Ctrl+스페이지 키보드 누르면 사용가능한 변수 및 함수 목록들이 보여진다.)

⑥ 그룹 소계는 1개 이상 반복적으로 추가 가능하다. 그룹 소계를 여러 개 지정하는 경우에는 그룹 소계를 지정하는 순서(위치)에 영향을 받는다.

* 그룹 소계는 상위에 선언된 그룹 소계의 그룹 조건에 따라 데이터가 그룹핑 처리된다. 예를 들어, 첫번째 그룹 행에 선언한 그룹 조건이 VAL\_1이고, 두번째 그룹 행에 선언한 그룹 조건이 VAL\_2인 경우에 두번째 그룹 행은 첫번째 그룹 조건의 VAL\_1과 더블어 두번째 행에서 선언한 그룹 조건 VAL\_2 값으로 데이터가 그룹핑되어 보여진다. 따라서, 그룹 소계를 2개 이상 사용하고자 하는 경우에는 데이터 그룹핑 범위가 큰 것들에서 작은 것들 순으로 그룹 소계 행을 추가 선언해야 한다.

예시화면)

그룹 소계가 2개인 경우



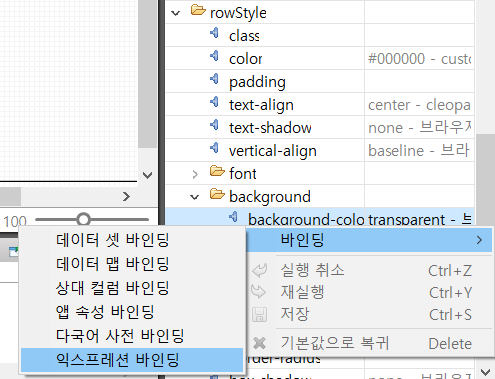


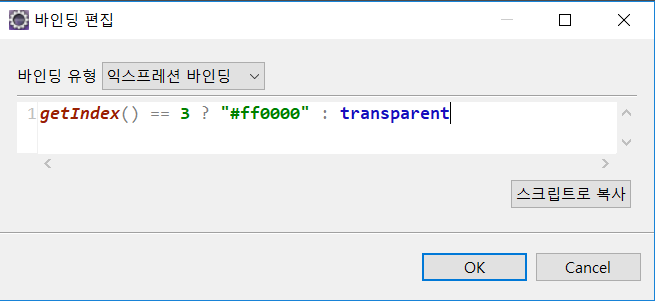
* 1. 그리드 ROW 스타일 변경

조건에 따라 그리드 ROW별 배경색 변경 및 폰트 색상 변경 작업이 필요할 수 있으며, 표현식(Expression) 바인딩을 통해 쉽게 적용 가능하다.

경로) 그리드 > 스타일 > rowStyle 속성에서 변경 가능

예를 들어 3번째 행(Row)의 배경색을 변경하고자 하는 경우에는 아래와 같은 표현식을 적용할 수가 있다.





* 1. 마스크/포맷 처리

인풋 컨트롤 계열에 마스크 처리 및 포맷처리 방법에 대해 기술한다.

■ 마스크 처리

마스크 처리를 지원하는 컨트롤은 마스크 에디터(MaskEditor) 및 데이트인풋(DateInput)이다.



* 마스크 에디터 컨트롤의 mask 속성에 지정한다.

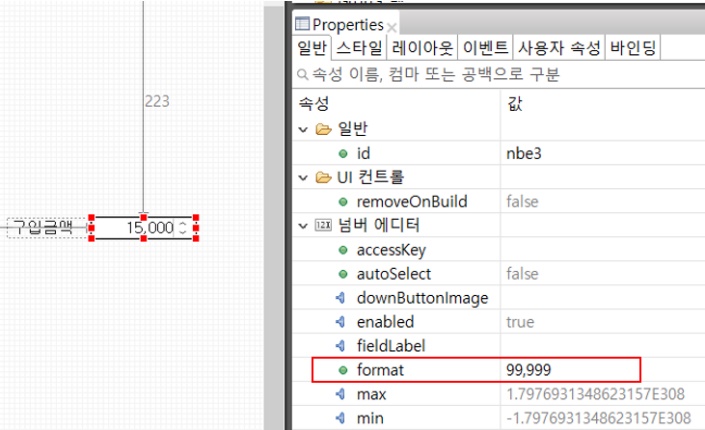
예시) 주민등록번호 마스크 처리(숫자만 입력 가능 처리시)

|  |
| --- |
| mask: 000000-0000000 |

|  |  |
| --- | --- |
| 데이터 타입 | 비고 |
| 문자열 타입 | | 포맷 | 설명 | | --- | --- | | 0 | 숫자 문자 입력 | | a | 영문 소문자 | | A | 영문 대문자 | | z | 영문 소문자 숫자 | | Z | 영문 대문자 숫자 | | x | 영문 대소문자 | | X | 영문 대소문자 숫자 | |
| 날짜 타입 | | 포맷 | 설명 | | --- | --- | | YYYY | 4자리 연도: 0001...9999 | | YY | 2자리 연도: 00..99 | | Y | 전체 연도: 0001...9999 | | MM | 두 자리 수의 월: 01...12 | | M | 월: 1...12 | | DD | 두 자리 수의 날짜: 01...31 | | D | 날짜: 1...31 | | h | 12시간: 1...12 | | hh | 두 자리 수의 12시간: 01...12 | | H | 24시간: 0...23 | | HH | 두 자리 수의 24시간: 00...23 | | m | 분: 0...59 | | mm | 두 자리 수의 분: 00...59 | | s | 초: 0...59 | | ss | 두 자리 수의 초: 00...59 | |

■ 포맷 지정

포맷(format)을 지원하는 컨트롤은 넘버에디터(NumberEditor), 데이트인풋(DateInput), 아웃풋(Output)이다.



* format 속성에 지정한다.

넘버 에디터를 사용한 금액 표시를 위한 포맷 지정시는 아래와 같다.

* 천단위 콤마(,) 표현시: 3자리 마다 콤마(,)를 넣은 포맷을 지정한다.

|  |
| --- |
| 예시1) s#,##0  예시2) 999,999,999 |

* 소수점 표현시: 점(.) 이후에 소수점 자릿수를 표현한다.

|  |
| --- |
| 예시1) #,##0.99  예시2) 90.00 |

* 바인딩되는 값이 널(NULL)인 경우에도 디폴트로 0으로 표시하고자 하는 경우에는 금액 마지막 포맷에 ‘0’을 지정한다.

|  |
| --- |
| 예시1) s#,##0  예시2) 999,999,990 |

* 수정/입력되는 경우에 입력 가능한 숫자의 길이(Length)는 format 속성에 지정된 포맷 문자열의 길이만큼으로 제한된다.

포맷(format) 지정시에 고려할 사항들은 아래와 같다.

|  |  |
| --- | --- |
| 구분 | 내용 |
| 아웃풋(Output) | 인원수, 금액 등의 숫자 표현하는 경우에 사용  - dataType 속성: number  - format 속성: s#,##0 또는 s#,##9 |
| 읽기전용 항목/컬럼인 경우 | 읽기 전용 항목 또는 그리드 컬럼인 경우에는 format값은 아래와 같이 지정한다.  - format 속성: s#,##0 또는 s#,##9 |
| 수정/입력 가능한 항목/컬럼인 경우 | 넘버 에디터에 format 지정시에는 해당 속성에 지정한 포맷 길이만큼만 입력이 가능하다.  따라서, 12자리 금액을 입력 및 표시하기 위해서는 format에도 12자리만큼 ‘0’ 또는 ‘9’를 지정해야 한다.  예를 들어, format=9,999로 지정한 경우에는 4자리 금액까지만 입력이 가능하다.  예시) 12자리 금액 입력시 format값은 아래와 같다.  Format: 999,999,999,999 (또는 NULL인 경우 ‘0’표시시 999,999,999,990) |

※ 적용 규칙

프로젝트 UI 소스에 대해서는 전체적으로 아래와 같은 공통적인 규칙을 적용한다.

|  |
| --- |
| 1) Output 컨트롤의 숫자 포맷은 s#,###를 사용한다.  - 단, 값이 NULL인 경우에도 ‘0’을 표시하고자 하는 경우에는 s#,##0을 사용한다.  2) 읽기 전용의 NumberEditor 컨트롤의 포맷은 s#,###를 사용한다.  - 단, 값이 NULL인 경우에도 ‘0’을 표시하고자 하는 경우에는 s#,##0을 사용한다.  3) 편집 가능한 NumberEditor 컨트롤의 포맷은 입력 가능한 최대 자릿수를 기준으로 포맷을 지정하여 사용한다.  - 금액을 표시하는 경우에는 최대 12자리 길이의 999,999,999,999포맷을 지정한다.(단, 값이 NULL인 경우에도 ‘0’을 표시하는 경우에는 999,999,999,990을 사용한다.)  - 음수(-)를 입력받을 수 있는 경우에 한해서만 포맷에 s를 덧붙여 지정한다. |

|  |  |
| --- | --- |
| 데이터 타입 | 비고 |
| 숫자 타입 | | **포맷** | **설명** | | --- | --- | | s | 음수/양수 문자표시 여부. 첫 번째로 지정해야 동작합니다. | | 0 | 숫자를 입력 받으며 소수점에서는 값이 없으면 강제로 0이 채워집니다. | | 9 | 숫자를 입력 받으며 소수점에서는 값을 채우지 않습니다. | | , | 숫자 구분자 콤마를 표시합니다. | | . | 소수점을 표시합니다. | | # | [0-9]입력 받으며 값이 없으면 채우지 않습니다. 이 포매팅은 길이 제한 없이 숫자 설정이 됩니다. | |
| 날짜 타입 | | 포맷 | 설명 | | --- | --- | | YYYY | Year: 0001...9999 | | YY | Year: 00...99 | | Y | Year: 0001...9999 | | M | Month: 1...12 | | Mo | Month: 1st...12th | | MM | Month: 01...12 | | MMM | Month: Jan ... Dec | | MMMM | Month: January ... December | | DDD | Day of Year: 1...365 | | DDDo | Day of Year: 1st...365th | | DDDD | Day of Year: 001...365 | | DD | Day of Month: 01...31 | | D | Day of Month: 1...31 | | Do | Day of Month: 1st...31st | | d | Day of Week: 0 1 ... 5 6 | | do | Day of Week: 0th 1st ... 5th 6th | | dd | Day of Week: Mo | | ddd | Day of Week: Mon | | dddd | Day of Week: Monday | | Q | Quarter: 1 2 3 4 | | Qo | Quarter: 1st 2nd 3rd 4th | | w | Week of Year: 1 2 ... 52 53 | | wo | Week of Year: 1st 2nd ... 52nd 53rd | | ww | Week of Year: 01 02 ... 52 53 | | W | Week of Year(ISO): 1 2 ... 52 53 | | Wo | Week of Year(ISO): 1st 2nd ... 52nd 53rd | | WW | Week of Year(ISO): 01 02 ... 52 53 | | A | AM/PM: AM PM | | a | AM/PM: am pm | | h | Hour: 1...12 | | hh | Hour: 01...12 | | H | Hour: 0...23 | | HH | Hour: 00...23 | | k | Hour: 1...24 | | kk | Hour: 01...24 | | m | Minute: 0...59 | | mm | Minute: 00...59 | | s | Second: 0...59 | | ss | Second: 00...59 | |

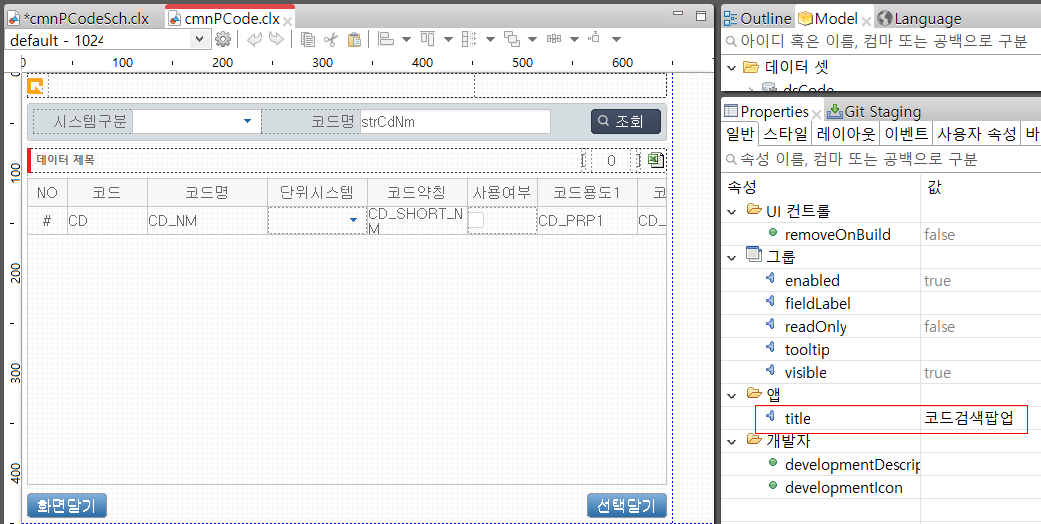
* 1. 팝업 호출

팝업 호출시에는 공통 API를 사용해서 호출한다.

|  |
| --- |
| util.Dialog.open(app, "app/cmn/cmnPCode", 650, 450, function(/\*\*@type cpr.events.CUIEvent \*/e){  /\*\*@type cpr.controls.Dialog\*/  var dialog = e.control;  var returnValue = dialog.returnValue;  if(returnValue != null){  //팝업에서 데이터 선택시 처리  app.lookup(“ipbCd”, returnValue.CD);  }  }, initValue); |

* 팝업 타이틀: 해당 팝업UI 화면 앱의 title 속성에 지정
* 팝업 화면에도 appHeader.clx UDC를 추가한다.(팝업 타이틀 및 메시지 디스플레이)

<팝업 예시>



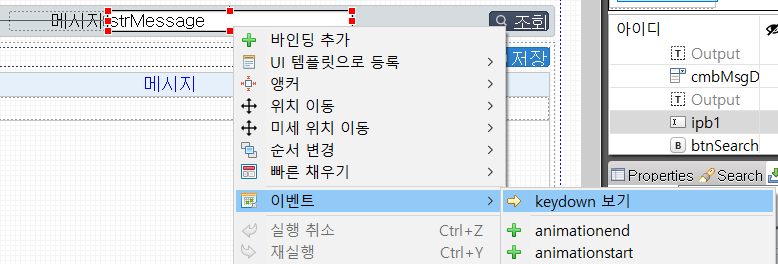
■ 기타

팝업창이 [화면닫기] 버튼 또는 [X]버튼을 통해 닫히든 어떤 형태로 팝업이 닫히기 전에 필수 로직을 체크한다거나, 부모창에 값을 반환해야 하여 화면을 갱신해야 하는 등의 액션이 필요한 경우에는 반드시 팝업 화면의 before-unload 이벤트를 구현한다.

|  |
| --- |
| /\*  \* Body에서 before-unload 이벤트 발생 시 호출.  \* 앱이 언로드되기 전에 발생하는 이벤트 입니다. 취소할 수 있습니다.  \*/  function ***onBodyBeforeUnload***(/\* cpr.events.CEvent \*/ e){  var vsAttcFileNo = app.lookup("dmParam").getValue("strAttcFileNo");  var vnFileCnt = app.lookup("grdFile").rowCount;  //첨부파일이 없으면... 모두 삭제된 걸로 판단  if(vnFileCnt < 1){  vsAttcFileNo = "";  }    //부모창에 리턴값 반환  app.setHostProperty("returnValue", {"attcFileNo":vsAttcFileNo, "fileCnt":vnFileCnt});  } |

* 1. 조회 조건영역 처리관련

조회조건 내에 문자열 검색 등을 위해 인풋박스(inputBox) 컨트롤이 존재하는 경우에는 사용자의 편의성을 위해, 기본적으로 해당 컨트롤에 keydown 이벤트를 등록하여 사용자가 조건을 입력하고 엔터키를 누를 경우에 자동 조회되도록 처리한다. 단, 해당 인풋박스에 다른 이벤트(value-change 등)이 추가되어 있거나, 업무 특성에 의해 keydown 이벤트를 사용하지 않아야 하는 경우에는 제외한다.



|  |
| --- |
| /\*  \* 인풋 박스에서 keydown 이벤트 발생 시 호출.  \* 사용자가 키를 누를 때 발생하는 이벤트.  \*/  function ***onIpb1Keydown***(/\* cpr.events.CKeyboardEvent \*/ e){  if(e.keyCode == cpr.events.KeyCode.ENTER){  //Enter키 입력시, 조회 버튼 클릭 이벤트 발생  util.Control.dispatchEvent(app, "btnSearch", "click");  }  } |

* 1. 검색팝업 처리 규칙

검색용 팝업 UI는 기본적으로 아래와 같이 3가지 파트로 구성한다.

|  |
| --- |
|  |

* 검색어입력 필드, 팝업실행버튼, 검색결과명칭 표시 필드
* 검색어 입력에 명칭을 표시하고 코드는 내부적으로 처리할 경우 명칭필드는 생략할 수 있다.
* [처리규칙]

1. 기본적으로 3개의 컨트롤로 구성한다.
   * + 검색 입력 컨트롤: 코드 또는 코드명을 입력하여 검색할 수 있는 용도
     + 팝업 실행 버튼: 팝업호출
     + 코드명 디스플레이 컨트롤: 코드명을 보여주기 위한 용도(Read-Only)
2. 코드/코드명 inputBox Control에 value-change이벤트를 등록한다.
3. 코드 또는 코드명으로 해당 데이터를 조회 할 수 있는 트렌젝션 및 서비스를 생성한다.
4. **코드 또는 코드명으로 검색 후, 1건인 경우에는 코드 필드에는 코드값을, 코드명 필드에는 코드명을 보여준다**.
5. 0건이 검색된 경우에는 팝업을 띄우며, 부모창의 검색어는 Clear한다. 조회는 하지 않는다.
6. 데이터가 2건 이상이 검색된 경우 입력된 검색어를 가지고 팝업화면을 띄우며, 부모창의 검색어는 Clear한다.(입력된 검색어를 팝업의 조회조건에 디폴트로 세팅해주고, 자동조회 한다.)
   1. UDC 개발

화면 UI에서 사용되는 검색용 팝업 호출을 위한 컴포넌트를 UDC(User Define Control)을 구현하여 여러 화면에 임포트하여 사용할 수 있다.

* 데이터 검색용 모듈(단건 검색 및 팝업호출)은 UDC로 개발한다. (참고UDC: **udcCmnPCode.clx**)
* UDC로 개발되는 컴포넌트는 단순한 비즈니스 로직과 UI로 구성되어야 함
* 여러 화면에서 사용하기 위해(임베디드 형태로 구현한) 개발된 화면은 UDC로 구현하지 않고, 임베디드 앱으로 개발한다.(예 – 공통 주소입력 화면 등)
  + 즉, 그리드, 프리폼 등이 포함된 화면 등은 UDC 대상이 아님

아래는 단건 및 팝업 호출용 UDC 모듈 개발방법에 대한 예시이다.

① 명명규칙

* udc + 단위시스템[3]+약어(대소문자 혼용)
* 해당 UDC가 단건 검색 + 팝업 호출용이라면, 파일명은 호출하는 팝업 화면의 파일명에 udc를 붙인 명칭으로 명명한다.

② UDC의 UI는 아래와 같이 구성한다.

* 해당 UDC는 조회조건 또는 프리폼 내의 항목으로 사용되거나, 그리드(Grid)의 컬럼으로 사용될 수 있기 때문에 UI에는 입력을 위한 InpuBox 및 Button으로만 구성한다.

|  |
| --- |
|  |

* 라벨 및 해당 데이터의 명칭을 보여주는 컨트롤은 각 화면단에서 별도로 추가하여 사용한다.

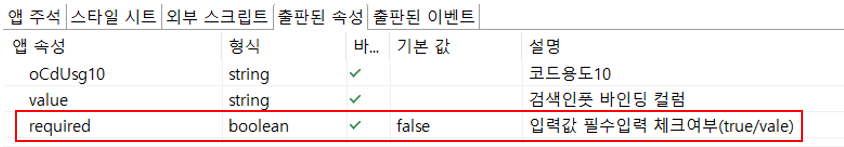
|  |
| --- |
|  |

* 검색 버튼은 사이즈 고정(Fix)이며, 인풋박스 컨트롤은 가변 사이즈로 적용한다.

③ 필드라벨(fieldLable)은 일반 컨트롤과 동일하게, 해당 UDC 컨트롤의 속성에 지정하며, 필요한 경우 해당 UDC의 앱속성으로 지정할 수도 있다.

* 필드라벨: 필수 입력

④ 필수입력 여부는 해당 UDC 컨트롤의 앱속성에 별도 지정한다.



필수값 여부에 따라 UDC내에서는 해당 앱의 속성이 변경될 때의 이벤트 내에서 필수 체크되어야 하는 컨트롤의 userAttr의 속성을 추가하는 로직을 구현해야 한다.

|  |
| --- |
| /\*  \* Body에서 property-change 이벤트 발생 시 호출.  \* 앱의 속성이 변경될 때 발생하는 이벤트 입니다.  \*/  function ***onBodyPropertyChange***(/\* cpr.events.CPropertyChangeEvent \*/ e){  if(e.property == "required"){  //필수 입력 체크 조건 설정  if(app.getAppProperty("required") === true){  var vcIpbCode = app.lookup("ipbCode");  vcIpbCode.fieldLabel = app.getHostProperty("fieldLabel"); //필드라벨 지정  vcIpbCode.userAttr({"required":"Y"}); //컨트롤 필수여부 사용자 속성 지정  }  }  } |

⑤ UDC내에서 데이터 검색을 위해 입력으로 받아야 하는 항목들은 해당 UDC의 앱 속성으로 정의한다. 입력 항목들의 명명규칙은 소문자 ‘i’로 시작한다.



⑥ UDC내에서 검색된 데이터에 대한 반환 항목들 역시 해당 UDC의 앱 속성으로 정의한다. 반환 항목들의 명명규칙은 소문자 ‘o’로 시작한다.



⑦ 해당 UDC가 그리드 내에 컬럼으로 사용될 수 있는 경우에는 그리드의 컬럼과의 값 매핑을 위핸 해당 UDC의 앱 속성에 value 속성이 들어가야 하며, 해당 value속성으로 값이 그리드 컬럼에 보여지거나 컬럼값으로 셋팅이 된다.

※ 주의사항

1) UDC에 인풋박스 계열의 컨트롤이 있는 경우, 해당 UDC가 그리드 컬럼으로 사용되는 경우에 Enter 이벤트가 동작하지 않도록 막는다.

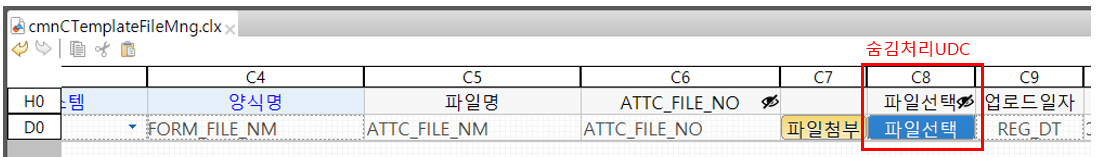
- 제한 사유: 그리드 편집상태에서 enter키를 사용하게 되면 편집상태가 선택상태로 되면서 udc에서 해당 udc가 어떤row를 참조하고있는지 잃어버리는 오류 방지하기 위한 목적임

- 참고 소스: udcCmnPCode.clx

|  |
| --- |
| function onIpbCodeKeydown(/\* cpr.events.CKeyboardEvent \*/ e){  //그리드 편집상태에서 enter키를 사용하게 되면 편집상태가 선택상태로 되면서  //udc에서 해당 udc가 어떤row를 참조하고있는지 잃어버리는 오류 방지하기 위해 Enter키 막기  if(e.keyCode == cpr.events.KeyCode.ENTER){  e.preventDefault();  }  } |

2) 버튼으로만 이루어진 UDC인 경우에, 해당 UDC가 그리드 컬럼으로 사용되는 경우 그리드의 뷰잉/편집 모드 동작으로 인하여 그리드가 뷰잉 모드일 때 해당 UDC가 보여지지 않고 사용자가 해당 셀을 클릭하여 편집모드로 들어갔을 경우에만 UDC로 만든 버튼이 보여지게 되어 사용자 식별성이 떨어지는 것을 방지하기 위해, 해당 UDC 버튼 컬럼은 숨김 처리하고 별도 버튼을 그리드 컬럼에 추가 후 버튼 클릭시 숨겨진 UDC의 버튼이 동작하도록 처리한다.

- 제한 사유: eXbuilder6의 그리드는 데이터 조회시에는 뷰잉(View) 모드로 동작하고, 사용자가 해당 셀을 클릭한 경우에 비로소 편집(Edit) 모드로 진입하게 된다. 그리드 내의 몇 몇 컨트롤(Checkbox, Button, RadioButton)을 제외 일반 컨트롤들 역시 편집 모드에서만 해당 컨트롤이 보이게 된다. UDC 역시 뷰잉 모드인 경우에는 해당 컨트롤이 보이지 않게 되며, 이러한 이유로 버튼으로만 구성된 UDC는 사용자 식별성을 위해 숨김처리해야 한다.



* 1. 임베디드 앱 개발

여러 화면에서 동일하게 임베디드되어서 사용되어져야 하는 화면을 임베디드앱으로 개발할 수 있다.

- 그리드, 프리폼이 있는 화면 등은 임베디드 앱으로 개발한다.

- 임베디드되는 화면은 헤더(embAppHeader)를 포함한다.

- 개발은 일반 화면을 개발하는 것과 동일하게 개발한다.

■ 함수 노출(exports)

메인화면 내에서 임베디드된 앱내의 특정 메소드를 호출하기 위해서는 임베디드앱에서 관련 함수들을 외부로 노출시켜야 한다.

예를 들어, 임베디드앱 내의 그리드에 데이터를 가져오는 doList() 함수가 있는 경우, 해당 함수를 노출시켜야 메인화면에서 해당 함수 호출이 가능하다.

|  |
| --- |
| /\*\*  \* 외부에 함수 노출(doList)  \*/  exports.doList = doList;  /\*\*  \* 첨부파일 목록데이터를 조회한다.  \* @param psAttcFileNo - 첨부파일 번호  \*/  function doList(psAttcFileNo){  app.lookup("dmParam").setValue("strAttcFileNo", psAttcFileNo);    util.Submit.send(app, "subList", null, function(pbSuccess){  if(pbSuccess){    }  });  } |

메인 화면에서는 해당 임베디드앱 객체의 callAppMethod()를 사용하거나, 공통 API인 util.EmbApp.callAppMethod()를 사용하여 해당 앱 내의 특정 노출된 함수를 호출한다.

\*\* 주의사항: 해당 임베디드 내의 함수를 호출할 때, 해당 임베디드앱이 ready 상태가 완료된 이후에야 앱 내의 함수 호출이 가능하다. 따라서, 임베디드앱의 ready 상태와 상관없이 호출하고자 하는 경우에는 공통 API를 사용하여 호출하면 된다.

|  |
| --- |
| util.EmbApp.callAppMethod(app, "embappPsnAddr", "호출 함수명", ["파라메터 값"]); |

■ 그리드 초기화

만약 임베디드앱 화면에 그리드가 들어가는 UI 형태인 경우에, 그리드 초기화를 수행하고자 한다면, embAppHeader(임베디드 헤더)에 그리드ID를 설정한다.

1. 코드 주석작성 방법

코드표준의 주석은 프로그래머 및 유지보수 담당자, 사용자에게 매우 중요한 사항이며, 코드 작성시 각 Tier별 주석 표준을 잘 따르도록 해야 한다.

* 1. UI (eXbuilder6)
* 사용자정의 멤버변수 및 메소드(함수)에 주석을 추가한다.
* 함수 주석의 경우에는 파라메터 인자값에 대한 유형(type) 및 설명을 자세하게 기술한다.
* 만약 인자값 파라메터가 선택인 경우에는 해당 파라메터 주석에 (Optional)을 지정하게 되면, eXbuilder6 디자이너상에 코드 Assist할 때 해당 인자값(Arguments)이 없는 함수를 보여주는 등의 기능을 추가적으로 제공한다.

|  |
| --- |
| /\*\*  \* 메시지 목록데이터를 조회한다.  \* @param {String} psStatus - 조회 상태(저장 후 조회인 경우에는 'save' 구분값 넘김)  \* @return void  \*/  function doList(psStatus){  …  } |

* 1. JAVA

■ 클래스(Class) 주석

- 시스템 : 학사, 행정, 공통

- 단위시스템 : 단위시스템 기술

- 프로그램명 : 프로세서를 호출하는 프로그램명

- 설명 : 해당클래스 요약설명

- 이력사항 : 변경 시 마다 날짜, 개발자, 변경내용 작성

|  |
| --- |
| /\*\*  \* 시 스 템 : 공통  \* 단위시스템 : 공통시스템  \* 프로그램명 : 메시지관리  \* 설 명 : 서버단 오류 및 업무 메시지를 관리한다.  \*  \* 이력사항  \* 2018. 8. 16. 홍길동 최초작성  \*/  @Controller  @RequestMapping("/CmnMessage")  **public** **class** CmnMessageController {  …  } |

■ 메소드(Method) 주석

- 메소드명 : 해당 메소드 이름

- 설명 : 해당메소드 설명

- 이력사항 : 변경 시 마다 날짜, 개발자, 변경내용 작성

- param : argument, argument설명

- return : 리턴 값 타입

- throws : Exception만 기술

|  |
| --- |
| /\*\*  \* 메소드명 : list  \* 설 명 : 메시지 목록 데이터 조회  \*  \* 이력사항  \* 2018. 8. 16. 홍길동 최초작성  \*  \* **@param** request  \* **@param** response  \* **@param** requestData  \* **@return**  \* **@throws** Exception  \*/  @RequestMapping("/list.do")  **public** View list(HttpServletRequest request, HttpServletResponse response, RequestData requestData) **throws** Exception {    } |

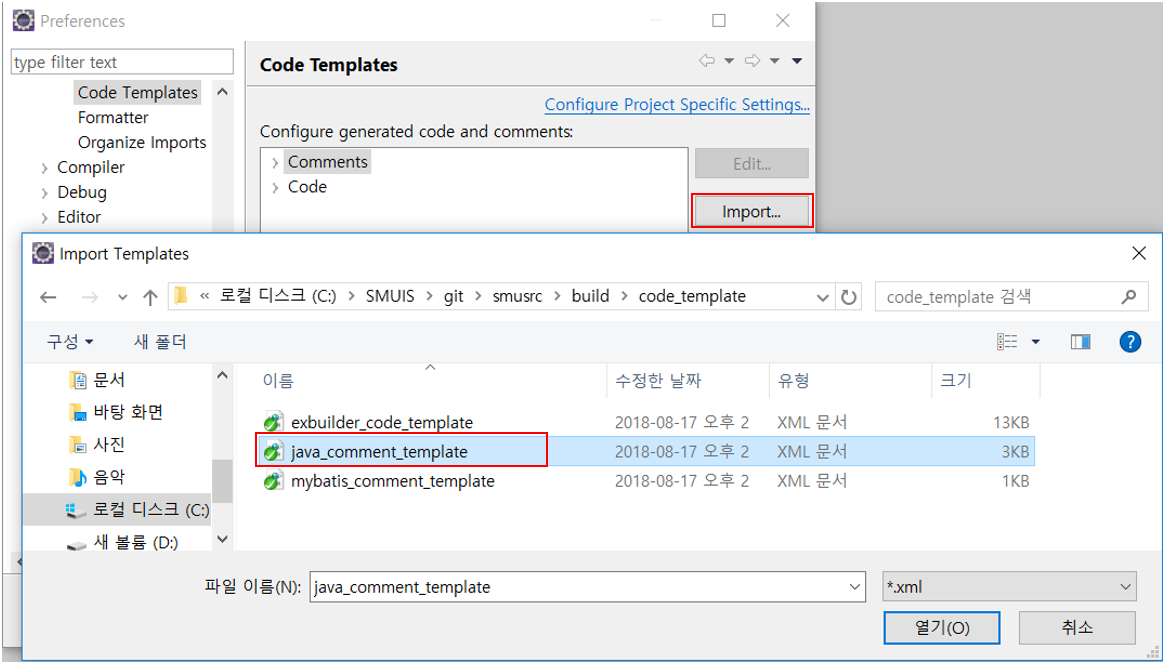
3) JAVA 주석 템플릿 설정 추가 방법

소스코드 작성시 주석을 편리하게 입력하기 위하여 JAVA 주석 템플릿을 설정하여 사용한다.

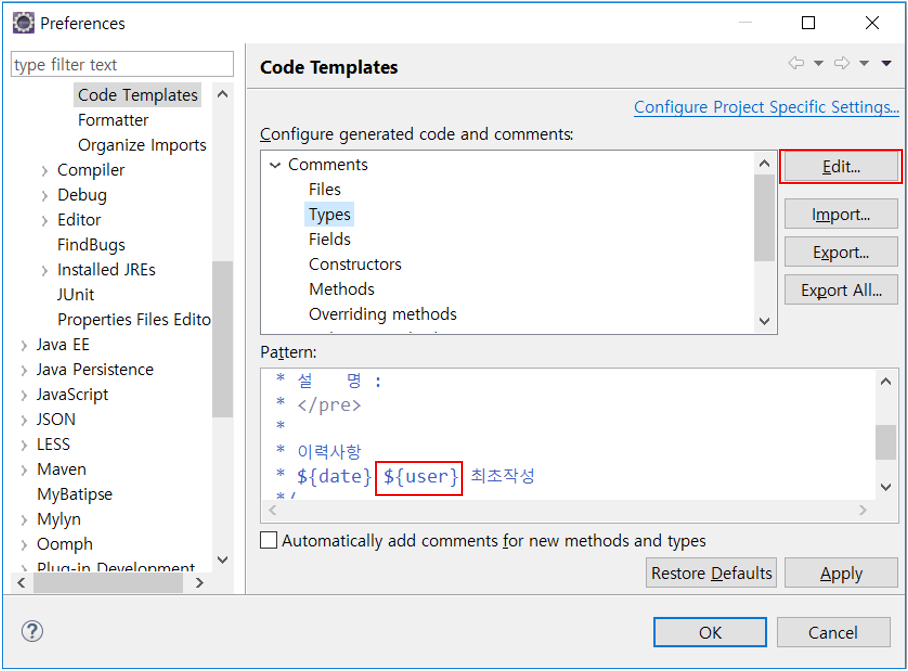
① 이클립스의 메뉴>Window>Preferences>Java>Code Style>Code Templates 선택한다.

② Import버튼을 클릭하여 체크아웃받은 workspace의 build/code\_template 경로의 java\_comment\_template.xml 파일을 선택한 후 [열기] 버튼을 클릭후, [Apply] 버튼을 클릭하여 적용한다.

|  |
| --- |
| 파일경로:build\code\_template\java\_comment\_template.xml |



③ Comments의 Files, Types, Methods 에서 코드 템플릿 확인  
- Types 코멘트를 [Edit] 버튼을 클릭하여 편집모드로 들어간 후, ${user} 부분을 개발자 본인의 이름으로 편집한다.



* 1. SQL 쿼리

실행 쿼리문에 대한 주석 표준은 아래와 같다.

1) 주석 표준

- 설명 : 쿼리에 대한 요약설명

- 주사용처 : 쿼리의 사용처

- 파라미터 : 쿼리에서 사용하는 파라미터 설명

- 변경이력 : 변경 시 마다 날짜, 개발자, 변경내용 작성

* 참고: DB 모니터링 도구등에서 쿼리 속도 분석 및 문제되는 쿼리 분석을 위해, JAVA 유틸에 의해 생성되는 One 테이블 CRUD 쿼리를 제외한 개발자가 직접 작성한 쿼리문에는 반드시 /\* QueryId: 네임스페이스.쿼리ID \*/ 형태의 주석을 추가한다.

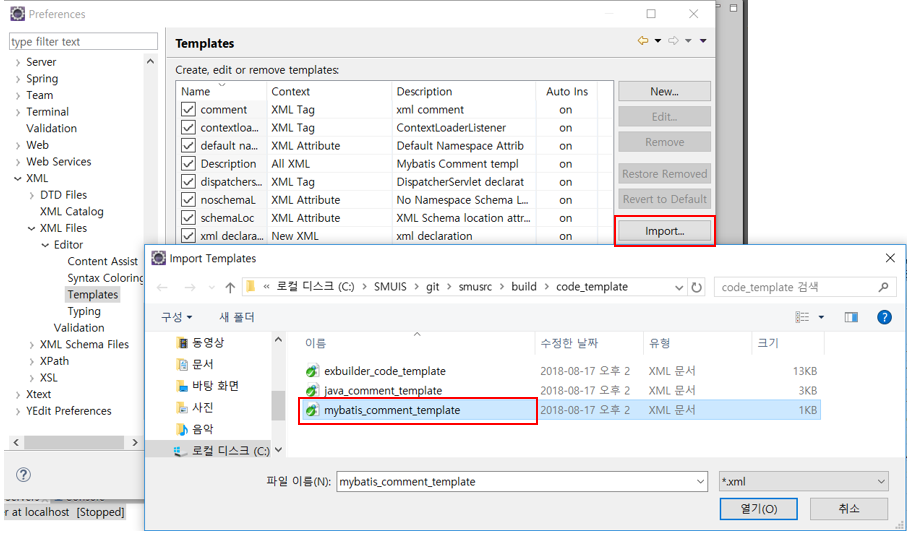
|  |
| --- |
| <!--  설명 : 공통코드 목록을 조회한다.  파라미터 : 코드, 정렬순서  주사용처 : 코드조회  변경이력 : 2018. 8. 24. SUL 최초작성-->  <select id=*"selectCmnCodeList"* resultType=*"HashMap"*>  /\* QueryId: cmn001.selectCmnCodeList \*/  <![CDATA[    ]]>  </select> |

2) SQL XML 주석 템플릿 설정 추가 방법

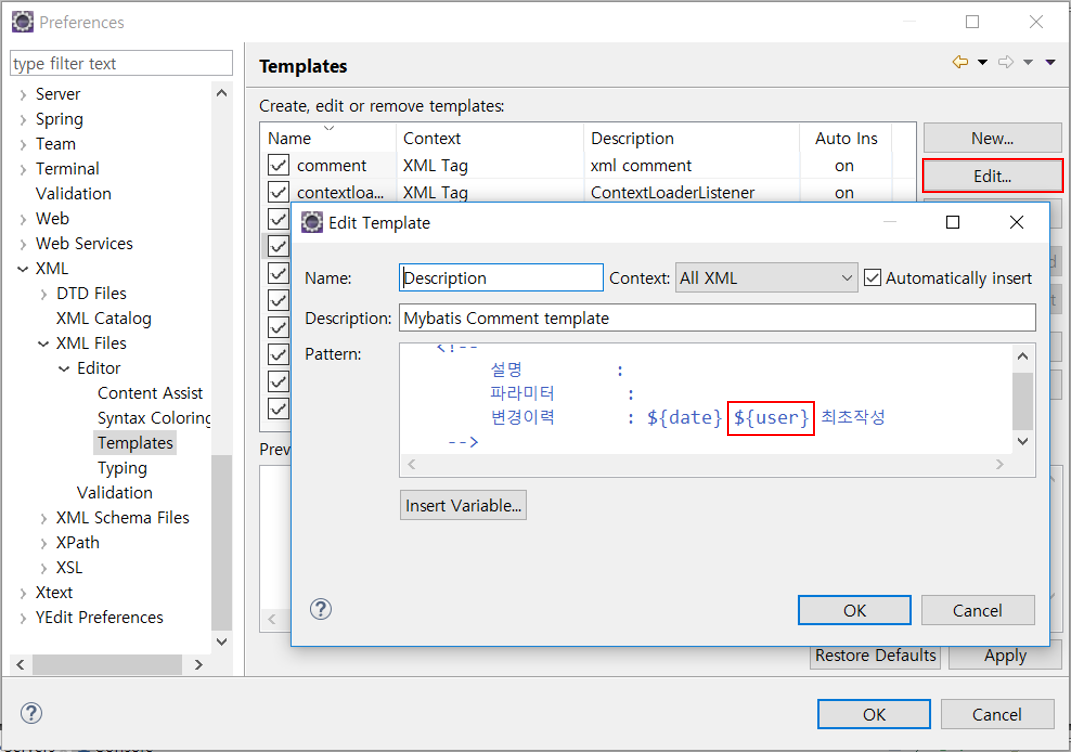
① 이클립스의 메뉴>Window>Preferences>XML>XML Files>Editor>Templates 선택

② Import버튼을 클릭하여 체크아웃받은 workspace의 build/code\_template경로의 mybatis\_comment\_template.xml을 선택한 후, [열기]버튼을 클릭후 [Apply] 버튼을 클릭하여 적용한다.

|  |
| --- |
| 파일경로: C:\SMUIS\git\smusrc\build\code\_template\mybatis\_comment\_template.xml |



③ 추가된 Description 코드 템플릿 확인은 Description를 선택후 [Edit] 버튼을 클릭하여 편집모드로 들어간 후, ${user} 부분을 개발자 본인의 이름으로 편집한다.



1. 서버단 코드 작성

전자정부 프레임워크 기반의 서버단 자바(JAVA) 소스 코드 작성 규칙은 아래와 같다.

서버단 소스는 크게 컨트롤러(Controller), 서비스(Service), SQL Query 파일로 구성되어 있다.

■ 소스 작성 기본 규칙

|  |  |
| --- | --- |
| 구분 | 내용 |
| Controller | * 컨트롤러(Controller)는 CUD가 발생하는 입력/일괄 화면에 대해 1개의 컨트롤러를 구현한다. * 조회, 출력, 조회용 팝업 화면에 대한 컨트롤러를 구현하지 않는다. * CUD 작업이 발생하는 팝업 화면에 대해서는 별도의 컨트롤러를 생성하지 않고, 해당 팝업의 메인이 되는 컨트롤러 클래스에 관련 로직을 구현한다. |
| Service | * 서비스(Service)는 CUD가 발생하는 DB 테이블당 1개씩 구현한다. * 서비스는 DB 트랜잭션 처리 단위이며, 인터페이스와 구현체(Impl)을 구현한다. |
| Query XML | * 파일명은 기본적으로 중메뉴 단위로 분리한다. * SQL 파일내의 작성한 쿼리가 5,000 라인을 넘어가는 경우 또는 개발자가 동일 업무파트를 같이 개발해야되는 경우에 한하여 파일을 분리하여 관리할 수 있다. 단, 분리할 경우에는 분리된 파일부터 파일명에 순번[2자리]를 준다. (cmn-base01.xml, cmn-base01.xml, cmn-base03.xml 등등) * 자바 유틸에 의해 생성되는 원 테이블들에 대한 CUD 쿼리는 tables 접미어를 붙인 파일에 추가 관리한다. (cmn-tables.xml, cmn-tables01.xml) |

* 1. 컨트롤러 클래스 작성
* 컨트롤러 클래스의 역할은 사용자 요청(URL 매핑)에 대한 요청 파라메터를 처리하고, 해당 로직을 처리하는 서비스를 호출하는 역할을 담당한다. 기본적으로 컨트롤러에는 비즈니스 로직을 처리하지 않는다. 단, 파일업로드 또는 엑셀 파일 Export 등의 처리인 경우에는 컨트롤러에 비즈니스 로직이 들어갈 수도 있다.

|  |
| --- |
| /\*\*  \* 시 스 템 : 공통  \* 단위시스템 : 공통시스템  \* 프로그램명 : 메시지관리  \* 설 명 : 서버단 오류 및 업무 메시지를 관리한다.  \*  \* 이력사항  \* 2018. 8. 16. SUL 최초작성  \*/  @Controller  @RequestMapping("/CmnMessage")  **public** **class** CmnMessageController {  } |

■ Controller 클래스 구성

① Controller 스캔이 가능하도록 클래스 위에 @Controller Annotation 추가

② UI단에서 요청 서브미션 액션 URL에 매핑되는 @RequestMapping URL 추가

|  |
| --- |
| @Controller  @RequestMapping("/CmnMessage")  **public** **class** CmnMessageController {  } |

③ 사용할 Service 위에 @Autowired Annotation 추가 (의존성 주입)

|  |
| --- |
| @Controller  @RequestMapping("/CmnMessage")  **public** **class** CmnMessageController {    @Autowired  **private** CmnCodeService cmnCodeService;  } |

④ 메소드명 규칙

|  |  |  |
| --- | --- | --- |
| 역할 | 메소드명 | 내용 |
| 초기화 메소드 | onLoad() | 화면 초기화에 필요한 데이터를 가져오기 위한 경우에 사용  Ex) 화면단 onLoad() 이벤트 내에서 호출 |
| 조회 메소드 | list(), listXXX() | 데이터 조회를 처리하기 위한 메소드 |
| 저장 메소드 | save() | 데이터 저장 로직을 처리하기 위한 메소드 |

**※ @RequestMapping URL Command와 Method명은 일치시킨다.**

|  |
| --- |
| /\*\*  \* 메소드명 : onLoad  \* 설 명 : 공통코드 및 화면초기화에 필요한 데이터 반환  \*  \* 이력사항  \* 2018. 8. 16. SUL 최초작성  \*  \* **@param** request  \* **@param** response  \* **@param** requestData  \* **@param** authentication  \* **@return**  \* **@throws** Exception  \*/  @RequestMapping("/onLoad.do")  **public** View onLoad(HttpServletRequest request, HttpServletResponse response, RequestData requestData) **throws** Exception {  String strLocale = requestData.getAppLocale();  //단위시스템 코드  requestData.setResponse("dsUnitSystemRcd", cmnCodeService.selectCmnCodeList("CMN003", strLocale));  //언어구분 코드  requestData.setResponse("dsLangDivRcd", cmnCodeService.selectCmnCodeList("CMN006", strLocale));  //메시지구분 코드  requestData.setResponse("dsMsgDivRcd", cmnCodeService.selectCmnCodeList("CMN037", strLocale));  **return** requestData.getDefaultView();  }    /\*\*  \* 메소드명 : list  \* 설 명 : 메시지 목록 데이터 조회  \*  \* 이력사항  \* 2018. 8. 16. SUL 최초작성  \*  \* **@param** request  \* **@param** response  \* **@param** requestData  \* **@param** authentication  \* **@return**  \* **@throws** Exception  \*/  @RequestMapping("/list.do")  **public** View list(HttpServletRequest request, HttpServletResponse response, RequestData requestData) **throws** Exception {  //요청 파라메터 셋팅  Map<String, String> mapParam = **new** HashMap<String, String>();  mapParam.put("UNIT\_SYSTEM\_RCD", requestData.getString("strUnitSystemRcd"));  mapParam.put("MSG\_DIV\_RCD", requestData.getString("strMessageDivCd"));  mapParam.put("MSG\_TEXT", requestData.getString("strMessage"));    requestData.setResponse("dsMessage", cmnMessageService.selectMessageList(mapParam));  **return** requestData.getDefaultView();  }    /\*\*  \* 메소드명 : save  \* 설 명 : 메시지 데이터 저장  \*  \* 이력사항  \* 2018. 8. 16. SUL 최초작성  \*  \* **@param** request  \* **@param** response  \* **@param** requestData  \* **@param** authentication  \* **@return**  \* **@throws** Exception  \*/  @RequestMapping("/save.do")  **public** View save(HttpServletRequest request, HttpServletResponse response, RequestData requestData) **throws** Exception {    cmnMessageService.saveMessage(requestData);    **return** requestData.getDefaultView();  } |

* 1. 서비스 작성
* 업무에 대한 비즈니스 로직은 서비스 클래스에서 구현해야 한다..
* **~~DB 트랜잭션은 서비스 메소드 단위로 처리된다. 따라서, 여러 업무 처리를 하나의 트랜잭션으로 묶어서 처리하기 위해서는 반드시 서비스 클래스에서 비즈니스 로직을 처리해야 한다.~~**
* ~~서비스 클래스 구현시에는 AOP를 이용한 DB 트랜잭션 처리를 위해 반드시 인터페이스를 생성하여야 한다. 이는 스프링의 AOP 처리 매커니즘에서 해당 인터페이스에 대한 프록시 클래스를 생성하도록 하기 위함이다~~.
* 서비스는 기본적으로 CUD가 발생하는 테이블당 매칭되는 1개의 서비스 클래스를 작성하는 것을 원칙으로 한다.
* 저장 메소드내에서 데이터 저장 순서는 삭제 🡪 신규 🡪 업데이트 순으로 처리한다.
* DB테이블 명칭은 [단위업무시스템 코드{3}]+순번 이지만, 해당 테이블에 대한 서비스 클래스 명칭은 의미있는 영문을 사용한다.

(ex: CmnMessageService, CmnDeptService 등)

■ Service 메소드

|  |  |  |
| --- | --- | --- |
| 역할 | 메소드명 | 내용 |
| 단건 조회 메소드 | **select**XXX(), count() | 단건 데이터 조회를 처리하기 위한 메소드 |
| 목록성 조회 메소드 | **select**XXX**List**() | 다건의 목록성 데이터 조회를 처리하기 위한 메소드 |
| 저장 메소드 | **save**XXX()  **insert**XXX()  **update**XXX()  **delete**XXX() | 데이터 저장 로직을 처리하기 위한 메소드 |

■ Service 클래스 구성

① 구현 부분을 개발하기 위한 공통 Service 상속

|  |
| --- |
| package org.tmt.iis.cmn.service;  @Service  public class **CmnMessageService** extends **AbstractService** {  @Override  public List<Map<String, Object>> selectMessageList(Map<String, String> mapParam) throws Exception {  return dao.selectList("cmn001.selectMessageList", mapParam);  }  } |

② Service 스캔이 가능하도록 클래스 위에 @Service Annotation 추가

③ 해당하는 서비스 클래스를 implements 한다.

|  |
| --- |
| @Service  **public** **class** CmnMessageService **extends** AbstractService {    **public** List<Map<String, Object>> selectMessageList(Map<String, String> mapParam) **throws** Exception {  **return** dao.selectList("cmn001.selectMessageList", mapParam);  }    **public** **void** saveMessage(RequestData requestData) **throws** Exception {  ParameterGroup dsMessage = requestData.getParameterGroup("dsMessage");  Iterator<ParameterRow> insertedRows = dsMessage.getInsertedRows();  Iterator<ParameterRow> updatedRows = dsMessage.getUpdatedRows();  Iterator<ParameterRow> deletedRows = dsMessage.getDeletedRows();    **while**(deletedRows.hasNext()){  dao.delete("cmn-tables01.deleteCmn0301", deletedRows.next().toMap());  }    **while**(insertedRows.hasNext()){  dao.insert("cmn-tables01.insertCmn0301", insertedRows.next().toMap());  }    **while**(updatedRows.hasNext()){  dao.update("cmn-tables01.updateCmn0301", updatedRows.next().toMap());  }  }  } |

* 1. 쿼리 작성
* 데이터등록, 수정, 삭제를 할 실제 query를 갖고 있는 파일이다.
* **SQL의 ID는 서비스(Service) 클래스의 메소드 명과 동일하게 작성한다.**
* SQL은 반드시 대문자를 사용한다. (SELECT 쿼리시 조회하는 컬럼을 소문자로 기술하는 경우에는 화면단에 반환되는 데이터 컬럼 역시 소문자로 반환되므로 주의가 필요하다.)
* SQL 문자열에 특수문자가 포함되는 경우 CDATA 태그로 감싼다.(이는 XML 파싱 도중 특수문자로 인해 SQL문자열을 추출하지 못하는 것을 방지하기 위함이다.)
* 조회 쿼리(SELECT 절)에는 반드시 첫 줄에 해당 쿼리 ID를 주석으로 추가한다.
* SELECT절 쿼리의 정렬은 좌측정렬을 기본으로 한다.
* WHERE절에 1 = 1와 같은 구문은 사용하지 않는다.
* ANSI-SQL 문법을 사용하지 않는다. (LEFT OUTER JOIN, INNER JOIN 등)
* SQL 인젝션 방지를 위해, 쿼리문에 치환(replace) 표현식 사용은 금지한다.

|  |
| --- |
| <select id=*"getAacCreditCardList"* parameterType=*"HashMap"* resultType=*"HashMap"*>  /\* QueryId: aac-base01.getAacCreditCardList \*/  <![CDATA[  SELECT ACCT\_UNIT\_RCD /\*회계단위\*/  , CARD\_NO /\*카드번호\*/  , CARD\_NM /\*카드명\*/  , CARD\_DIV\_RCD /\*카드구분\*/  , CARD\_COMP\_RCD /\*카드회사\*/  , EFFT\_YM /\*유효년월\*/  , LIMIT\_AMT /\*사용한도액\*/  , SETL\_BANK\_RCD /\*결제은행코드\*/  , SETL\_ACCT\_NO /\*결제계좌번호\*/  , SETL\_DD /\*월결제일자\*/  , USE\_YN /\*사용여부\*/  , PRT\_ORD /\*출력순서\*/  , REMARK /\*비고\*/  , ACCT\_HLD /\*예금주\*/  FROM AAC2103  WHERE ACCT\_UNIT\_RCD = #{strAcctUnitRcd}#  ]]>  <if test=*"strUseYn != null and strUseYn != ''"*>  AND NVL(USE\_YN, 'N') = #{strUseYn}  </if>  <if test=*"strCardNo != null and strCardNo != ''"*>  AND CARD\_NO LIKE '%' || #{strCardNo} || '%'  </if>  <![CDATA[  ORDER BY CARD\_NO  ]]>  </select> |

■ Mybatis SQL Syntax

기본적으로 MyBatis에서 제공하는 문법(Syntax)는 아래와 같다. 추가적인 문법들은 MyBatis 매뉴얼을 참고한다.

※ SQL-Injection 방지를 위해서, 쿼리문에서 치환 문법(${variable}) 사용은 금지한다.

|  |  |
| --- | --- |
| 구분 | Syntax |
| 변수 | #{ *variable* }사이에 변수 지정 |
| 동적 변수 | <if test=”*condition*”> 사용 |
| 동적 쿼리 Include | <include> 태그 사용 |
| 분기처리 | <choose><when/><otherwise/></choose> 태그 사용 |
| 반복 | <foreach></foreach> 태그 사용 |

■ CLOB 데이터 처리

CLOB 데이터를 조회하는 경우에는 아래와 같이 resultMap을 선언한 후 사용하면 된다.

|  |
| --- |
| <resultMap id=*"cmnInfoDocListForAll\_Result"* type=*"java.util.HashMap"* >  <result property=*"CONT"* column=*"CONT"* javaType=*"java.lang.String"* jdbcType=*"CLOB"* />  </resultMap>  <select id=*"selectCmnInfoDocListForAll"* parameterType=*"HashMap"* resultMap=*"cmnInfoDocListForAll\_Result"*>  <![CDATA[  SELECT MSG\_DIV\_RCD  , F\_CMN\_CODE\_NM( MSG\_DIV\_RCD ) AS MSG\_DIV\_RCD\_NM  , UNIT\_SYSTEM\_RCD  , F\_CMN\_CODE\_NM( UNIT\_SYSTEM\_RCD ) AS UNIT\_SYSTEM\_RCD\_NM  , CONT -- CLOB 데이터  FROM CMN0105  ]]>  <where>  <if test=*" MSG\_DIV\_RCD\_NM != null and MSG\_DIV\_RCD\_NM != '' "*>  AND F\_CMN\_CODE\_NM(MSG\_DIV\_RCD) LIKE #{MSG\_DIV\_RCD\_NM}||'%'  </if>  <if test=*" UNIT\_SYSTEM\_RCD != null and UNIT\_SYSTEM\_RCD != '' "*>  AND UNIT\_SYSTEM\_RCD = #{UNIT\_SYSTEM\_RCD}  </if>  </where>  </select> |

■ 프로시저 호출

- 프로시저 호출 SQL 쿼리 작성

|  |
| --- |
| <select id=*"procCmnTlgTriggerCreate"* statementType=*"CALLABLE"* parameterType=*"HashMap"* resultType=*"HashMap"*>  <![CDATA[  { call P\_CMN\_TLG\_TRIGGER\_CREATE(#{strQuery, mode=IN, jdbcType=VARCHAR}) }  ]]>  </select> |

① <select> 절에 작성해야 한다.

② <select>절의 statementType 속성을 CALLABLE로 지정해야 한다.

③ 프로시저에 넣어주는 인자값은 변수로 해당 인자값 변수, IN/OUT 유형, 데이터 유형등을 지정한다.

* 1. 시큐어 코딩

SW 개발과정에서 개발자 실수, 논리적 오류 등으로 인해 SW에 내포될 수 있는 보안취약점의 원인, 즉 보안약점을 최소화할 수 있도록 소소코드 구현단계에서 보안약점을 배제하기 위한 자주 실수하는 기본적인 JAVA 시큐어 코딩을 적용한다.

■ 적절하지 않은 예외처리

프로그램 수행 중에 함수의 결과 값에 대한 적절한 처리 또는 예외상황에 대한 조건을 적절하게 검사하지 않을 경우, 예기치 않은 문제를 야기 할 수 있다.

- 대응 방법

발생할 수 있는 예외들에 대해 적절한 예외 처리를 수행한다.

※ try ~ catch문 처리시, catch절에 Exception만 사용하면 안되며, 적절한 구체적인 예외를 1개 이상 기술해야 한다. (만약, 적절한 예외를 찾을 수 없는 경우에는 AppWorksException을 기술해도 된다.)

< 보안 취약한 예시 >

|  |
| --- |
| public void readFromFile(String fileName){  try{  File myFile = new File(fileName);  FileReader fr = new FileReader(myFile); // bad    // 광범위한 예외 클래스인 Exception을 사용하여 예외처리  } catch(Exception ex){ }  } |

< **안전한 보안 코딩 예시** >

|  |
| --- |
| public void readFromFile(String fileName) throws FileNotFoundException,  IOException, MyException {  try{  //Null check for fileName  if(fileName == NULL) throw new MyException(“error”);  File myFile = new File(fileName);  FileReader fr = new FileReader(myFile);  //발생할 수 있는 예외별 처리 수행  } catch(FileNotFoundException fe){ logger.debug(“파일을 찾을 수 없습니다.”) }  } catch(IOException ie){ logger.debug(“파일을 읽는 도중 오류가 발생했습니다.”); }  } catch(Exception e){ logger.debug(“오류가 발생하였습니다.”); }  } |

■ 오류메시지 통한 정보 노출

시스템이나 디버깅 정보를 드러내는 것은 악의적인 공격을 계획하기 쉽게 만든다. 즉, 시스템이나, 디버깅 정보는 output stream 이나 logging 기능을 통하여 새어나갈 가능성이 높다.

- 대응 방법

전체 Java 스택 정보가 노출되지 않도록 필요한 에러 메시지만 로깅처리한다.

※ JAVA에서 로깅처리는 System.out.println을 사용하면 안되며, 반드시 logger 클래스를 사용해 로깅을 처리해야 한다. 이는 try ~ catch 구문에서만 한정되지 않으며, JAVA 소스 전구간에서 로깅처리시 logger를 사용한다. (ServiceImpl 클래스에서는 상위 클래스에 정의된 logger를 사용하면 되며, Controller 클래스에서는 필요시 Logger 클래스를 선언해서 사용하면 된다.)

< 보안 취약한 예시 >

|  |
| --- |
| try {  ...  }catch (Exception e) {  //스택 정보가 사용자에게 노출될 가능성이 높다.  e.printStackTrace();  } |

< **안전한 보안 코딩 예시** >

|  |
| --- |
| try {  ...  }catch (AppWorksException e) {  //전체 스택정보를 출력하지 않고, 필요한 에러 정보만 로깅 처리한다.  logger.debug(e.getMessage());  } |

<Controller 로거 선언 예시>

|  |
| --- |
| @Controller  @RequestMapping("/CmnMessage")  **public** **class** CmnMessageController {  //Logger 선언  **private** Logger logger = LoggerFactory.*getLogger*(**this**.getClass());  } |

■ 오류 상황 대응 부재

try, catch 블록에서 catch 블록이 비어 있는 경우는 예외처리 코드가 포함되어 있지 않기 때문에, Program 실행 시 오류가 발생했을 경우, 원인을 파악하기가 쉽지 않다.

- 대응 방법

try, catch 블록으로 예외 처리시, 발생할 수 있는 예외들에 대해 적절한 처리를 수행해야 한다.

※ try ~ catch문 사용시, catch 문을 비워두면 안되며 반드시 적절한 예외 처리 또는 오류 메시지 로깅처리를 기술해야 한다.

< 보안 취약한 예시 >

|  |
| --- |
| private Connection conn;  public Connection DBConnect(String url, String id, String password) {  try {  DataSource datasource = (DataSource) ctx.lookup(CONNECT\_STRING);  conn = datasource.getConnection();  } catch (SQLException e) {  // Catch block is empty  } catch (NamingException e) {  // Catch block is empty  }  return conn;  } |

< **안전한 보안 코딩 예시** >

|  |
| --- |
| private Connection conn;  public Connection DBConnect(String url, String id, String password) {  try {  DataSource datasource = (DataSource) ctx.lookup(CONNECT\_STRING);  conn = datasource.getConnection();  } catch (SQLException e) {  // Proper Exception handling  if ( conn != null ) {  try {  conn.close();  } catch (SQLException e1) {  conn = null;  }  }  } catch (NamingException e) {  // Proper Exception handling.  if ( conn != null ) {  try {  conn.close();  } catch (SQLException e1) {  conn = null;  }  }  }  return conn;  } |

■ 부적절한 자원 해제

특정 자원이 할당이 되었지만, Java Virtual Machine의 garbage collector에 의해서 빠르게 자원회수가 되지 않을 경우에 Resource leak이 발생할 수 있다. 대부분의 Heap memory와 관련된 메모리 관리는 JVM이 책임지지만, Socket, Stream, Channel과 같은 자원은 그렇지 않다. 그러므로 이러한 자원은 프로그램 작성자가 적절히 해제를 해주는 코드를 작성하여야 하며, 그렇지 못할 경우에는 성능저하, 시스템기능의 멈춤, DOS(Denial of Service), 또 다른 자원을 획득함에 있어서의 실패를 야기할 수 있다.

- 대응 방법

Socket, Stream, Channel과 같은 자원 사용시 사용 후에는 반드시 해당 자원을 시스템에 반납하는 코딩을 작성해주어야 한다. 자원 해제는 반드시 예외가 발생해도 자원이 반환되도록 finally 블럭에서 선언해주어야 한다.

※ 파일(File)을 읽고 쓰는 작업 및 Socket 사용한 처리시에는 반드시 finally 블록에서 획득한 자원을 해제해주어야 한다.

< 보안 취약한 예시 >

|  |
| --- |
| public class ResourceLeakEx {  public void testResourceLeak() {  try {  BufferedWriter out = new BufferedWriter(new FileWriter(  new File("test.txt")));  out.write("This is Resource Leak sample code…");  out.newLine();  } catch (IOException e) {  // …  }  } /\* BUG : RESOURCE\_LEAK \*/  } |

< **안전한 보안 코딩 예시** >

|  |
| --- |
| public class ResourceLeakSafeEx {  public void testResourceLeak() {  try {  BufferedWriter out = new BufferedWriter(new FileWriter(  new File("test.txt")));  out.write("This is Resource Leak sample code…");  out.newLine();  } catch (IOException e) {  // …  } finally {  //예외가 발생하더라도 자원이 반환되도록 처리한다.  if (out != null) {  out.close(); /\* SAFE \*/  }  }  }  } |

■ SQL 인젝션

웹 응용 프로그램에 강제로 구조화 조회 언어(SQL) 구문을 삽입하여 내부 데이터베이스(DB) 서버의 데이터를 유출 및 변조하고 관리자 인증을 우회할 수도 있다. 이 공격은 MS SQL 서버뿐만 아니라 모든 관계형 데이터베이스 관리 시스템(RDBMS)에서 가능하다.

- 대응 방법

클라이언트에서 넘어오는 파라메터들을 이용하여 쿼리 생성하지 않도록 가이드하며, 실행되는 쿼리들에 대해 PreparedStatement 문법을 사용하도록 한다.

※ Mybatis 쿼리 작성시 가급적 동적 치환변수(${XXX}) 사용을 금지하며, 해당 쿼리의 경우에는 <choose> 또는 <foreach> 등 다른 문법을 사용하여 대체하여야 한다.

< 보안 취약한 예시 >

|  |
| --- |
| String userid = request.getParameter("userid");  String password = request.getParameter("password");    Statement stmt = conn.createStatement();  //입력받은 문자열을 조합하여 쿼리를 실행하는 경우  ResultSet rs =     stmt.executeQuery("select count(\*) as cnt from member where userid='"+userid+"' and password='"+password+"'");  //만약 userid 값이 admin' OR '1'='1 이고, password값이 abcd로 입력되었다면  //select count(\*) from member where userid='amdin' OR '1'='1' and password = 'abcd' 쿼리가 실행되어 타 계정(ex: admin)으로 로그인이 가능하게 된다. |

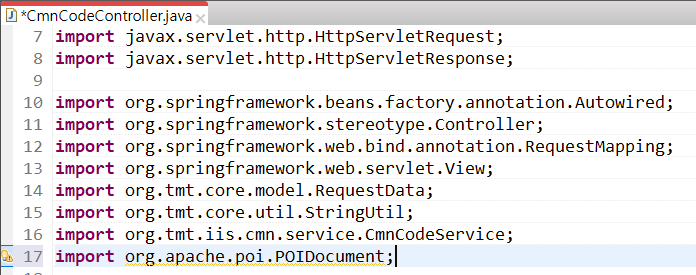
< **안전한 보안 코딩 예시** >

|  |
| --- |
| String userid = request.getParameter("userid");  String password = request.getParameter("password");  //PreparedStatement API를 사용해 사용자의 입력값들을 변수 처리되도록 함  PreparedStatement stmt = conn.prepareStatement("select count(\*) from member where userid=? and password=?");  stmt.setString(1, userid);  stmt.setString(2, password);  ResultSet rs = stmt.executeQuery(); |

■ 불필요한 소스 참조

자바 클래스 파일의 경우, 소스 상단에 임포트(import)되는 JAVA 라이브러리 중에 해당 소스에서 참조하지 않는 불필요한 라이브러리들이 존재하는 경우에 삭제한다.

<예시>



* org.apache.poi.POIDocument는 해당 소스에서 참조하지 않음

이클립스 에디터창에서 Ctrl + Shift + O 를 눌러 불필요한 참조를 제거한다.

* 1. 대용량 데이터 처리

대용량 데이터를 조회하는 경우에 한하여, 서버 자원 과사용 방지 및 화면 데이터 응답속도 개선을 위해 적용한다.

■ 적용대상

데이터 최대 조회 건수: 2만(20,000)건 이상

* 화면에 필수입력 조건만 넣고, 조회한 최대 건수를 기준으로 한다.
* 만약, 화면상에 조회기간(from ~ to)이 있는 경우에는 1차적으로는 기간 제한조건이 포함되어야 하며(예- 6개월치만 조회가능 등), 최대 조회기간으로 조회한 데이터 건수가 2만건이 넘으면 적용 대상이 된다.
* 기본은 데이터 건수가 2만건 이상이며, 1만인 경우에도 데이터 렌더링을 빠르게 하겠다고 하면 적용해도 무방하다.

■ 응답유형별 특징

대용량 데이터는 두가지 포맷으로 데이터를 응답처리할 수 있다.

JSON: 데이터를 JSON 포맷의 문자열로 생성

* 서버로부터 응답데이터인 JSON 문자열이 전부 내려오기 전까지는 화면단에서 eXbuilder6 라이브러리가 대기하고 있다가, 전체 데이터가 내려온 이후에 화면에 데이터를 렌더링을 시작하게 된다.

TSV: 데이터에 대한 헤더정보(1라인)와 데이터로 구성되어 있는 문자열 포맷을 의미한다.

* 서버로부터 전체 응답데이터가 내려오지 않았더라도, 서버로부터 내려온 데이터가 있으면 eXbuilder6는 곧바로 화면에 렌더링을 시작한다.
* JSON 보다는 데이터에 대한 노드 정보를 1건만 가지고 있기 때문에 응답데이터의 용량이 줄어들어 네트워크 부하를 줄여준다.
* 데이터내의 각 항목은 탭(Tab) 문자로 구분된다.

■ 제약사항

응답유형으로 TSV 포맷을 사용하고자 하는 경우에는 아래와 같은 제약사항이 존재한다.

* TSV는 하나의 응답만 처리 가능하다. 즉, 하나의 요청 액션에 여러 데이터셋 데이터를 한꺼번에 내려받을 수 없다.

■ 적용규칙

* 기본적으로는 대용량 조회 화면인 경우에 적용한다.
* 데이터가 2만건 이상인 경우 적용한다.
* 하나의 요청에 응답이 하나인 경우에는 응답데이터 유형을 TSV로 처리되도록 적용한다.
* 하나의 요청에 응답이 2개 이상인 경우에는 응답데이터 유형을 JSON으로 처리되도록 적용한다.
* 쿼리(SQL)에서 fetchSize를 적절하게 지정한다.

■ 적용방법

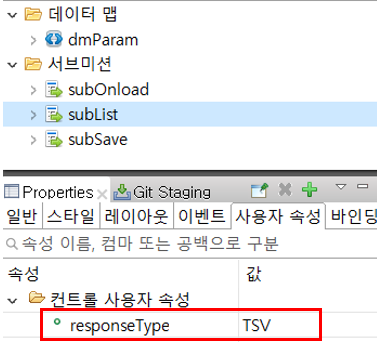
1) 서버단 컨트롤러(Controller) 클래스에서 공통 API를 통해 대용량 데이터 조회하도록 처리

2) 쿼리 조회시 DB로부터 리턴받는 Fetch 건수를 늘려주도록 처리

3) 서버 응답데이터 유형이 TSV인 경우에는, 화면단 UI의 서브미션(Submission) 컨트롤의 사용자 정의 속성인 responseType을 TSV로 지정한다.

<화면 파트>

응답 데이터 유형이 TSV인 경우에는 프로그래스바에 중간중간 서버로부터 내려받은 데이터 건수를 보여줄 필요가 있다. 따라서, TSV응답인 경우에는 화면단의 서브미션 컨트롤의 사용자 정의 속성을 TSV로 지정해야 한다.



<쿼리 파트>

Mybatis ORM의 경우에는 쿼리 조회 결과를 DB에서 Fetch 하는 건수를 지정할 수 있다. 기본 JDBC의 패치 건수는 10건 이다. 대용량 데이터를 조회하는 경우에는 DB로부터 Fetch하는 건수를 늘려서, 쿼리한 결과(ResultSet)를 가져오기 위해서 DB에 Fetch 요청 횟수를 줄여줄 수 있다.

적절한 Fetch 사이즈는 쿼리마다 다를 수 있으며, 일반적으로 아래 범위로 지정하면 된다.

주의사항: Fetch 크기를 무작정 늘린다고 해서 속도가 빨라지는 것은 아니며, 너무 크게 설정한 경우에는 리소스 낭비를 일으킬 수 있으니 주의가 필요한다.

|  |
| --- |
| FetchSize는 100 ~ 1,000 사이를 지정 |

Fetch 사이즈 관련 참고자료: <http://wiki.gurubee.net/pages/viewpage.action?pageId=3901810>

<쿼리 예시>

|  |
| --- |
| <!--  설명 : 구성원 목록 조회  파라미터 : 구성원구분/구성원ID/성명  주사용처 : 구성원관리  변경이력 : 2019. 1. 14. 고수현 최초작성  -->  <select id=*"selectPartyList"* parameterType=*"HashMap"* resultType=*"HashMap"* **fetchSize**=*"100"*>  /\* QueryId : cmn-user01.selectPartyList \*/  <![CDATA[  SELECT PARTY.PARTY\_ID  ,F\_CMN\_RESNO\_VIEW(#{RESNO\_VIEW\_YN},PARTY.RESNO) AS RESNO  ,F\_CMN\_RESNO\_VIEW(#{RESNO\_VIEW\_YN},PARTY.RESNO) AS RESNO\_ORI  ,PARTY.NM\_KOR  ,PARTY.NM\_ENG  ,PARTY.NM\_CHA  ,PARTY.GENDER\_RCD  ,PARTY.BRTDY  ,PARTY.LNR\_SLR\_DIV\_RCD  ,PARTY.NAT\_RCD  ,PARTY.WEDDING\_DIV\_RCD  FROM CMN2301 PARTY  ]]>  <where>  <!-- 구성원-미지정 -->  <if test=*"strUserDivCd != null and strUserDivCd != '' and strUserDivCd == 'NONE'"*>  AND PARTY.PARTY\_ID IN (SELECT PARTY\_ID FROM CMN2301 EXCEPT SELECT PARTY\_ID FROM V\_CMN2301\_INGT)  </if>  <if test=*"strUserDivCd != null and strUserDivCd != '' and strUserDivCd != 'NONE'"*>  AND EXISTS (SELECT 1 FROM V\_CMN2301\_INGT V WHERE V.PARTY\_ID = PARTY.PARTY\_ID AND V.USER\_DIV\_CD = #{strUserDivCd})  </if>  <if test=*"strPartyId != null and strPartyId != ''"*>  AND UPPER(PARTY.PARTY\_ID) = UPPER(#{strPartyId})  </if>  <if test=*"strNm != null and strNm != ''"*>  AND (UPPER(PARTY.NM\_KOR) LIKE '%'|| UPPER(#{strNm}) ||'%' OR  UPPER(PARTY.NM\_ENG) LIKE '%'|| UPPER(#{strNm}) ||'%' OR  UPPER(PARTY.NM\_CHA) LIKE '%'|| UPPER(#{strNm}) ||'%' )  </if>  <!-- 구성원-미지정 -->  <if test=*"strPartyDiv != null and strPartyDiv != '' and strPartyDiv == 'NONE'"*>  AND PARTY.PARTY\_ID IN (SELECT PARTY\_ID FROM CMN2301 EXCEPT SELECT PARTY\_ID FROM V\_CMN2301\_INGT)  </if>  </where>  ORDER BY PARTY\_ID  </select> |

<공통 API 사용 파트>

클래스: MassReponseService

|  |  |
| --- | --- |
| 메소드 | 비고 |
| selectDataToJSONStream() | 대량 데이터를 JSON 문자열 포맷으로 만들어서 응답 스트림에 내려보내다. |
| selectDataToTSVStream() | 대량 데이터를 TSV 문자열 포맷으로 만들어서 응답 스트림에 내려보내다. |
| exportExcelToStream() | 데이터를 엑셀파일로 만들어서 응답 스트림에 내려보내다. |

① Controller 클래스 상단에 MassResponseService 서비스 Authowired 선언

② 요청 처리 메소드의 return 타입은 void로 선언

③ MassResponse 서비스 메소드를 호출하여 데이터 조회

<JSON 응답처리 예시>

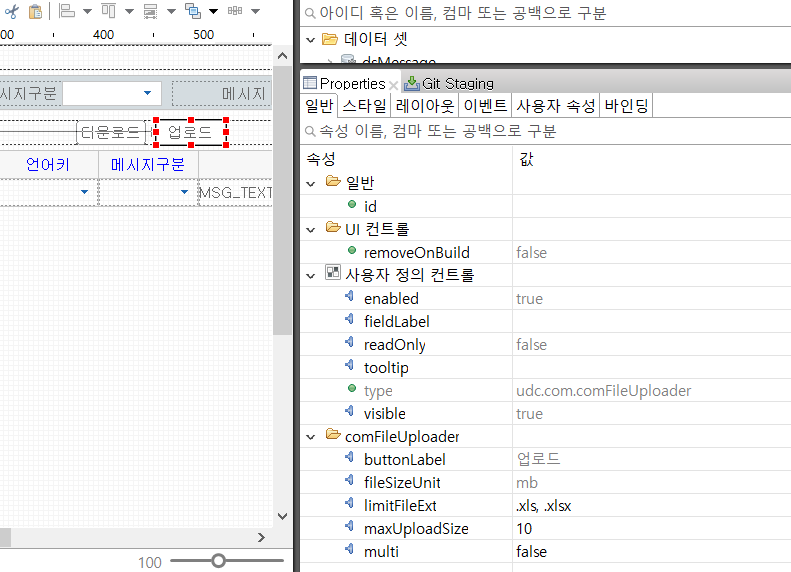
|  |
| --- |
| @Controller  @RequestMapping("/CmnParty")  **public** **class** CmnPartyController {  @Autowired  **private** MassResponseService massService;  /\*\*  \*  \* <pre>  \* 메소드명 : list  \* 설 명 : 구성원 (재직) 목록 조회  \* </pre>  \*  \* 이력사항  \* 2019. 1. 11. ksh 최초작성  \*  \* **@param** request  \* **@param** response  \* **@param** data  \* **@return**  \* **@throws** Exception  \*/  @RequestMapping("/list.do")  **public** **void** list(HttpServletRequest request, HttpServletResponse response, RequestData data) **throws** Exception {  // request parameter setting  Map<String, String> param = **new** HashMap<String, String>();  param.put("strUserDivCd", data.getString("strUserDivCd"));  param.put("strPartyId", data.getString("strPartyId"));  param.put("strNm", data.getString("strNm"));    param.put("RESNO\_VIEW\_YN", data.getString(ProcessConstants.***SSN\_FULL\_VIEW***));    massService.selectDataToJSONStream(request, response, "dsParty", "cmn-user01.selectPartyList", param);  } |

<TSV 응답처리 예시>

|  |
| --- |
| @Controller  @RequestMapping("/CmnParty")  **public** **class** CmnPartyController {  @Autowired  **private** MassResponseService massService;  /\*\*  \*  \* <pre>  \* 메소드명 : list  \* 설 명 : 구성원 (재직) 목록 조회  \* </pre>  \*  \* 이력사항  \* 2019. 1. 11. ksh 최초작성  \*  \* **@param** request  \* **@param** response  \* **@param** data  \* **@return**  \* **@throws** Exception  \*/  @RequestMapping("/list.do")  **public** **void** list(HttpServletRequest request, HttpServletResponse response, RequestData data) **throws** Exception {  // request parameter setting  Map<String, String> param = **new** HashMap<String, String>();  param.put("strUserDivCd", data.getString("strUserDivCd"));  param.put("strPartyId", data.getString("strPartyId"));  param.put("strNm", data.getString("strNm"));    param.put("RESNO\_VIEW\_YN", data.getString(ProcessConstants.***SSN\_FULL\_VIEW***));    massService.selectDataToTSVStream(request, response, "cmn-user01.selectPartyList", param);  } |

1. 공통모듈 활용방법
   1. 첨부파일 업로드

파일 업로드 UDC(udcCmnPFileUpload.clx)를 화면 프리폼 또는 그리드 컬럼에 추가하여 사용한다.



<관련 속성>

|  |  |
| --- | --- |
| 속성명 | 내용 |
| buttonLabel | UDC 버튼명칭 |
| fileSizeUnit | 파일 업로드 용량 체크 단위(영문 소문자 사용)  (byte, kb, mb, gb) |
| limitFileExt | 업로드 가능 파일 확장자명  해당 속성은 HTML5 파일 업로드 확장자 제한 규칙을 따름  해당 속성이 지정되어 있지 않은 경우, 공통에서 제한하고 있는 업로드 불가 확장자(실행파일(.exe), 스크립트 파일 등등)을 제외한 모든 파일 유형 업로드 가능  예) .xls, .xlsx  .xxx : 특정 파일 확장자만 허용(ex- .txt, .doc, .xlsx 등)  audio/\*: 모든 오디오 파일  video/\*: 모든 비디오 파일  image/\*: 모든 이미지 |
| maxUploadSize | 최대 업로드 가능한 파일 사이즈를 제한하고자 하는 경우에 지정  디폴트: 50Mb |
| multi | 다중 파일 업로드 가능여부(true/false)  디폴트는 단건 파일 업로드만 가능 |
| attcFileNo | 첨부파일 번호(ATTC\_FILE\_NO) |
| uploadUrl | 첨부파일 업로드 요청 Action URL  - 공통 업로드 로직이 아닌 별도 업무단에서 업로드 로직을 구현하는 경우에만 지정. |
| onlyDownload | 다운로드만 가능한 첨부파일 목록 팝업을 호출하는 경우 |
| userDefineStorePgmId | 부모 프로그램ID 가 아닌 사용자 정의한 파일저장소에 대한 별도 프로그램ID  - ex) 메뉴ID  - 기본적으로 첨부파일은 프로그램ID별로 저장됨  - 프로그램ID는 메뉴정보에서 자동 취득됨 |

■ 업로드 경로 지정

업로드된 첨부파일은 프로그램별로 스토리지에 저장되며, 업로드 경로는 아래와 같다.

|  |
| --- |
| 실제 파일경로: 단위시스템{3}/프로그램ID/년월  예시) cmn/cmnTemplate/201901/ |

<업로드 경로 지정방법>

① appworks.properties 파일에 프로그램ID별 경로를 추가한다.

② 프로그램ID.fileStorePath 키값에 경로를 지정한다.

|  |
| --- |
| 예시)  aacCVendInfo.fileStorePath=/aac/vendInfo |

■ 첨부파일 완료 콜백이벤트

첨부파일 업로드 이후에 UI화면단에서 후처리를 하고자 하는 경우에는 콜백함수를 재정의할 수 있다.

이벤트명: uploadComplete

반환 값)

첨부파일 번호: event.attcFileNo

파일명: event.fileNm

파일 개수: event.fileCnt

■ 첨부파일 상태

공통 파일 업로드를 통한 첨부파일 업로드인 경우에, 파일 업로드 후에는 해당 파일의 상태는 임시저장 상태가 된다. 관련된 업무데이터가 업데이트/저장된 경우에 해당 파일의 상태를 ‘저장완료’ 상태로 변경한다.

- 임시저장(**TEMP**): 파일이 서버 스토리지에 저장된 상태이며, 아직 업무쪽 데이터에 첨부파일 정보가 들어가지 않은 상태임을 의미

- 저장완료(**SAVE**): 파일이 서버 스토리지에 저장되었으며, 업무쪽 첨부파일번호 컬럼에도 데이터가 들어가서 참조할 수 있는 상태임을 의미

■ 첨부파일 업로드 로직 처리

1) 데이터 조회시

- 첨부파일 정보가 있는 업무인 경우, 조회 쿼리문에 이전 첨부파일 순번을 추가적으로 SELECT한다. 이는 데이터 신규 또는 변경시에 첨부파일의 상태값을 업데이트하는데 사용하기 위함이다.

|  |
| --- |
| <select id=*"selectCmnTemplateFileMngList"* parameterType=*"HashMap"* resultType=*"HashMap"*>  /\* QueryId : cmn-base01.selectCmnTemplateFileMngList \*/  <![CDATA[  SELECT ATTC\_FILE\_NO  , F\_CMN\_FILE\_INFO(ATTC\_FILE\_NO, 'NM') ATTC\_FILE\_NM  , ATTC\_FILE\_NO AS ORI\_ATTC\_FILE\_NO  , FORM\_FILE\_NM  , UNIT\_SYS\_RCD  , REG\_DT  , CHARGER\_OPEN\_YN  , CRT\_USER\_ID  FROM CMN4503  ]]>  <where>  <if test=*"UNIT\_SYS\_RCD != null and UNIT\_SYS\_RCD != ''"* >  AND UNIT\_SYS\_RCD = #{UNIT\_SYS\_RCD}  </if>  <if test=*"FORM\_FILE\_NM != null and FORM\_FILE\_NM != ''"* >  AND FORM\_FILE\_NM LIKE '%' || #{FORM\_FILE\_NM} || '%'  </if>  </where>  ORDER BY UNIT\_SYS\_RCD,REG\_DT  </select> |

2) 데이터 신규시

- 업무 데이터 신규 저장된 경우, 해당 데이터에 관련된 첨부파일의 상태정보를 업데이트 한다.

3) 데이터 변경시

- 업무 데이터 신규 저장된 경우, 해당 데이터에 관련된 첨부파일의 상태정보를 업데이트 한다.

4) 데이터 삭제시

- 업무 데이터 삭제가 된 경우, 해당 데이터에 관련된 첨부파일도 같이 삭제가 되어야 한다.

|  |
| --- |
| @Override  **public** **void** savecmnTemplateFile(RequestData requestData) **throws** Exception{  ParameterGroup dsHoliday = requestData.getParameterGroup("dsTemplateFile");  Iterator<ParameterRow> insertedRows = dsHoliday.getInsertedRows();  Iterator<ParameterRow> updatedRows = dsHoliday.getUpdatedRows();  Iterator<ParameterRow> deletedRows = dsHoliday.getDeletedRows();    Map<String, String> mapFile = **new** HashMap<String, String>();  Map<String, String> param = **null**;  //업무 데이터 삭제시, 첨부파일도 삭제한다.  **while** (deletedRows.hasNext()) {  param = deletedRows.next().toMap();  dao.delete("cmn-tables01.deleteCmn4503", param);    //첨부파일 삭제  mapFile.clear();  mapFile.put("ATTC\_FILE\_NO", param.get("ATTC\_FILE\_NO"));    cmnFileService.**deleteCmnFileByAttcFileNo**(mapFile);  }    //업무 데이터 신규시, 첨부파일 정보 업데이트  **while** (insertedRows.hasNext()) {  param = insertedRows.next().toMap();    dao.insert("cmn-tables01.insertCmn4503", param);    //첨부파일 상태정보 업데이트(TEMP 🡪 SAVE)  cmnFileService.**commitCmnFile**(param.get("ATTC\_FILE\_NO"), param.get("ORI\_ATTC\_FILE\_NO"));  }    //업무 데이터 변경시, 첨부파일 정보 업데이트  **while** (updatedRows.hasNext()) {  param = insertedRows.next().toMap();  dao.update("cmn-tables01.updateCmn4503", updatedRows.next().toMap());    //첨부파일 상태정보 업데이트(TEMP 🡪 SAVE)  cmnFileService.**commitCmnFile**(param.get("ATTC\_FILE\_NO"), param.get("ORI\_ATTC\_FILE\_NO"));  }  } |

* 1. 첨부파일 다운로드

파일 다운로드 UDC(udcCmnPFileDownload.clx)를 화면 프리폼 또는 그리드 컬럼에 추가하여 사용한다.

파일 다운로드에서 제공하는 기능은 아래와 같다.

* 공통 첨부파일 다운로드
* 양식파일 다운로드
* 기타 직접파일 다운로드

<관련 속성>

|  |  |  |
| --- | --- | --- |
| 구분 | 속성명 | 내용 |
| 공통 속성 | buttonLabel | UDC 버튼명칭 |
| 공통 첨부 | attcFileNo | [첨부파일 번호]  공통 파일업로드를 사용한 경우의 첨부파일 번호를 지정한다. |
| 공통 첨부 | attcFileSeq | [첨부파일 순번]  공통 파일업로드를 사용한 경우의 첨부파일 순번을 지정한다.  (옵션) – 지정되지 않은 경우는 디폴트 1임 |
| 양식 다운로드 | templateFileNm | 공통 양식관리에 등록된 양식 파일을 다운로드하고자 하는 경우에 양식 파일명을 지정함 |
| 그 외 | fileNm | 공통 파일업로드를 사용하지 않은, 직접 업로드한 파일을 다운로드 받고자 하는 경우에 사용.  [다운로드 받을 파일명]  해당 속성이 지정되어 있지 않은 경우에는 filePath에서 파일명을 추출한 파일명이 됨  예) 샘플.txt |
| 그 외 | filePath | 통 파일업로드를 사용하지 않은, 직접 업로드한 파일을 다운로드 받고자 하는 경우에 사용.  다운로드 받을 서버상의 파일 경우(파일명 포함)  예) /cmn/temp/테스트파일.txt |
| 기타 속성 | responseType | 응답 reponse 유형을 지정  대용량 파일 다운로드시에는 filedown으로 지정하는게 적합함 |

* 1. 엑셀 업로드팝업

일반적인 엑셀 파일 업로드는 공통 팝업을 사용하여 처리한다. 공통 팝업을 사용하지 않는 경우에는 각 업무 화면에서 직접 Apache POI 라이브러리를 사용해서 직접 구현해야 한다.

공통 엑셀 업로드 기능을 사용하기 위해서는 각 업무화면에서 엑셀업로드용 UDC(comButtonExcel.clx) 를 추가하여 사용한다.

[엑셀 업로드 팝업 예시 및 기능]



* [파일열기]: 엑셀 파일을 선택하는 다이얼로그를 오픈하며, 선택된 파일을 서버에 업로드 한다.
* [삭제]: 선택된 그리드의 행을 삭제한다.
* [자료내려받기]: 해당 그리에 있는 데이터를 부모창 그리드로 내려준다.

■ 기능 설명

- 그리드의 헤더 컬럼은 부모창의 대상 그리드의 정보를 읽어서 자동으로 셋팅됨

- 그리드의 헤더 컬럼을 콤보박스로 사용자가 변경할 수 있음(즉, 첫번째 컬럼의 데이터는 C 컬럼으로 지정 가능)

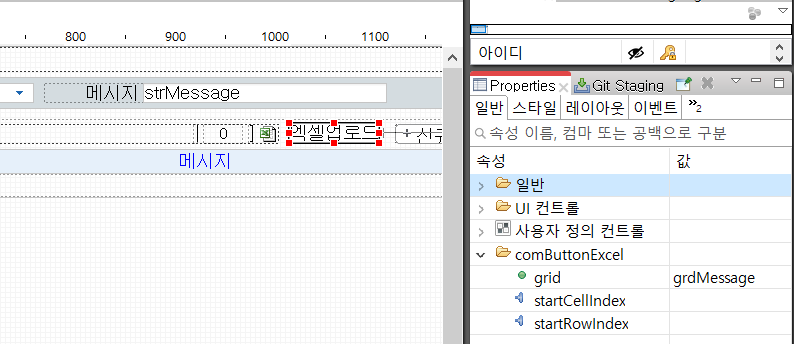
- 헤더 콤보박스에서 타 컬럼에 이미 매핑된 항목은 선택 불가

- [자료내려받기]시 헤더 컬럼에 지정된 부모창 그리드의 컬럼에 데이터가 매핑되어 신규 ROW로 추가됨

■ 적용방법

① 화면에 UDC를 추가한다.

② 엑셀 업로드 대상 정보를 설정한다.



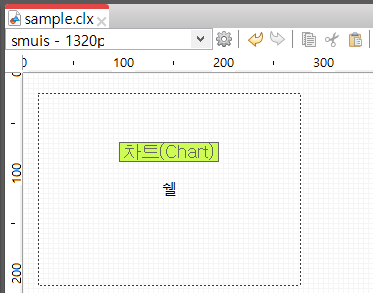
* grid: 대상 그리드 객체
* startRowIndex: 엑셀파일에서 데이터를 읽을 ROW의 시작인덱스(지정하지 않은 경우… 부모창의 그리드의 헤더 정보를 가지고 자동으로 셋팅됨)
* startCellIndex: 엑셀파일에서 데이터를 읽을 CELL의 시작인덱스(디폴트: 0부터임)
* ignoreDuplicate: 중복된 ROW 데이터 제외하고 내려받을지 여부(DataSet에 설정된 PK컬럼 기준)
  1. 차트(Chart) 사용

데이터를 시각적으로 표시하기 위해서 차트(Chart) 라이브러리를 사용한다.

차트(Chart) 라이브러리는 D3 기반인 eChart를 사용한다. ( https://ecomfe.github.io/echarts-doc/public/en/index.html)

eXbuilder6로 개발된 UI화면에서 차트(Chart)를 보여주기 위해서는 공통 임베디드앱 화면을 사용한다.

① 차트 앱을 해당 화면에 임베디드앱으로 cmnEChart.clx를 화면에 추가한다.



② 해당 차트(Chart) UDC의 drawChart() 함수를 호출하여 원하는 데이터를 차트로 표시한다.

|  |
| --- |
| var option = {  xAxis: {  type: 'category',  data: ['월', '화', '수', '목', '금', '토', '일']  },  yAxis: {  type: 'value'  },  series: [{  data: [820, 932, 901, 934, 1290, 1330, 1320],  type: 'line'  }]  };  app.lookup("embChart").callAppMethod("drawChart", ["차트샘플", option]);  //심플 차트를 그린다.  app.lookup("embChart").callAppMethod("drawSimpleChart", "차트샘플", 'line', ['월', '화', '수', '목', '금', '토', '일'], [820, 932, 901, 934, 1290, 1330, 1320]); |

* xAxis : X측 카테고리/값
* yAxis : Y측 카테고리/값
* series : 차트 시리즈(여러 개의 차트 유형을 포함할 수 있음)

차트(Chart)는 Line, Bar, Pie 등 다양한 형태로 데이터를 표현할 수 있으며, 차트에 대한 샘플은 아래의 사이트를 참고한다.

데모 사이트: <https://ecomfe.github.io/echarts-examples/public/index.html>

1. 변경 내용: 변경이 발생되는 위치와 변경 내용을 자세히 기록(장/절과 변경 내용을 기술한다.) [↑](#footnote-ref-1)