Anyframe XP Query Plugin

Version 1.6.0

I.Introduction

Installation

Command 창에서 다음과 같이 명령어를 입력하여 Anyframe XP Query Plugin을 설치한다.

```
mvn anyframe:install -Dname=xp-query
```

installed(mvn anyframe:installed) 혹은 jetty:run(mvn clean jetty:run) command를 이용하여 설치 결과를 확인해볼 수 있다.

플러그인 설치 후 브라우저를 통해서 Anyframe XP Query Plugin을 확인 하기 위해서는 아래의 추가 작업이 필요하다.

- **TOBESOFT가 제공하는 UX Studio(UI 개발툴) 혹은 XPLATFORM RUNTIME client plugin(ver 9.2)을 설치한다.**
- Anyframe XP Query Plugin을 동작시키기 위해서는 TOBESOFT가 제공하는 XPLATFORM_SERVER_License.xml, XPLATFORM_Client_License.xml 총 2가지 license가 필요하다. 현재 Anyframe XP Query Plugin 내에는 두 가지 라이센스가 모두 포함 되어 있으며, license file의 상세 위치는 다음과 같다.
  - XPLATFORM_SERVER_LICENSE.xml : xplatform-api-x.x.x.jar 파일 내에 존재
  - XPLATFORM_CLIENT_LICENSE.xml : {PROJECT_HOME}\src\main\webapp\xp-query\basic\ 폴더에 존재

Anyframe XP Query Plugin 개발 환경

Anyframe XP Query Plugin에서 제공하는 예제 화면은 XPLATFORM Runtime 9.2 버전을 기준으로 개발 되었다. XPLATFORM Runtime client나 UX Studio가 상이한 버전으로 설치 된 경우, 비정상 동작 하거나 기동 되지 않는 문제가 생길 수 있으니 참고한다.

Dependent Plugins

Plugin Name	Version Range
Query Ria [http://dev.anyframejava.org/docs/anyframe/plugin/optional/query-ria/1.6.0/reference/htmlsingle/query-ria.html]	2.0.0 > * > 1.4.0

MySQL 사용 시 유의사항

본 샘플 어플리케이션은 ID 채번을 위해 Database의 Sequence/Function 기능을 이용 하고있다. 샘플 어플리케이션을 MySQL DB를 사용하는 환경에서 설치할 때, Function을 사용자가 MySQL Client 프로그램을 이용하여 직접 등록해야 한다. Function 생성 구문은 [프로젝트 폴더]/db/scripts/xp-query-insert-data-mysql.sql 파일에 작성 되어있다.

II.XPLATFORM Integration

Anyframe에서는 RIA 개발 플랫폼인 XPLATFORM과 쉽고 편하게 연계할 수 있도록 XPQueryService와 HttpXPMessageConverter를 제공하고 있다.

1.HttpXPMessageConverter

XPLATFORM을 사용하는 환경에서 Client UI는 서버로 XML형태의 데이터를 전송한다. 이 데이터를 이용하여 Business Service를 실행하기 위해서는 변환 작업을 거쳐야 한다. Anyframe에서는 복잡한 변환 로직을 간편하게 처리하기 위해 HttpXPMessageConverter를 제공한다.

```
public class HttpXPMessageConverter implements HttpMessageConverter<Object> {
 // ...중 략
 public Object read(Class<? extends Object> clazz,
   HttpInputMessage inputMessage) throws HttpMessageNotReadableException {
  try {
   HttpServletRequest request = ((ServletServerHttpRequest) inputMessage)
     .getServletRequest();
   return new XPRequestHandler(request, contentType, encoding);
  } catch (Exception e) {
   logger.error(e.getMessage());
   throw new HttpMessageNotReadableException("Could not transform [" + clazz + "]", e);
  }
 }

 public void write(Object object, MediaType mediaType,
   HttpOutputMessage outputMessage) throws HttpMessageNotWritableException {
  if (outputMessage instanceof ServletServerHttpResponse) {
   try {
    ((XPResponseHandler) object).sendData(
      ((ServletServerHttpResponse) outputMessage)
        .getServletResponse(), contentType, encoding);
   } catch (Exception e) {
    logger.error(e.getMessage());
    new HttpMessageNotWritableException("Could not write [" + object + "]", e);
   }
  }
 }
```

HttpXPMessageConverter를 이용하기 위해서는 @RequestBody, @ResponseBody Annotation을 이용해야 한다. 이에 대한 자세한 내용은 Anyframe Core Plugin 매뉴얼의 @RequestBody, @ResponseBody 부분 [http://dev.anyframejava.org/docs/anyframe/plugin/essential/core/1.0.4/reference/htmlsingle/core.html#core_springmvc_controller_implementation_requestbody]을 참고한다. (http://dev.anyframejava.org/docs/anyframe/plugin/essential/core/1.0.4/reference/htmlsingle/core.html#core_springmvc_controller_implementation_requestbody)

HttpXPMessageConverter를 이용한 Controller 코드를 살펴보면 다음과 같다.

```
@Controller
@RequestMapping("/xpQueryMovie.do")
public class MovieController {

 @Inject
 @Named("xpQueryMovieService")
 private MovieService movieService;

 @RequestMapping(params = "method=getList")
 @ResponseBody
 public XPResponseHandler getList(
   @RequestBody XPRequestHandler requestHandler) throws Exception {
  VariableList inputVariableList = requestHandler.getInputVariableList();
  DataSetList inputDataSetList = requestHandler.getInputDataSetList();
```

```
 VariableList outputVariableList = new VariableList();
 DataSetList outputDataSetList = new DataSetList();

 try {
  movieService.getList(inputVariableList, inputDataSetList,
    outputVariableList, outputDataSetList);

  return new XPResponseHandler(outputDataSetList, outputVariableList);
 } catch (Exception e) {
  return setFailMessage(outputDataSetList, outputVariableList, e);
 }

}

// ...중략
```

위 코드를 살펴보면 일반적인 Spring MVC 아키텍쳐를 활용하는 Annotation 기반의 Controller 클래스임을 확인할 수 있다. @RequstBody Annotation을 이용하여 Request를 XPRequestHandler 객체로 변환하고, 최종적으로 XPResponseHandler 객체를 리턴하면 @ResponseBody Annotation을 이용하여 XPLATFORM Data를 이용한 Response를 구성하도록 되어있다.

위 코드에서 사용된 XPRequestHandler, XPResponseHandler 객체는 Request/Response로부터 XPLATFORM Data를 핸들링 하기 위해 제공되는 객체이다. 상세한 설명은 아래의 표를 참고 하도록 한다.

- XPRequestHandler

Method Name	Description	
getInputDataSetList()	Client에서 전송한 DataSetList를 리턴	
getInputVariableList()	Client에서 전송한 VariableList를 리턴	

- XPResponseHandler

Method Name	Description	
addVariableList(VariableList variableList)	Client로 전송하는 VariableList를 추가	
addDataSet(DataSet dataSet)	Client로 전송하는 DataSetList에 DataSet을 추가	
setResultMessage(int errorCode, String message)	Client Callback에서 사용할 에러 메세지와 에러 코드를 설정	
addVariableList(String key, Object value)	Client로 전송하는 VariableList에 주어진 key/value 쌍을 추가	
addVariableList(String key, Object value)	Client로 전송하는 VariableList에 주어진 key/value 쌍을 추가	

XPLATFORM에서는 서버에서 대용량 데이터를 클라이언트로 전송할 때 전체 데이터를 분할해서 전송하는 Firstrow 방식을 제공한다. HttpXPMessageConverter에서는 DataSet에 담긴 대량의 데이터를 사용자가 입력한 값(nextDataSize) 만큼의 Row로 잘라서 CSV 포맷으로 전송하도록 구현 되어있다. 대용량 데이터가 VariableList의 형태로 저장된 경우에는 Firstrow 방식을 사용할 수 없음을 유의해야 한다. 아래는 Firstrow 처리를 위한 XPResponseHandler 생성자의 파라미터에 대한 설명이다.

변수명	설명	
boolean isFirstrow	Firstrow 방식으로 전송할지 여부	
boolean isCompression	Firstrow 방식으로 전송할 때, 압축 할지 여부	

변수명	설명	
int nextDataSize	Firstrow 방식으로 전송할 때, DataSet Row 분할 기준이 되는 Data Size 값	

2.XPQueryService

프리젠테이션 레이어 개발 시 Ria 제품인 XPLATFORM 또는 MiPlatform 등을 기반으로 할 경우 각 제품은 사용자 입력 사항을 제품 고유의 데이터 형태에 저장하여 전달한다. 따라서 Query 서비스를 이용하여 DB 데이터를 처리하기 위해서는 "제품 고유의 데이터 전달 형태 <-> Map 또는 VO 간의 변환"을 위한 추가 작업이 필요하며, 이로 인해 대량의 데이터를 다루는 경우 성능 저하가 발생할 가능성이 크다. Query 서비스에서는 기본 QueryService를 확장하여, 특정 Ria제품에 최적화된 형태의 구현체를 제공함으로써 개발 편의성과 응답 속도 향상을 도모하고자 한다. 다음에서는 XPLATFORM에 최적화된 XPQueryService 사용 방법에 대해서 살펴보도록 한다.

2.1.XPQueryService 활용

XPQueryService는 XPLATFORM 고유의 데이터 전달 형태로부터 사용자가 입력한 데이터를 추출하여 해당 DB에 반영하는 역할을 수행한다. XPLATFORM 기반으로 프리젠테이션 레이어를 개발하는 경우, XPQueryService를 통해 XPLATFORM 고유의 데이터 전달 형태인 com.tobesoft.xplatform.data.DataSet, com.tobesoft.xplatform.data.VariableList를 Map, VO 형태로의 변환 없이 그대로 이용할 수 있게 된다.

Pagination시 유의 사항

PagingJdbcTemplate 속성 정의시에는 반드시 DBMS에 적합한 PagingSQLGenerator를 셋팅해 주어야 한다. 적절한 PagingSQLGenerator가 존재하지 않는 경우에는 QueryService에서 제공하는 org.anyframe.query.impl.jdbc.generator.DefaultPagingSQLGenerator를 사용할 수 있으나, DefaultPagingSQLGenerator를 이용하여 findXXX() 메소드를 실행하면 QueryService 내부적으로 조건에 해당하는 모든 데이터를 fetch한 이후 ResultSet Cursor의 위치를 이동시키는 방식으로 특정 페이지에 속한 데이터를 걸러낸다. 이 때 ResultSet Cursor를 움직이는 로직에서 상당한 시간이 소요되어 다량의 데이터 조회시 성능에 심각한 영향을 끼칠 수 있다. 따라서, DefaultPagingSQLGenerator 사용은 권장하지 않는다.

2.1.1.XPQueryService 속성 정의 파일

다음은 XPQueryService를 정의한 context-xp-query.xml 파일의 일부이다. XPQueryService는 내부적으로 RiaQueryService를 통해 데이터 접근 처리를 수행하므로 RiaQueryService에 대한 참조 관계 설정이 필요하다.

```
<bean id="xpQueryService" class="org.anyframe.xp.query.impl.XPQueryServiceImpl">
 <property name="namedParamJdbcTemplate" ref="xpPagingNamedParamJdbcTemplate"/>
 <!--xp-query-lobHandler-START-->
 <property name="lobHandler" ref="lobHandler"/>
 <!--xp-query-lobHandler-END-->
 <property name="sqlRepository" ref="xpSqlLoader"/>
 <lookup-method name="getRowCallbackHandler" bean="xpRowCallbackHandler"/>
 <lookup-method name="getCallableStatementCallbackHandler"
 bean="xpCallableStatementCallbackHandler"/>
 <lookup-method name="getPrintWriterRowCallbackHandler"
 bean="xpPrintWriterRowCallbackHandler"/>
</bean>

<bean id="xpCallableStatementCallbackHandler"
 class="org.anyframe.xp.query.impl.jdbc.mapper.XPCallableStatementCallbackHandler"
 scope="prototype"/>

<bean id="xpRowCallbackHandler"
 class="org.anyframe.xp.query.impl.jdbc.mapper.XPDataSetCallbackHandler" scope="prototype"/>
```

```
<bean id="xpPrintWriterRowCallbackHandler"
 class="org.anyframe.xp.query.impl.jdbc.mapper.XPPrintWriterCallbackHandler"
 scope="prototype"/>

   <bean id="xpJdbcTemplate" class="org.anyframe.query.impl.jdbc.PagingJdbcTemplate">
    <property name="dataSource" ref="dataSource"/>
    <property name="exceptionTranslator" ref="exceptionTranslator"/>
    <!--xp-query-pagingSQLGenerator-START-->
       <property name="paginationSQLGetter" ref="hsqldbPagingSQLGenerator"/>
    <!--xp-query-pagingSQLGenerator-END-->
   </bean>

   <bean id="xpPagingNamedParamJdbcTemplate"
 class="org.anyframe.query.impl.jdbc.PagingNamedParamJdbcTemplate">
    <constructor-arg ref="xpJdbcTemplate"/>
   </bean>

<bean name="xpSqlLoader" class="org.anyframe.query.impl.config.loader.MappingXMLLoader">
 <property name="mappingFiles">
  <value>classpath:/sql/xp-query/mapping-*.xml</value>
 </property>
 <property name="nullchecks">
  <map>
   <entry key="VARCHAR" value="" />
  </map>
 </property>
 <property name="skipError" value="true" />
</bean>
```

2.1.2.Mapping XML

다음은 앞서 정의한 XPQueryService에서 사용할 쿼리문을 정의한 mapping-xp-query-movie.xml 파일의 일부로, Named Parameter를 이용한 쿼리문들을 포함하고 있다.

```
<query id="createXPMovie" isDynamic="true" mappingStyle="upper">
 <statement>
  INSERT INTO
  MOVIE (MOVIE_ID, TITLE, DIRECTOR, GENRE_ID, ACTORS, RUNTIME, RELEASE_DATE, TICKET_PRICE,
 NOW_PLAYING, POSTER_FILE)
  VALUES
 (:MOVIE_ID, :TITLE, :DIRECTOR, :GENRE_ID, :ACTORS, :RUNTIME, :RELEASE_DATE, :TICKET_PRICE, :NOW_PLAYING, :POST
 </statement>
</query>

...중 략...

<query id="findXPMovieList" isDynamic="true" mappingStyle="upper">
 <statement>
  <!--xp-query-findXPMovieList-START-->
  SELECT
   MOVIE_ID, TITLE, DIRECTOR, GENRE_ID, ACTORS, RUNTIME, RELEASE_DATE, TICKET_PRICE,
 NOW_PLAYING, POSTER_FILE
  FROM MOVIE
  WHERE
   TITLE like '%' || :SEARCH_TITLE || '%'
   AND NOW_PLAYING = :SEARCH_NOW_PLAYING
  ORDER BY RELEASE_DATE DESC
  <!--xp-query-findXPMovieList-END-->
 </statement>
```

```
</query>
```

2.1.3.Test Code

다음에서는 XPQueryService를 이용하여 앞서 언급한 매핑 XML 파일에 정의된 INSERT, SELECT, UPDATE, DELETE 쿼리문을 실행하는 테스트 코드의 일부이다.

```
/**
 * [Flow #-1] Positive Case : 미리 세팅되어 있는 Database의 값을 XPQueryService의 update
 * method를 호출해 값을 수정하는 TestCase다. 수정될 값이 세팅되어 있는 DataSet를
 XPQueryService
 * update()의 argument로 직접 사용해 Database의 값이 제대로 수정 됐는지 검증한다.
 *
 * @throws Exception
 */
@Test
public void testUpdateDataSet() {
 insertDataSet();

 Map<String, String> queryMap = new HashMap<String, String>();
 queryMap.put(XPQueryService.QUERY_UPDATE, "updateXPQueryService");
 int resultUpdate = xpQueryService.update(queryMap, makeUpdateDataSet());
 Assert.assertEquals(1, resultUpdate);

 DataSet resultDataSet = findDataSet("bbnydory00");

 assertDataSet(resultDataSet, "bbnydory00", "2012-12-01", 12345678,
   1234.5678, "Anyframe XPQueryService Test. - UPDATE");
}

...중략...

/**
 * [Flow #-3] Positive Case : query가 실행 되기 전 비즈니스 로직의 있을 경우 ActionComman클래
스의
 * 해당메소드를 구현해서 Query실행 전, 후 ActionCommand의 메소드가 정상적으로 호출 되는지
 Query 실행이 정상적으로
 * 동작하는지 검증한다.
 *
 * @throws Exception
 */
@Test
public void testProcessAllDataSetWithActionCommand() {
 insertDataSet();

 Map<String, String> queryMap = new HashMap<String, String>();
 queryMap.put(XPQueryService.QUERY_INSERT, "createXPQueryService");
 queryMap.put(XPQueryService.QUERY_UPDATE, "updateXPQueryService");
 queryMap.put(XPQueryService.QUERY_DELETE, "deleteXPQueryService");
 int resultUpdate = xpQueryService.update(queryMap, makeAllDataSet(),
   new XPActionCommand() {

    public void postDelete(DataSet record, int currentRow) {
    }

    public void postInsert(DataSet record, int currentRow) {
    }

    public void postUpdate(DataSet record, int currentRow) {
    }
```

```
    public void preDelete(DataSet record, int currentRow) {
    }

    public void preInsert(DataSet record, int currentRow) {
     record.set(currentRow, "TEST_VARCHAR2",
       "Anyframe preUpdate");
    }

    public void preUpdate(DataSet record, int currentRow) {
    }
   });
 Assert.assertEquals("Fail to process all with ActionCommand.", 3,
   resultUpdate);

 DataSet resultDataSet = findDataSet("bbnydory88");

 assertDataSet(resultDataSet, "bbnydory88", "2012-12-01", 12345678,
   1234.5678, "Anyframe preUpdate");
}
```

위 소스 코드 중 testProcessAllDataSetWithActionCommand() 메소드에서는 ActionCommand를 이용하여 DB에 데이터를 입력하기 전에 특정 칼럼의 값을 변경하고 있다. 이와 같이 XPQueryService는 org.anyframe.xp.query.XPActionCommand 를 구현한 별도 ActionCommand를 인자로 함께 전달하는 경우 입력 데이터를 DB에 반영하기 전/후에 대한 공통 처리를 수행할 수 있도록 지원한다. 예를 들어 입력받은 개별 Row를 DB에 신규 등록하기 전에 신규 식별자 값이 셋팅되어야 한다면, Loop을 돌면서 각 Row를 추출한 뒤 식별자를 셋팅하는 별도 로직없이 preInsert() 로직 내에 식별자 생성 구문이 추가된 ActionCommand 객체만 전달하면 되는 것이다.

III.Simplification

3.Anyframe XP Query

어플리케이션의 UI를 XPLATFORM을 사용해 개발 할 경우, XPLATFORM 고유의 데이터 형태를 DB에 반영하기에는 많은 어려움이 있다.

예를 들어 DataSet에 10개의 Column과 10개의 Insert 할 Record가 있는 경우, 개발자가 일반적인 JDBC 코딩을 하기 위해서는 DataSet의 10개의 Column을 일일이 꺼내야 하고, 10번의 반복문을 실행 시키면서 Insert 동작을 수행하는 로직을 작성 해야 한다.

또 DB에서 값을 조회 하고자 할 경우에는 ResultSet의 메타 정보를 이용해 DataSet의 Column을 설정하고, 반복문을 수행 하면서 ResultSet에 저장된 값들을 DataSet에 추가하는 로직을 작성 해야 한다.

Anyframe XP Query는 XPLATFORM의 고유한 데이터 형태를 사용해서 DB에 조회/입력/수정/삭제 동작을 수행하기 위한 공통 비즈니스 서비스와 컨트롤러 클래스를 제공한다.

Anyframe XP Query의 장점은 아래와 같다.

- DataSet, VariableList와 같은 XPLATFORM 고유의 데이터 형태를 변환하지 않고 비즈니스 서비스 개발을 할 수 있다.
- 추가적인 비즈니스 로직이 필요 없는 CRUD에 대해서는 비즈니스 서비스 개발 없이 Query Mapping File에 필요한 Query만 작성하면 된다.
- 확장이 필요한 부분만 오버라이드 해서 사용할 수 있기 때문에 비즈니스 서비스 개발이 쉽다.
- 기능이 중복되거나 불필요한 클래스를 생성하지 않기 때문에 전체 클래스 수가 줄어 들고 유지보수 또한 용이하다.

Anyframe XP Query는 크게 Controller, Service, Dao로 구성 되어 있다.

- **Controller** – XPController : 사용자 요청에 따라 비즈니스 서비스의 메소드를 호출하고 결과값을 화면으로 전달하는 공통 컨트롤러 역할을 담당한다.
- **Service 인터페이스** – XPService : DataSetList, VariableList를 이용해 DB에서 데이터를 조회, 추가, 삭제, 수정 등을 할 수 있는 API를 제공한다.
- **Service 구현 클래스** – XPServiceImpl : XPService의 구현 클래스로 DataSet과 실행 하고자 하는 Query Id를 짝지은 후 XPDao의 메소드를 호출하고 Query실행 결과를 DataSetList에 추가한다.
- **Dao 클래스** – XPDao : 파라미터의 형태에 따라 적절한 XPQueryService의 메소드를 호출해 쿼리를 실행한다.

3.1.Controller

일반적인 Spring MVC 형태의 Controller 클래스를 구현하려면 사용자의 요청 별로 Controller 클래스를 구현해야 하므로 개발해야 할 Controller 클래스 수가 많아지고 유지보수 또한 어려워지는 단점이 있다. Anyframe에서는 이러한 단점을 보완하기 위해서 XPLATFORM 기반의 UI를 통한 사용자 요청을 처리할 수 있도록 Spring MVC의 AbstractController를 구현한 공통 컨트롤러 클래스인 XPController를 제공한다.

3.1.1.XPController

JSP 기반의 UI일 경우, 사용자 요청에 따라 Controller가 호출되고, Controller에서는 비즈니스 서비스 호출 결과 값을 결과 페이지에 전달하는 로직이 필요하다. 그러나 XPLATFORM 기반의 UI에서는 화면과 서버간의 주고받는 데이터의 유형(DataSetList, VariableList)이 동일하고, 요청 화면과 결과 화면이 같으므

로 공통화 처리가 가능해진다. 따라서, 비즈니스 서비스 호출 외에 별도 로직이 없을 때는 XPController를 공통 Controller로 사용할 수 있다.

아래는 XPController의 handleRequestInternal()의 일부로, 화면에서 전달받은 비즈니스 서비스의 Bean Id 를 이용해 WebApplicationContext에서 비즈니스 서비스 객체를 얻어온다. 실행할 비즈니스 서비스의 Bean Id와 메소드 이름은 Client에서 선언한 dsService DataSet의 SERVICE Column 값(예: movieService.getPagingList)에 의해 결정된다.

```
public class XPController extends AbstractController {

 public ModelAndView handleRequestInternal(HttpServletRequest request, HttpServletResponse
 response)
   throws Exception {

  HttpInputMessage inputMessage = new ServletServerHttpRequest(request);
  HttpOutputMessage outputMessage = new ServletServerHttpResponse(response);

  XPRequestHandler requestHandler = (XPRequestHandler) messageConverter
    .read(XPRequestHandler.class, inputMessage);
  XPResponseHandler responseHandler = null;

  VariableList inputVariableList = requestHandler.getInputVariableList();
  DataSetList inputDataSetList = requestHandler.getInputDataSetList();

  // ... 중략

  String serviceName = inputVariableList.getString(SERVICE_NAME);
  Object bean = getApplicationContext().getBean(serviceName);
  Method method = getMethod(bean, inputVariableList.getString(METHOD_NAME));

  DataSetList outputDataSetList = new DataSetList();
  VariableList outputVariableList = new VariableList();
  try {
   method.invoke(bean,
     new Object[] { requestHandler.getInputVariableList(),
 requestHandler.getInputDataSetList(),
       outputVariableList, outputDataSetList });

   // ...중략
    // in case of using general PlatformResponse
    responseHandler = new XPResponseHandler(outputDataSetList, outputVariableList);
    responseHandler.setResultMessage(0, "Success");
  } catch (Exception e) {
   logger.error("Can not invoke a dispatch method name", e);

   String msg = e.getMessage();

   if (msg == null)
    msg = "Fail to process client request.";

   responseHandler = new XPResponseHandler(outputDataSetList, outputVariableList);
   responseHandler.setResultMessage(-1, msg);
  }
  messageConverter.write(responseHandler, MediaType.APPLICATION_XML, outputMessage);
  return null;
 }
..중략
```

만약 아래 그림의 설정처럼 SERVICE의 값이 없을 경우에는 비즈니스 서비스의 Bean ID는 xpService이고, 메소드 이름은 dsService의 SVC_ID값의 prefix로 결정된다.

Rows:

No	SVC_ID	QUERY_LIST	SERVICE
1	saveAllBoard	querySet1=createXPBoard,updateXPBoard,removeXPBoard	
2	getPagingListBoard	querySet1=findXPBoardList	
3	getListCommunity	querySet1=findXPCommunityList	

prefix로는 get, getList, getPagingList, create, update, remove, saveAll이 올 수 있다. prefix가 getList일 경우에는 XPService의 getList()가 실행된다.

3.2.Service

Anyframe XP Query의 Service는 Interface인 XPService와 구현 클래스인 XPServiceImpl로 구성 되어있다.

3.2.1.XPService

XPService는 XPLATFORM의 고유 데이터 형태인 VariableList와 DataSetList를 이용하여 외부에 제공할 수 있는 일반적인 기능을 정의하고 있는 인터페이스 클래스이다. 아래는 XPService 클래스의 소스코드로, 모든 메소드의 입력 파라미터는 VariableList inVl, DataSetList inDl, VariableList outVl, DataSetList outDl이며, Return Type은 void인 것을 확인 할 수 있다.

```
public interface XPService {
    ..중략

 void getList(VariableList inVl, DataSetList inDl, VariableList outVl, DataSetList outDl)
 throws Exception;

 void getPagingList(VariableList inVl, DataSetList inDl, VariableList outVl, DataSetList
 outDl) throws Exception;

 void create(VariableList inVl, DataSetList inDl, VariableList outVl, DataSetList outDl)
 throws Exception;
    ..중략
}
```

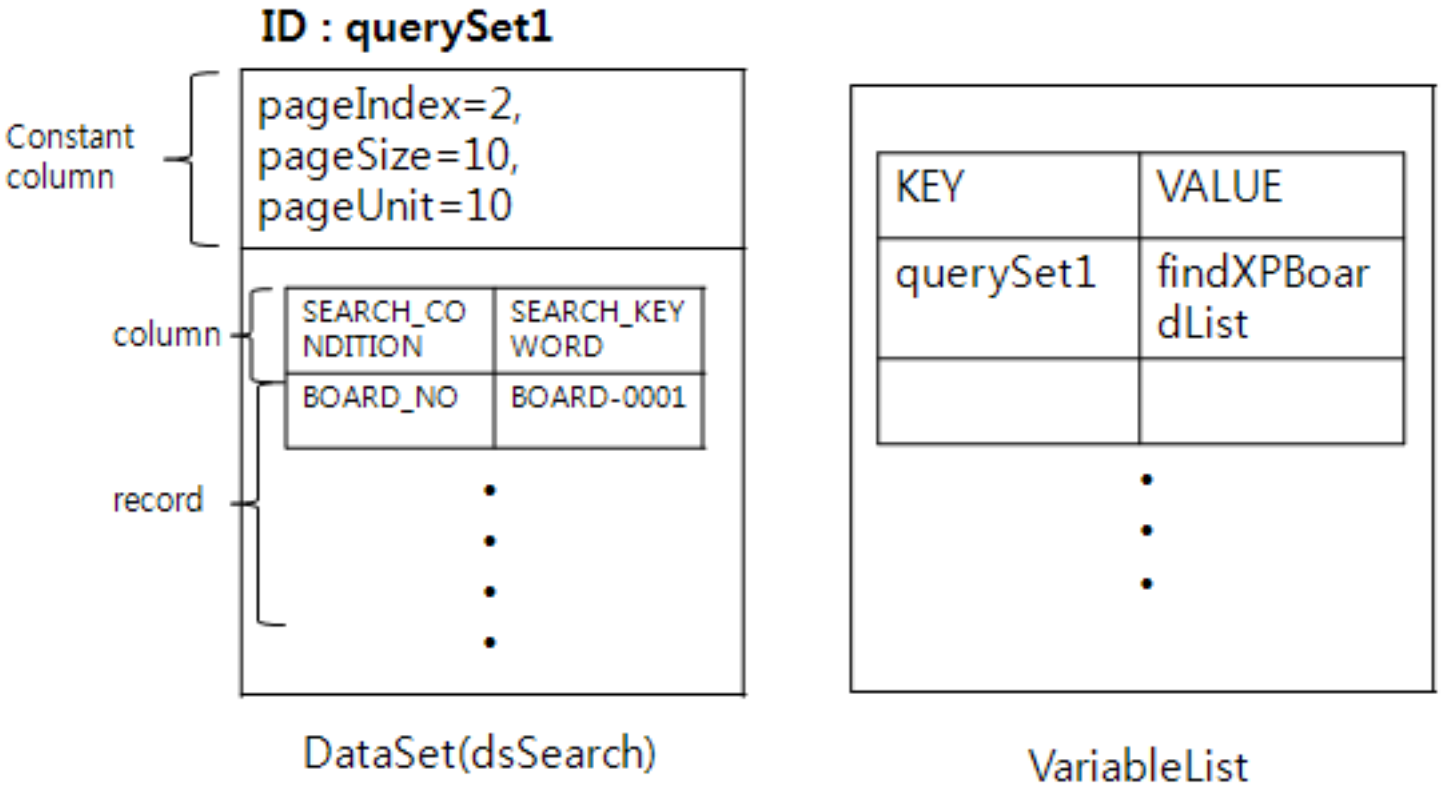
3.2.2.XPServiceImpl

XPServiceImpl은 XPService의 구현 클래스로 dsService에 설정된 정보를 기반으로 XPDao의 메소드를 호출하도록 구성 되어있다.

Rows:

No	SVC_ID	QUERY_LIST	SERVICE
1	saveAllBoard	querySet1=createXPBoard,updateXPBoard,removeXPBoard	
2	getPagingListBoard	querySet1=findXPBoardList	
3	getListCommunity	querySet1=findXPCommunityList	

위의 2번 Row와 같이 dsService를 설정 했다면 VariableList에 아래 그림과 같이 값이 서버로 전달된다.



아래는 XPServiceImpl의 getPagingList() 메소드의 코드중 일부분이다. 이 메소드는 특정 페이지에 속한 데이터를 조회 하는 기능을 제공한다.

getPagingList() 메소드에서는 위의 그림에서와 같이 입력 파라미터로 전달된 DataSetList에서 DataSet의 이름이 "querySet1"인 DataSet을 얻어와 inDs 변수에 담는다. 그 후 VariableList에서 querySet1이라는 KEY에 해당하는 값을 변수 queryId에 할당한다. 주어진 queryId와 inDs 변수를 이용하여 XPDao의 메소드를 호출한 후 결과값을 다시 화면으로 전달하기 위해서 outDs에 "querySet1" 이라는 이름으로 담는다.

```
public void getPagingList(VariableList inVl, DataSetList inDl, VariableList outVl,
 DataSetList outDl)
  throws Exception {

 int querySetCount = getQuerySetCount(inVl, outVl);

 String queryId = null;
 DataSet inDs = null;
 DataSet outDs = null;

 for (int i = 1; i <= querySetCount; i++) {
  queryId = inVl.getString("querySet" + i);
  inDs = inDl.get("querySet" + i);
  if (inDs != null) {
   outDs = xpDao.getPagingList(queryId, inDs);
  }
  outDs.setName("querySet" + i);
  outDl.add(outDs);
 }
}
```

XPServiceImpl의 다른 메소드들도 이와 같이 입력 파라미터로부터 추출한 DataSet과 Query ID를 이용하여 사용자의 요청을 처리한다.

아래의 그림은 XPService를 이용하기 위해 dsService DataSet을 설정한 모습이다.

Rows:

No	SVC_ID	QUERY_LIST	SERVICE
1	saveAllBoard	querySet1=createXPBoard,updateXPBoard,removeXPBoard	
2	getPagingListBoard	querySet1=findXPBoardList	
3	getListCommunity	querySet1=findXPCommunityList	

dsService의 값을 정의할 때 SVC_ID의 prefix에 따라 querySet에 정의할 수 있는 queryId의 갯수가 달라진다. prefix가 get, getList, getPagingList, create, update, remove인 경우에는 하나의 querySet에 하나의 queryId가 정의 되어야 한다. 하지만 prefix가 saveAll 인 경우에는 QUERY_LIST에 "querySet1=createXPBoard,updateXPBoard,removeXPBoard" 와 같이 추가/수정/삭제 작업을 수행 할 3개의 Query ID가 순서대로 정의 되어 있어야 한다. 조회 메소드의 경우(get, getList, getPagingList) 서버에서 처리 후, DataSet의 이름을 검색 조건으로 사용했던 DataSet ID(querySet + 번호)로 설정해서 화면으로 넘겨주도록 되어있다.

3.3.XPDao

XPDao는 XPQueryService를 이용해 Query를 실행 시키는 Class이다.

아래의 Java 코드는 서버로 전송된 DataSet Record를 DB Table에 저장(추가, 수정, 삭제) 하는 saveAll() 메소드 구현 부분이다.

```
public int saveAll(Map<String, String> queryMap, DataSet inDs, XPActionCommand
 actionCommand) throws QueryException {
 if (actionCommand == null) {
  return xpQueryService.update(queryMap, inDs);
 }
 else {
  return xpQueryService.update(queryMap, inDs, actionCommand);
 }
}
```

insert, update, delete를 수행할 Query ID를 담은 Map객체와, 각 Row Record를 담고있는 DataSet을 이용해 XPQueryService의 update()를 호출 하도록 구현 되어있다.

XPLATFORM과 Anyframe의 연계 구조에서 DAO를 구현할 경우 XPDao를 사용할 것을 추천하고, 꼭 필요한 경우에 한해 확장해서 사용하도록 한다.

3.4.Extension of XPServiceImpl

XPService에서 제공하는 기능 외에 추가적인 기능이 필요한 경우에는 API를 추가로 정의 하거나 해당 메소드를 오버라이드 할 수 있다.

아래의 코드는 DataSet의 Record를 DB에 Insert 하기 전 "MOVIE_ID" Column에 유일한 아이디 값을 셋팅하기 위해 saveAll() 메소드를 오버라이드 해 기능을 확장 구현한 예제이다.

```
@Service("xpQueryMovieService")
@Transactional(rollbackFor = {Exception.class}, propagation = Propagation.REQUIRED)
public class MovieServiceImpl extends XPServiceImpl implements MovieService{

 @Inject
 public MovieServiceImpl(XPDao xpDao){
  super.xpDao = xpDao;
 }
```

```
    ..중략
 public void saveAll(VariableList inVL, DataSetList inDl, VariableList outVl, DataSetList
 outDl) throws Exception{
  Map<String, String> sqlMap = new HashMap<String, String>();
  sqlMap.put(XPQueryServiceImpl.QUERY_INSERT, "createXPMovie");
  sqlMap.put(XPQueryServiceImpl.QUERY_UPDATE, "updateXPMovie");
  sqlMap.put(XPQueryServiceImpl.QUERY_DELETE, "removeXPMovie");

  xpDao.saveAll(sqlMap, inDl.get("dsSave"), new MovieActionCommand());
 }

}
```

위 예제에서는 XPDao의 saveAll() 메소드 호출 시 Anyframe에서 제공하는 XPActionCommand를 구현한 MovieActionCommand를 추가 파라미터로 전달하고 있다. XPActionCommand를 이용하면 특정 쿼리문의 수행 전/후에 필요한 비즈니스 로직을 수행 하도록 구현 할 수 있다.

3.4.1.[참고] XPActionCommand

XPActionCommand는 XPQueryService의 save() 메소드가 호출 됐을 때 Insert, Update, Delete Query를 실행 하기 전/후 필요한 비즈니스 로직을 추가할 수 있도록 제공되는 인터페이스이다. XPActionCommand 인터페이스를 구현한 별도의 클래스를 정의하고 해당 메소드에 비즈니스 로직을 추가하면 된다.

아래의 코드는 앞서 언급한 MovieActionCommand 클래스의 preInsert() 메소드 구현부분이다. DataSet을 특정 Table에 Insert 하기 전에 Primary Key에 해당하는 MOVIE_ID Column에 값을 셋팅 하도록 구현되어 있다.

```
public class MovieActionCommand implements XPActionCommand {

 public void preInsert(DataSet dataSet, int currentRow) {
  String id = "MV-" + System.currentTimeMillis();
  dataSet.set(currentRow, "MOVIE_ID", id);
 }
}
```

따라서, XPQueryService의 save() 메소드에서는 DataSet의 Status가 'INSERT"인 Record를 DB에 저장하기 전 MovieActionCommand의 preInsert를 호출해서 앞서 정의해둔 추가 로직을 실행 한다.

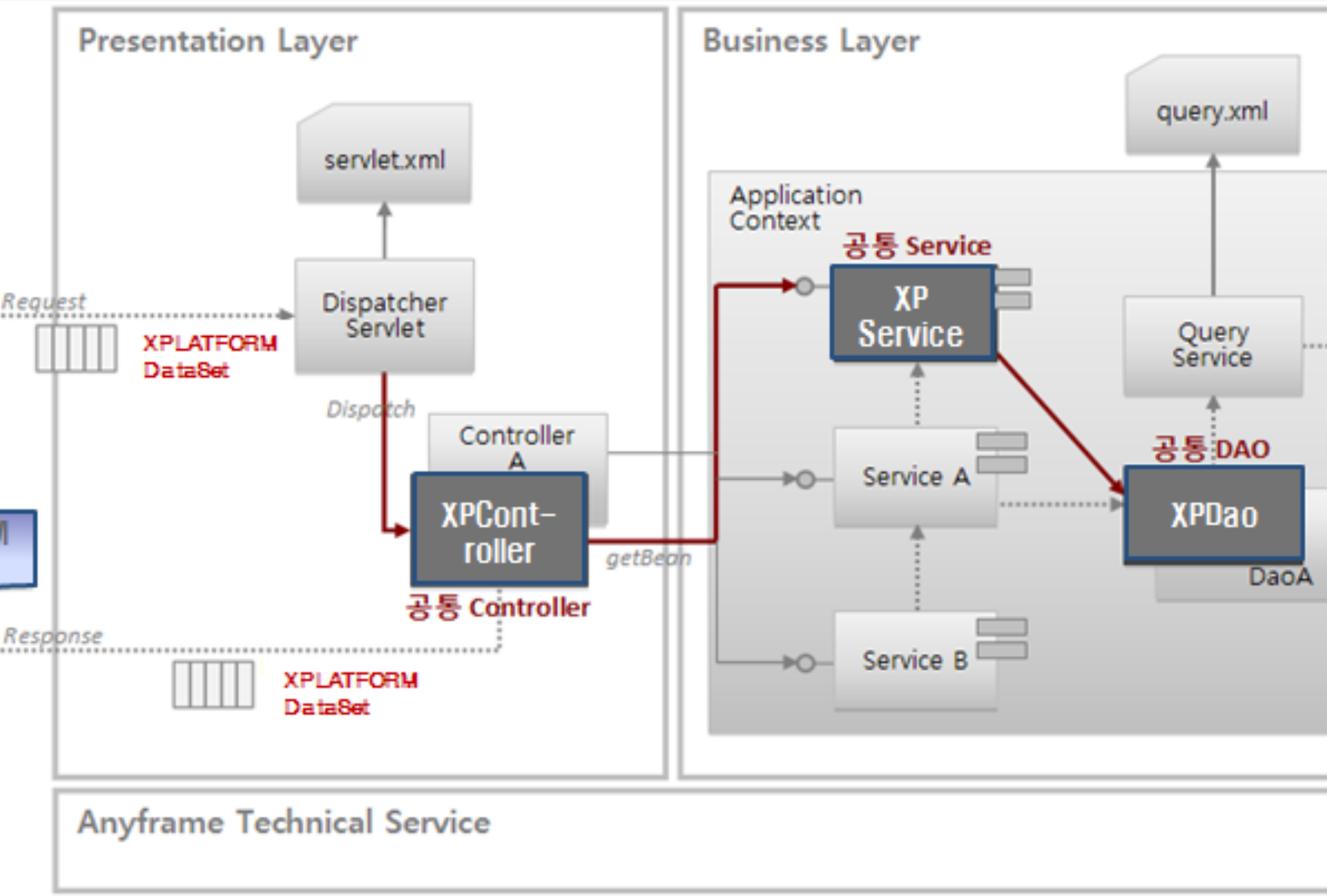
IV.XPLATFORM UI Sample

Java EE 어플리케이션 개발 프레임워크인 Anyframe Java와 Ria UI 개발 플랫폼인 XPLATFORM에 대한 연계 방안을 제시함으로써 Anyframe Java와 함께 XPLATFORM을 기반으로 어플리케이션을 개발하는 프로젝트에 효과적인 개발 방법을 제안 하고자 한다. 프로젝트 초기에 개발 템플릿을 정의하는데 소요되는 시간을 줄이고자 참조할 수 있는 XPLATFORM 화면 샘플 및 Eclipse 프로젝트 샘플을 제공하고, 설치에서부터 구현 기능 및 적용 방법에 대한 설명을 하고자 한다.

Anyframe XP Query Plugin은 XPLATFORM을 사용하여 UI 개발을 진행할 때, Anyframe 기반으로 Java EE 어플리케이션을 개발하는 방법을 보여주고, 기존에 Anyframe과 XPLATFORM을 사용하여 진행 되었던 몇몇 프로젝트에서 추출한 기본적인 화면 예제를 코드와 함께 제공하고 있다.

4.Architecture

본 문서의 이해를 돕기 위하여 Anyframe XP Query Plugin 전체 소프트웨어 아키텍처를 그림으로 나타내면 다음과 같다.



위의 아키텍처는 공통 Controller와 공통 Service, 공통 Dao에 의해서 최적화되며, 일반적인 CRUD 형태의 작업은 이들을 이용하는 공통 Flow(Common Flow)를 통해서 수행 된다. 만약 단순 CRUD가 아닌 복잡한 비즈니스 로직을 구현하는 경우에 대해서는 공통 Service를 대체하는 별도의 Service(예: Service A, Service B)를 구현하면 되며, 파일 업로드/다운로드, 로그인 등과 같이 공통 Controller의 범위를 벗어나는 Controller의 경우에는 Service와 마찬가지로 별도의 Controller(예: Controller A)를 구현하여 수행한다.

위의 그림에서 볼 수 있듯이 Model 2 MVC 구조에 기반한 프레젠테이션, 비즈니스 레이어는 각 레이어별로 개발 생산성을 향상 시키기 위한 공통 클래스를 사용하기 때문에, 경우에 따라서 다음과 같은 3가지 유형으로 개발을 진행 할 수 있다.

(1) 단일 테이블에 대한 단순 CRUD 기능의 경우에는 UI XFDL + Query XML 만을 작성

(2) 복잡한 로직을 가진 기능의 경우에는 UI XFDL + Service + Query XML을 작성

(3) 파일 업로드, 다운로드 등과 같이 표준적인 인터페이스를 가지지 않는 웹 컨트롤러를 개발하는 경우에는 공통 Controller 대신 별도의 Controller + UI XFDL + Service + Query XML을 작성

Anyframe XP Query Plugin에서는 총 15개의 화면 샘플을 제공한다. 이 중 1~14번 화면은 (1)의 방식을, 15번 화면은 (2)의 방식을, 사용자 로그인 기능과 파일 업/다운로드 기능은 (3)의 방식으로 개발되어 있다.

4.1.Presentation Layer

Anyframe XP Query Plugin에 적용된 Web 프레임워크는 MVC모델의 View와 Controller 영역을 담당하며, 이는 프레젠테이션 레이어에 해당된다. 이번 절에서는 프레임워크 기반 개발 시 기본적으로 필요한 설정 파일 구성에 대해서 기술 하도록 한다.

web.xml, *-servlet.xml 파일의 설정은 다음과 같다.

- web.xml

  web.xml은 Web Application배포 지시자로써, JAVA EE 환경에서 해당 Web Application이 서버상에 어떻게 배포되어야 하는지에 대하여 기술하는 XML 파일이다. 프레임워크 기반 개발과 관련하여 web.xml 작성 방법은 Anyframe 매뉴얼을 참조하도록 한다. (web.xml 설정 [http://dev.anyframejava.org/docs/anyframe/plugin/essential/core/1.6.0/reference/htmlsingle/core.html#core_springmvc_configuration_webxml])

- xp-query-servlet.xml

  본 샘플에서는 각각의 샘플 화면들은 공통 Controller인 XPController를 사용하고 있으므로 해당 Controller를 xp-servlet.xml 파일에 Bean으로 등록 하였다. 그 밖의 로그인 기능, 파일첨부 기능을 구현한 Controller의 경우 Annotation 기반으로 작성되었기 때문에 추가적으로 설정 파일에 Bean 등록을 할 필요는 없다.

```
<!-- XPController bean definition -->
<!-- Because XPController can't be scanned by context:component-scan as
 its package isn't "org.anyframe.plugin.*" -->
<bean name="/xpController.do" class="org.anyframe.xp.query.web.XPController" >
 <property name="messageConverter" ref="httpXpMessageConverter" />
</bean>

<!-- HttpXPMessageConverter bean definition for XPLATFORM Data transform -->
<bean id="httpXpMessageConverter"
 class="org.anyframe.xp.query.web.converter.HttpXPMessageConverter" />
```

  위의 Controller 정의에서 볼 수 있듯이, 단순 CRUD 기능의 경우에는 개발자가 별도의 Controller를 구현할 필요 없이 XPController를 사용하면 된다. 다만 파일 업로드/다운로드와 같이 UI와의 인터페이스가 별도로 구성되거나, 또는 XPController에서 지원하지 못하는 기능을 추가하고자 하는 경우, 개발자의 필요에 따라 @RequestBody/@ResponseBody Annotation을 이용하여 Controller를 신규로 작성할 수 있다 . 사용자 정의 Controller 예제는 본 샘플의 LoginController, MovieController 등이 있다.

4.2.Business Layer

본 샘플은 Anyframe XP Query를 기반으로 구현되어 대부분의 비즈니스 로직이 공통화 처리 되어있다. 하지만 특정 비즈니스 로직을 구현 해야 하는 경우 사용자 정의 서비스를 만들어야 하는데, 본 절에서는 사용자 정의 서비스 구현에 필요한 기본적인 설정 파일 구성과 예제에 대해 기술한다.

주요 설정 파일은 다음과 같다.

- mapping-xp-xxx.xml

  QueryService를 사용할 경우 실행하고자 하는 쿼리를 정의한 파일로, 작성방법은 Anyframe 매뉴얼을 참조하도록 한다. (Query Service Manual Page [http://dev.anyframejava.org/docs/anyframe/plugin/optional/query/1.6.0/reference/htmlsingle/query.html#query_mapping])

- context-xp-query.xml

context-xxx.xml은 Spring에서 관리하는 Bean 속성 정의 파일로, context-xp-query.xml에는 Anyframe XP Query의 공통 Service, 공통 Dao에 대한 Bean 정의가 명시되어 있다.

```
<bean name="xpService" class="org.anyframe.xp.query.service.impl.XPServiceImpl">
 <constructor-arg ref="xpDao" />
</bean>

<bean name="xpDao" class="org.anyframe.xp.query.service.impl.XPDao">
 <constructor-arg ref="xpQueryService" />
</bean>
```

공통 서비스를 이용하지 않고 비즈니스 로직을 추가 해야 하는 경우, 먼저 아래의 코드와 같은 XPService를 상속받은 interface 클래스를 생성해야 한다.

```
public interface MovieService extends XPService{

 public void getList(VariableList inVl, DataSetList inDl, VariableList outVl, DataSetList
 outDl) throws Exception;

 public void saveAll(VariableList inVl, DataSetList inDl, VariableList outVl, DataSetList
 outDl) throws Exception;

 public void create(DataSet ds) throws Exception;

 public void get(VariableList inVl, DataSetList inDl, VariableList outVl, DataSetList outDl)
 throws Exception;

 public void update(VariableList inVl, DataSetList inDl, VariableList outVl, DataSetList
 outDl) throws Exception;
}
```

다음으로 앞서 만든 interface 클래스를 구현하는 구현체를 개발 해야 한다. 이 때, XPServiceImpl 클래스를 상속하고, DB 연결을 위해서 XPDao를 Inject 받는다. 아래의 코드는 XPServiceImpl 클래스를 상속받은 예제이다.

```
@Service("xpQueryMovieService")
@Transactional(rollbackFor = {Exception.class}, propagation = Propagation.REQUIRED)
public class MovieServiceImpl extends XPServiceImpl implements MovieService{

 @Inject
 public MovieServiceImpl(XPDao xpDao){
  super.xpDao = xpDao;
 }

 public void getList(VariableList inVL, DataSetList inDl, VariableList outVl, DataSetList
 outDl) throws Exception{
  DataSet outDs = xpDao.getList("findXPMovieList", inDl.get("dsSearch"));
  outDs.setName("dsResult");
  outDl.add(outDs);
 }

 //...중략
```

위의 예제와 같이, 구현한 클래스를 Bean으로 등록하기 위해서 클래스 정의부분 상단에 @Service Annotation을 정의 해야 한다. @Service, @Transaction 등과 같은 Annotation 관련 내용은 Anyframe 매뉴얼을 참고 하도록 한다. (Annotation 설정) [http://dev.anyframejava.org/docs/anyframe/plugin/essential/core/1.6.0/reference/htmlsingle/core.html#core_spring_annotation_mc]

5.Sample UI

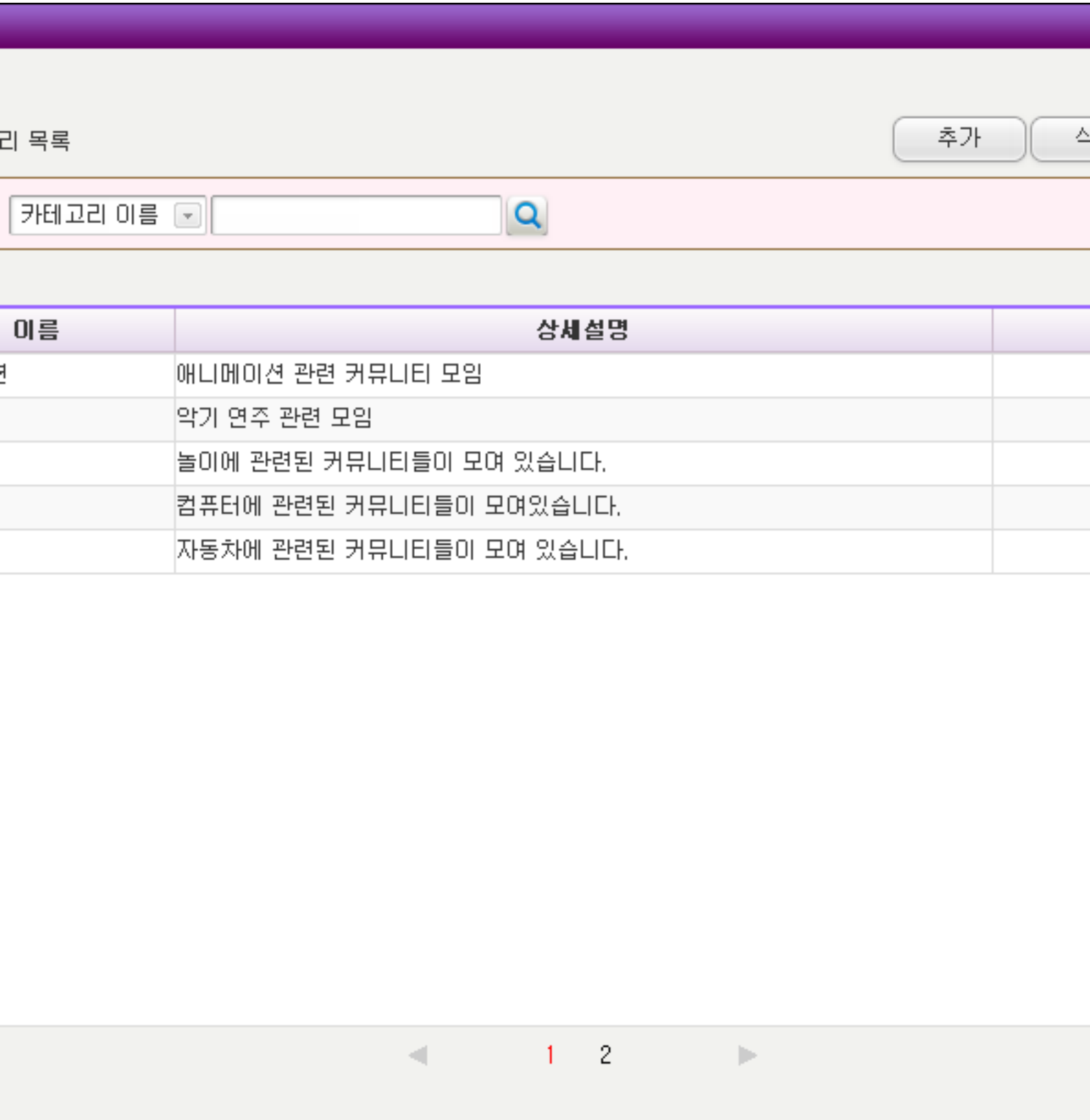
5.1.Introduction

본 샘플 어플리케이션에서 제공하는 화면은 총 15개이다. 각각의 화면은 커뮤니티의 카테고리, 커뮤니티, 각 커뮤니티의 게시글, 사용자 등을 관리하는 기능으로 구성되어 있다. 각 샘플 화면을 통해 UI 컴포넌트의 기본적인 사용법, 이벤트 핸들링, 스크립트 사용법, Anyframe에서 제공하는 공통 모듈을 사용하기 위한 DataSet 구성 방법, Validation 처리 방법 등을 보여준다. 각 UI 컴포넌트의 자세한 사용법은 UX Studio의 Help 매뉴얼을 참조하기 바란다.

5.2.Set of Sample UI

5.2.1.Grid(검색 + Grid 내 입력/수정 + Paging Control + Validation)

커뮤니티 카테고리 목록을 조회하고 카테고리를 관리하는 기능의 화면이다. Grid 하단에는 Paging Control이 존재한다. 본 샘플에서 Paging은 페이지 번호 클릭 시에 해당 페이지의 데이터만 테이블에 가져오는 방식으로 구현되어 있다.

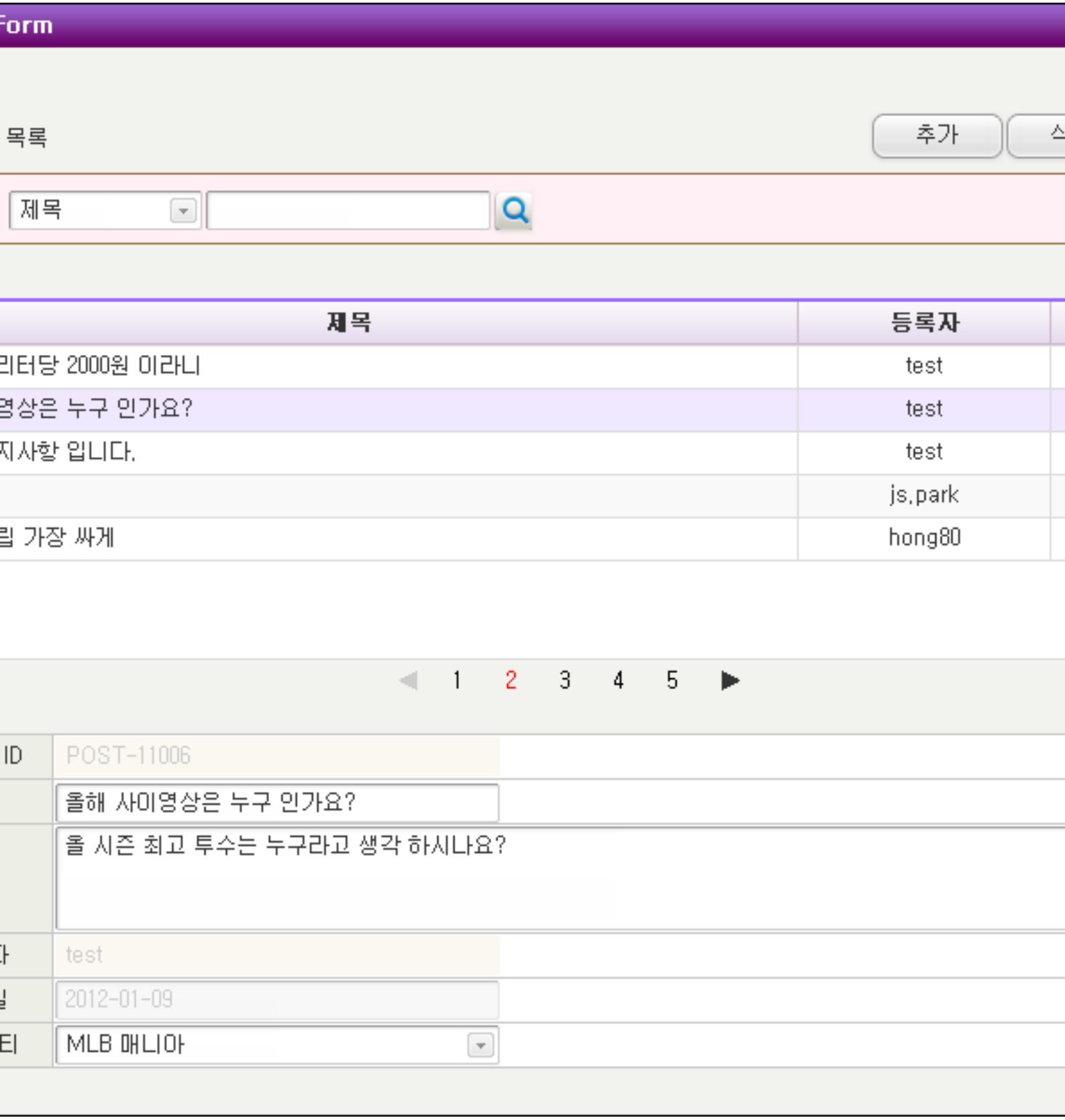


- 추가 버튼을 클릭하면 Grid에 Row가 추가되고 신규 생성할 카테고리 정보를 입력할 수 있다.
- 기존 카테고리 정보를 Grid에서 직접 수정할 수 있다.
- 카테고리 삭제는 Grid에서 삭제할 카테고리를 선택하고 삭제 버튼을 클릭하면 된다. Grid에서 Shift키나 Ctrl키를 이용한 Multi Row 선택이 가능하므로 여러 항목을 한꺼번에 삭제할 수 있다.

- 사용자가 입력/수정/삭제 등 편집한 내용은 저장 버튼을 클릭했을 때 한꺼번에 테이블에 저장된다.

5.2.2.Grid+Form(검색 + Grid +입력Form + Paging Control + Validation)

커뮤니티에 등록된 게시글 목록을 조회하고, 게시글을 관리하는 화면이다. Grid 하단에는 Paging Control이 존재한다.

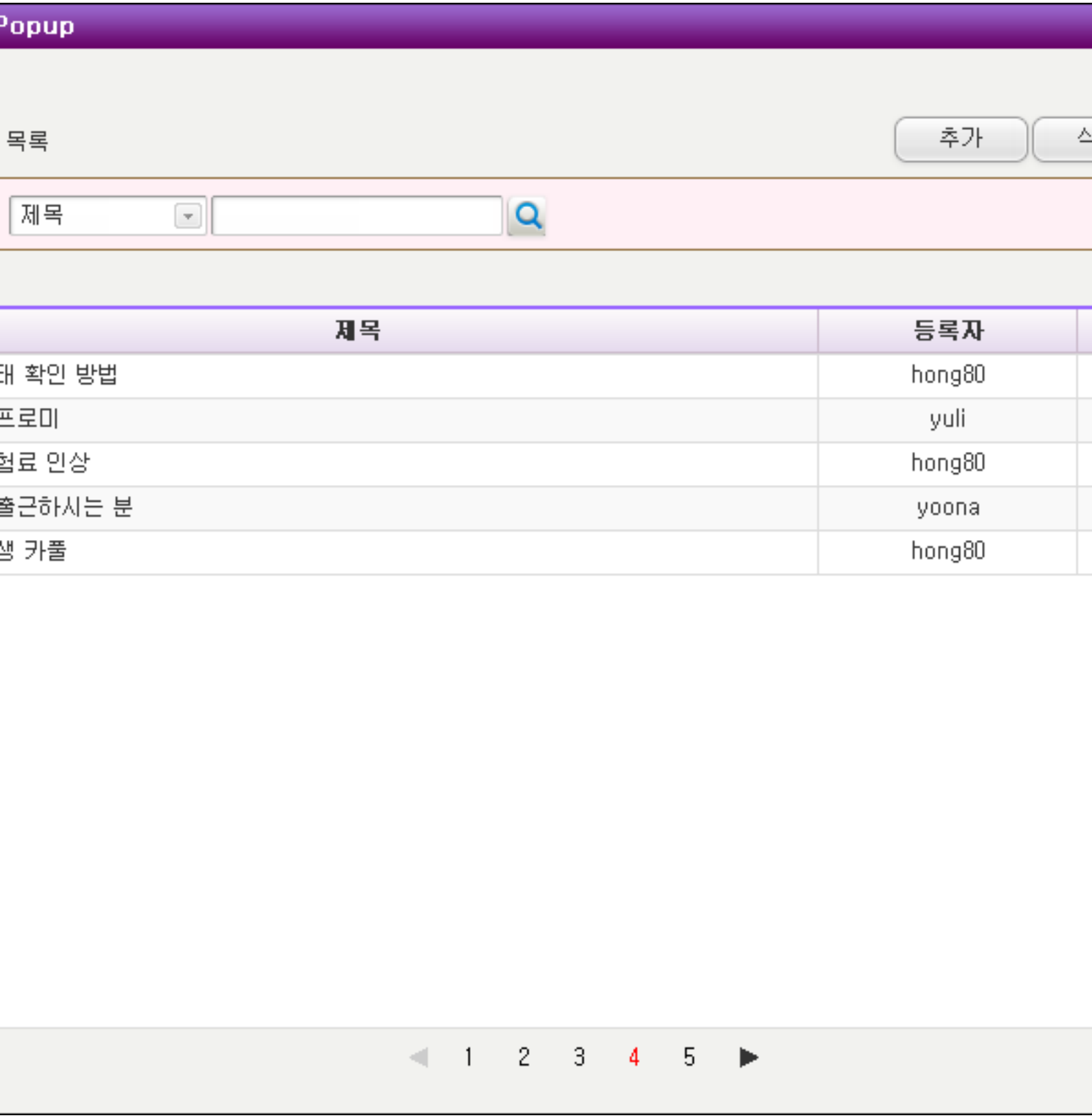


- 목록에서 게시글을 선택하면 하단 입력 Form에서 해당 게시글에 대한 상세 내용을 조회하고 수정할 수 있다.
- 추가 버튼을 클릭하면 Grid에 Row가 추가되고 하단에 빈 입력 Form에서 신규 게시글을 작성할 수 있다.

- 게시글 삭제는 Grid에서 삭제할 게시글을 선택하고 삭제 버튼을 클릭하면 된다. Grid에서 Shift키나 Ctrl키를 이용한 Multi Row 선택이 가능하므로 여러 항목을 한꺼번에 삭제할 수 있다.
- 사용자가 입력/수정/삭제 등 편집한 내용은 저장 버튼을 클릭했을 때 한꺼번에 테이블에 저장된다.

5.2.3.Grid+Popup(검색 + Grid + 입력Form 팝업 + Paging Control + Validation)

커뮤니티에 등록된 게시글 목록을 조회하고, 게시글을 관리하는 화면이다. Grid 하단에는 Paging Control이 존재한다.

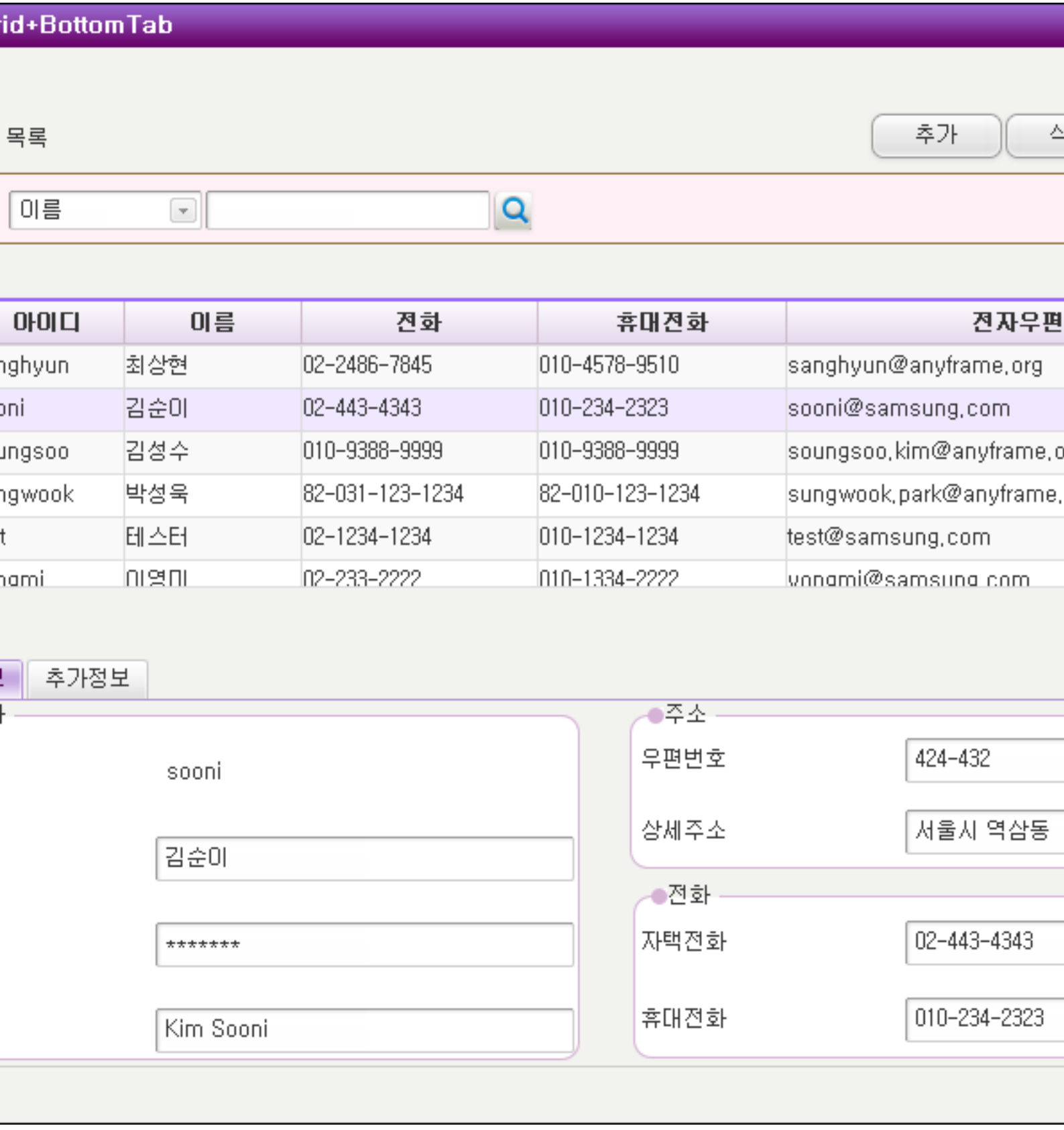


- 목록에서 게시글을 선택하여 더블 클릭하면 상세 내용을 조회하고 수정할 수 있는 팝업 창이 나타난다.
- 추가 버튼을 클릭하면 빈 입력 Form이 팝업 창으로 나타나고 신규 게시글을 작성할 수 있다.
- 게시글 삭제는 Grid에서 삭제할 게시글을 선택하고 삭제 버튼을 클릭하면 된다. Grid에서 Shift키나 Ctrl키를 이용한 Multi Row 선택이 가능하므로 여러 항목을 한꺼번에 삭제할 수 있다.

- 사용자가 입력/수정/삭제 등 편집한 내용은 저장 버튼을 클릭했을 때 한꺼번에 테이블에 저장된다.

5.2.4.TopGrid+BottomTab(검색 + 상단 Grid + 하단 Tab 입력 Form + Validation)

사용자 목록을 조회하고, 관리하는 화면이다. 하단에는 사용자의 상세정보를 조회할 수 있는 Tab이 존재한다.

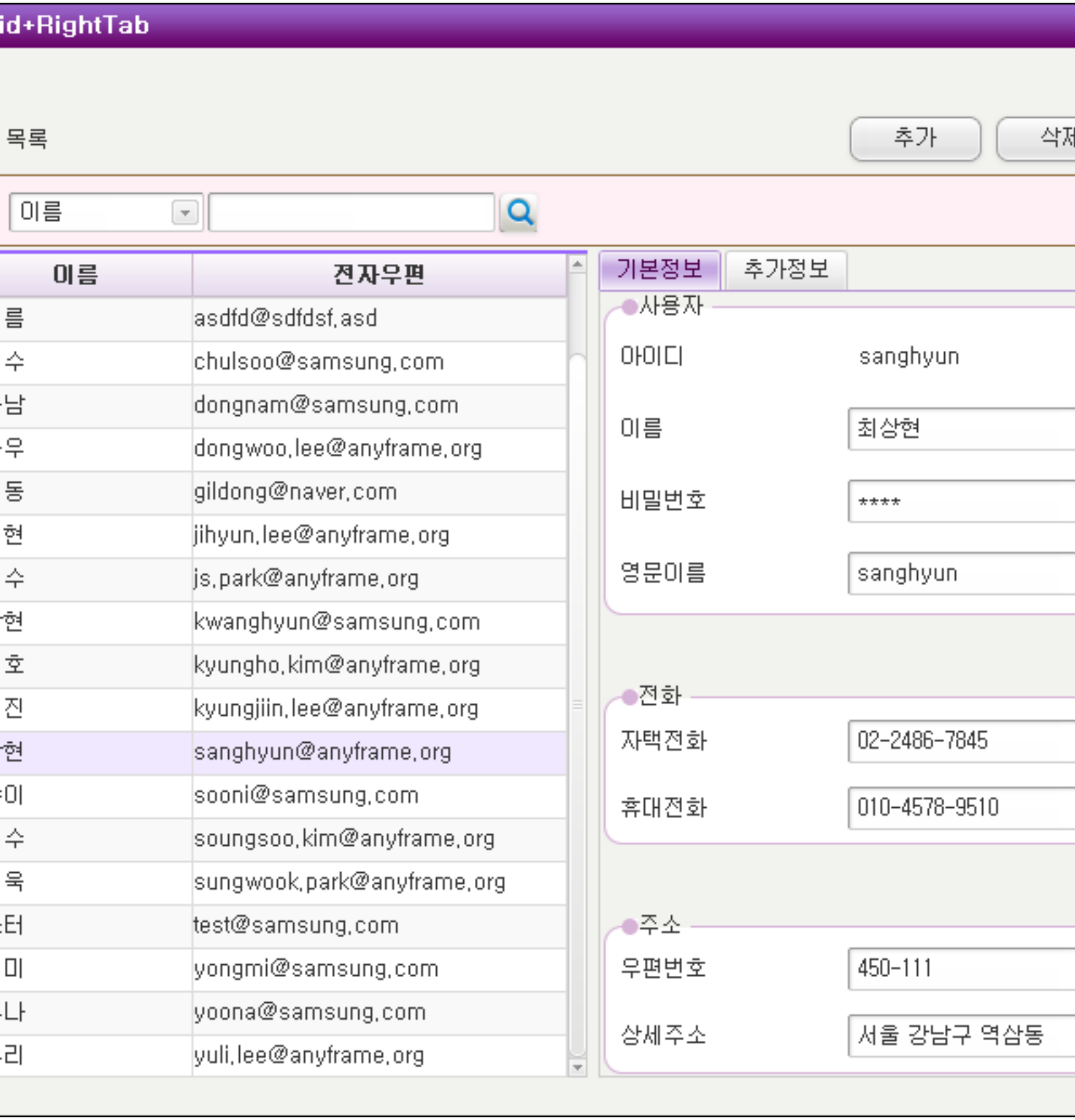


- 목록에서 사용자를 선택하면 하단 Tab 입력 Form에서 해당 사용자에 대한 상세 내용을 조회하고 수정할 수 있다.
- 추가 버튼을 클릭하면 Grid에 Row가 추가되고 하단에 빈 Tab 입력 Form이 나타나 신규 사용자를 생성할 수 있다.

- Grid에서 삭제할 사용자의 Checkbox에 체크를 하고 삭제 버튼을 클릭하면 사용자를 삭제할 수 있다. 또한 Checkbox를 이용해서 여러 사용자를 한꺼번에 삭제할 수 있다.
- 입력/수정/삭제 등 편집한 내용은 저장 버튼을 클릭했을 때 한꺼번에 테이블에 저장된다.

5.2.5.LeftGrid+RightTab(검색 + 좌측 Grid + 우측 Tab 입력Form + Validation)

사용자 목록을 조회하고, 관리하는 화면이다. 우측에 사용자의 상세정보를 조회할 수 있는 Tab이 존재한다.

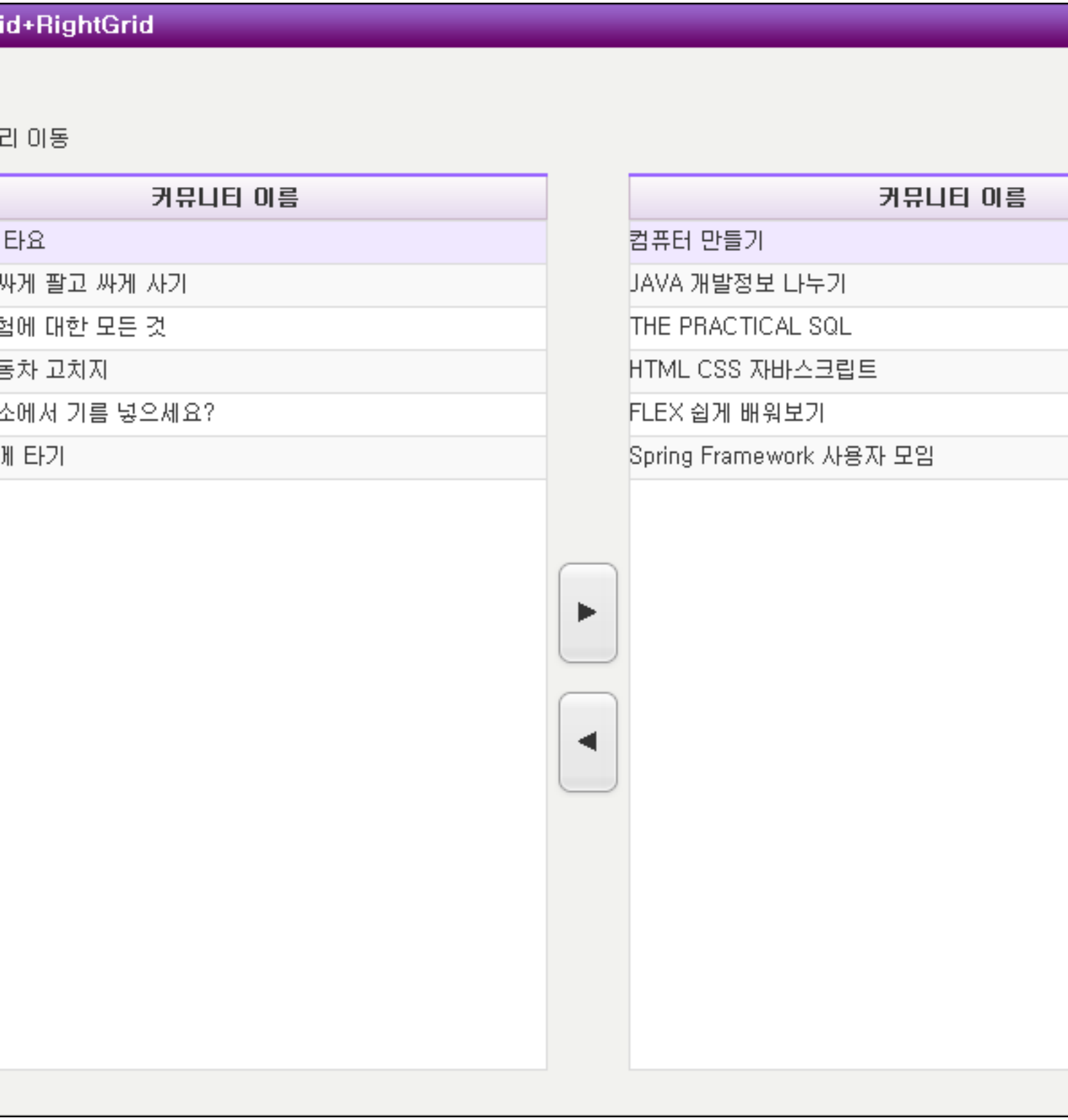


- 목록에서 사용자를 선택하면 우측 Tab 입력 Form에서 해당 사용자에 대한 상세 내용을 조회하고 수정할 수 있다.
- 추가 버튼을 클릭하면 Grid에 Row가 추가되고 우측에 빈 Tab 입력 Form이 나타나 신규 사용자를 생성할 수 있다.

- Grid에서 삭제할 사용자의 Checkbox에 체크를 하고 삭제 버튼을 클릭하면 사용자를 삭제할 수 있다. 또한 Checkbox를 이용해서 여러 사용자를 한꺼번에 삭제할 수 있다.
- 입력/수정/삭제 등 편집한 내용은 저장 버튼을 클릭했을 때 한꺼번에 테이블에 저장된다.

5.2.6.LeftGrid+RightGrid(좌/우Grid 간 항목 이동)

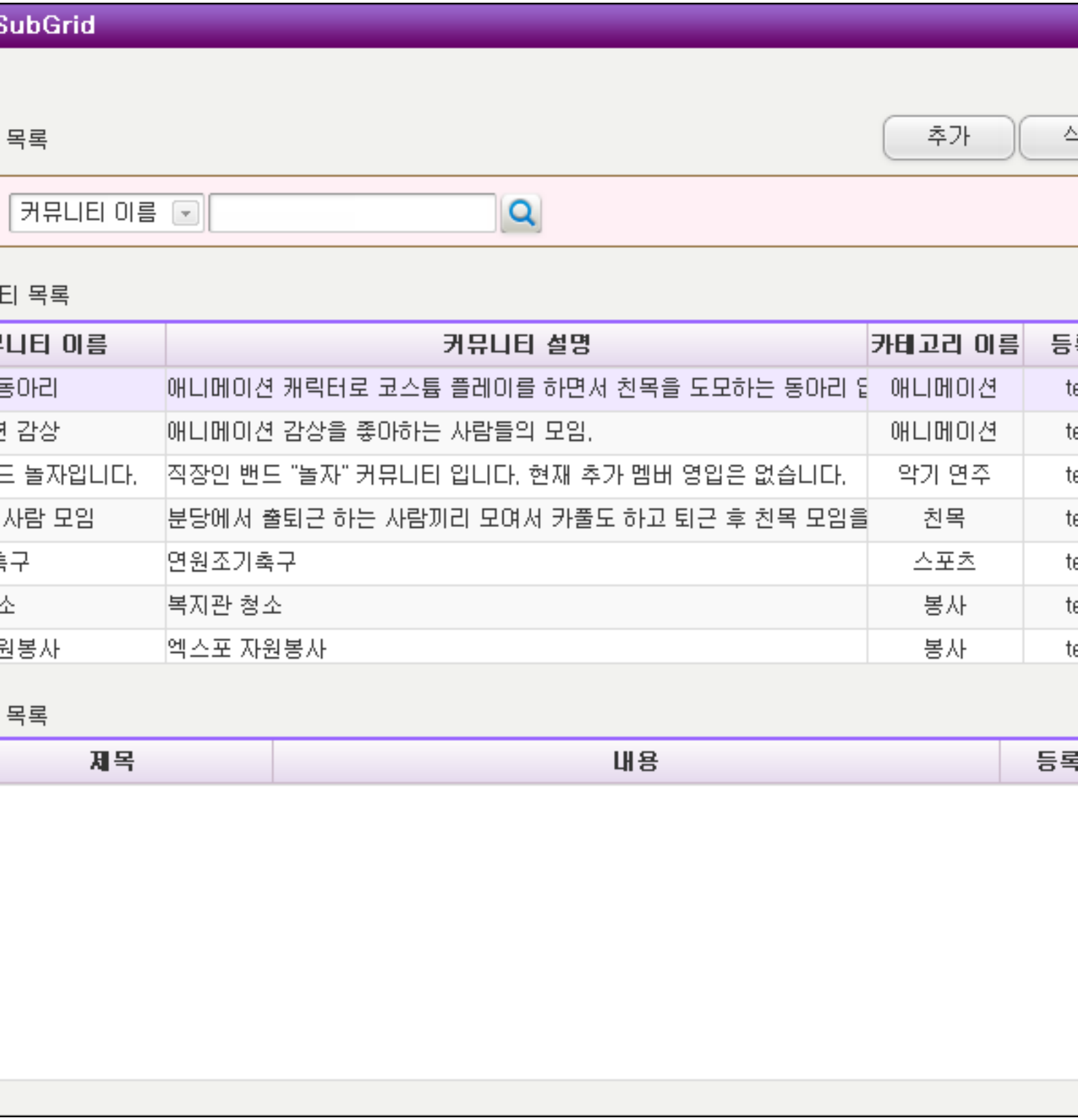
커뮤니티의 카테고리를 이동하는 화면이다. 좌측에 컴퓨터 카테고리에 속한 커뮤니티 목록이 나열되고, 우측에 자동차 카테고리에 속한 커뮤니티가 나열된다. 화면 가운데 버튼을 이용하여 커뮤니티를 이동시킬 수 있다. 좌/우 Grid 간 항목 이동 방법을 보여주는 샘플이다.



- 목록에서 커뮤니티를 선택하고 좌측으로 이동하는 버튼 또는 우측으로 이동하는 버튼을 클릭하면 Grid에서 커뮤니티가 이동한다.
- 편집한 내용은 저장 버튼을 클릭했을 때 한꺼번에 테이블에 저장된다.

5.2.7.Grid+SubGrid(검색 + 상단 Master Grid + 하단 Sub Grid + Validation)

커뮤니티 목록과 커뮤니티에 등록된 게시글 목록을 조회하고, 게시글을 관리할 수 있는 화면이다. 하단의 커뮤니티 목록은 상단의 카테고리 목록을 클릭했을 때 해당 카테고리 ID를 가지고 XPLATFORM DataSet Method API인 filter()를 사용해서 출력한다. 하단 커뮤니티 목록에서 입력데이터에 대한 Validation 처리가 적용되어 있다.

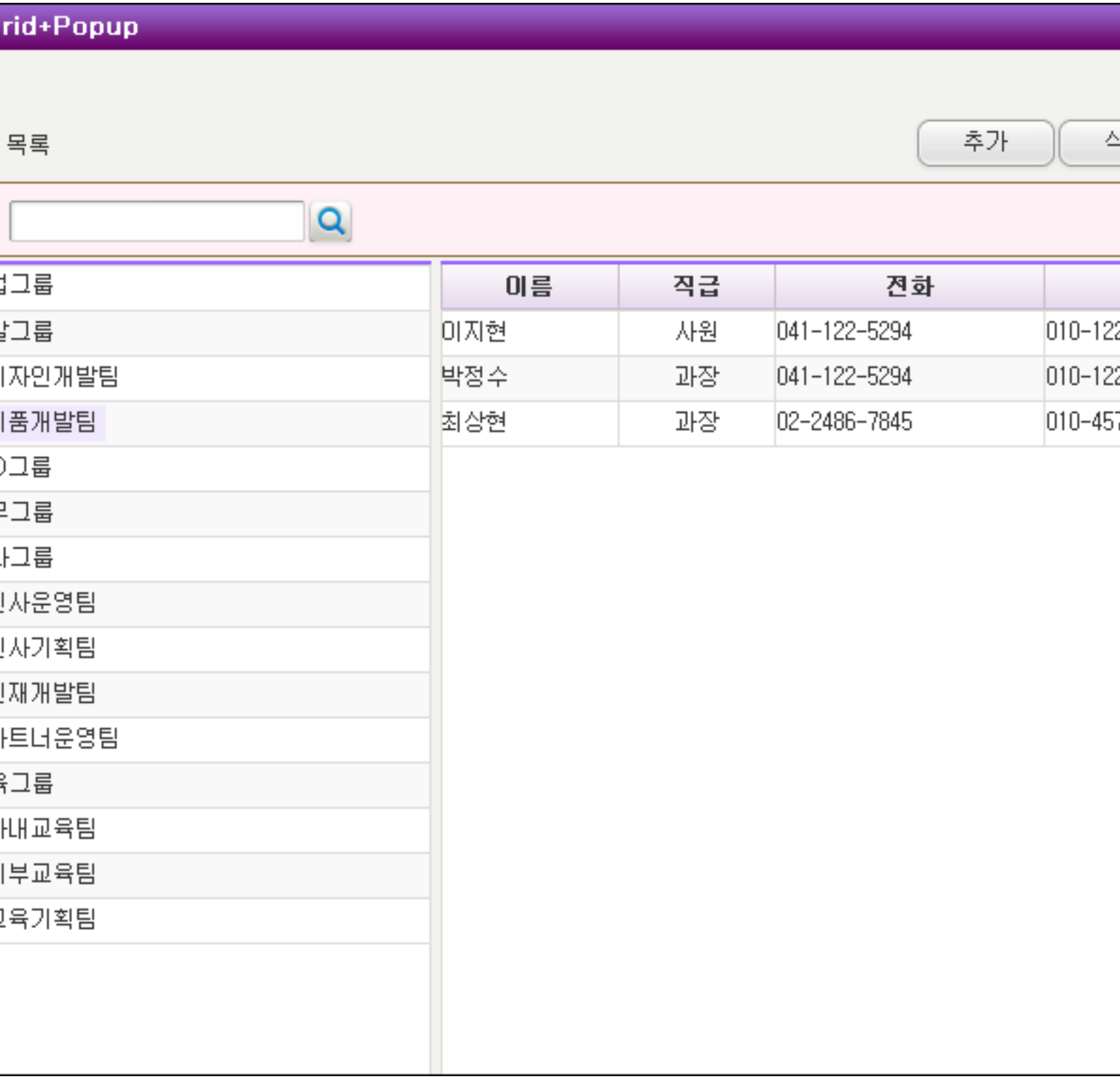


- 상단 Master Grid에서 커뮤니티를 선택하면 하단 Sub Grid에 해당 커뮤니티에 속한 게시글 목록이 나타난다.
- 추가 버튼을 클릭하면 Sub Grid에 Row가 추가되고 신규 게시글을 작성할 수 있다.
- 기존 게시글 정보를 Grid에서 직접 수정할 수 있다.

- Grid에서 삭제할 게시글의 Checkbox에 체크를 하고 삭제 버튼을 클릭하면 된다. 또한 Checkbox를 이용해서 여러 게시글을 한꺼번에 삭제할 수 있다.
- 사용자가 입력/수정/삭제 등 편집한 내용은 저장 버튼을 클릭했을 때 한꺼번에 테이블에 저장된다.
- Grid 입력 데이터에 대해 공통Script함수인 gfnDsCheckValid()를 이용한 Validation 처리가 적용되어 있다.

5.2.8.TreeGrid+Popup(자동완성기능 + 검색 + Tree + Grid + Tab 입력 Form 팝업 + Validation)

부서 목록을 Tree로 조회하고, 부서에 속한 사용자를 관리하는 화면이다. Tree와 Grid 사용 방법을 제시하고 있다. 입력데이터에 대한 Validation 처리가 적용되어 있다. Tree 검색을 돕기 위한 검색어 자동완성 기능 또한 구현 되어 있다.

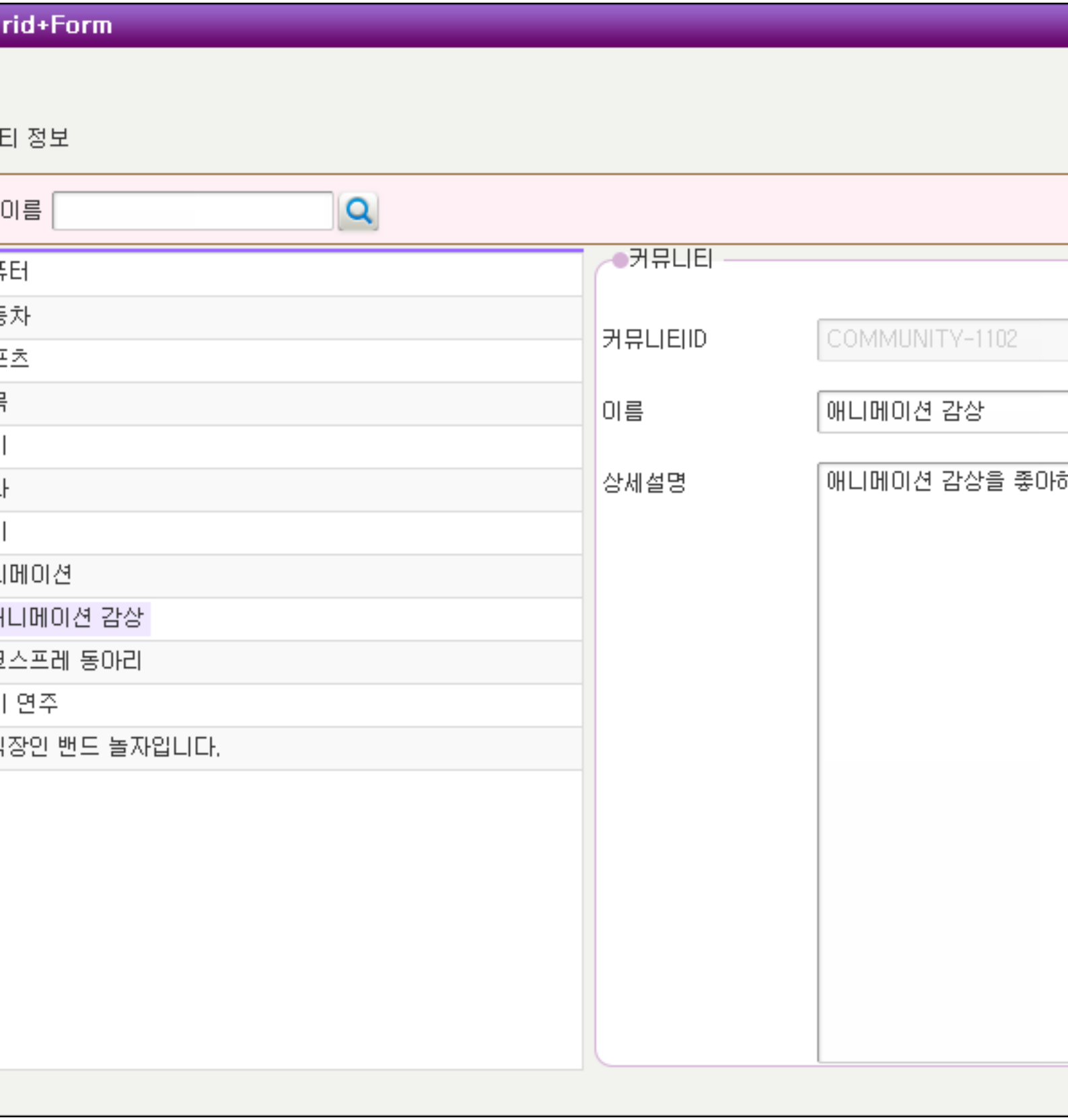


- 좌측 Tree에서 부서를 선택하면 우측 Grid에 해당 부서에 속한 사용자 목록이 나열된다.
- 우측 목록에서 사용자를 선택하여 더블 클릭하면 사용자에 대한 상세 내용을 조회하고 수정할 수 있는 Tab 입력 Form 팝업 창이 나타난다.
- 추가 버튼을 클릭하면 빈 Tab 입력 Form 팝업 창이 나타나 신규 사용자를 생성할 수 있다.
- 사용자 삭제는 Grid에서 삭제할 사용자를 선택하고 삭제 버튼을 클릭하면 된다. Grid에서 Shift키나 Ctrl키를 이용한 Multi Row 선택이 가능하므로 여러 항목을 한꺼번에 삭제할 수 있다.

- 입력/수정/삭제 등 편집한 내용은 저장 버튼을 클릭했을 때 한꺼번에 테이블에 저장된다.
- 공통Script함수인 gfnDsCheckValid()를 이용한 Validation 처리가 적용되어 있다.

5.2.9.TreeGrid+Popup(검색 + Tree + 입력 Form)

카테고리와 커뮤니티 목록을 Tree로 조회하고, 커뮤니티 정보를 관리하는 화면이다. Tree와 입력 Form 사용 방법을 제시하고 있다.

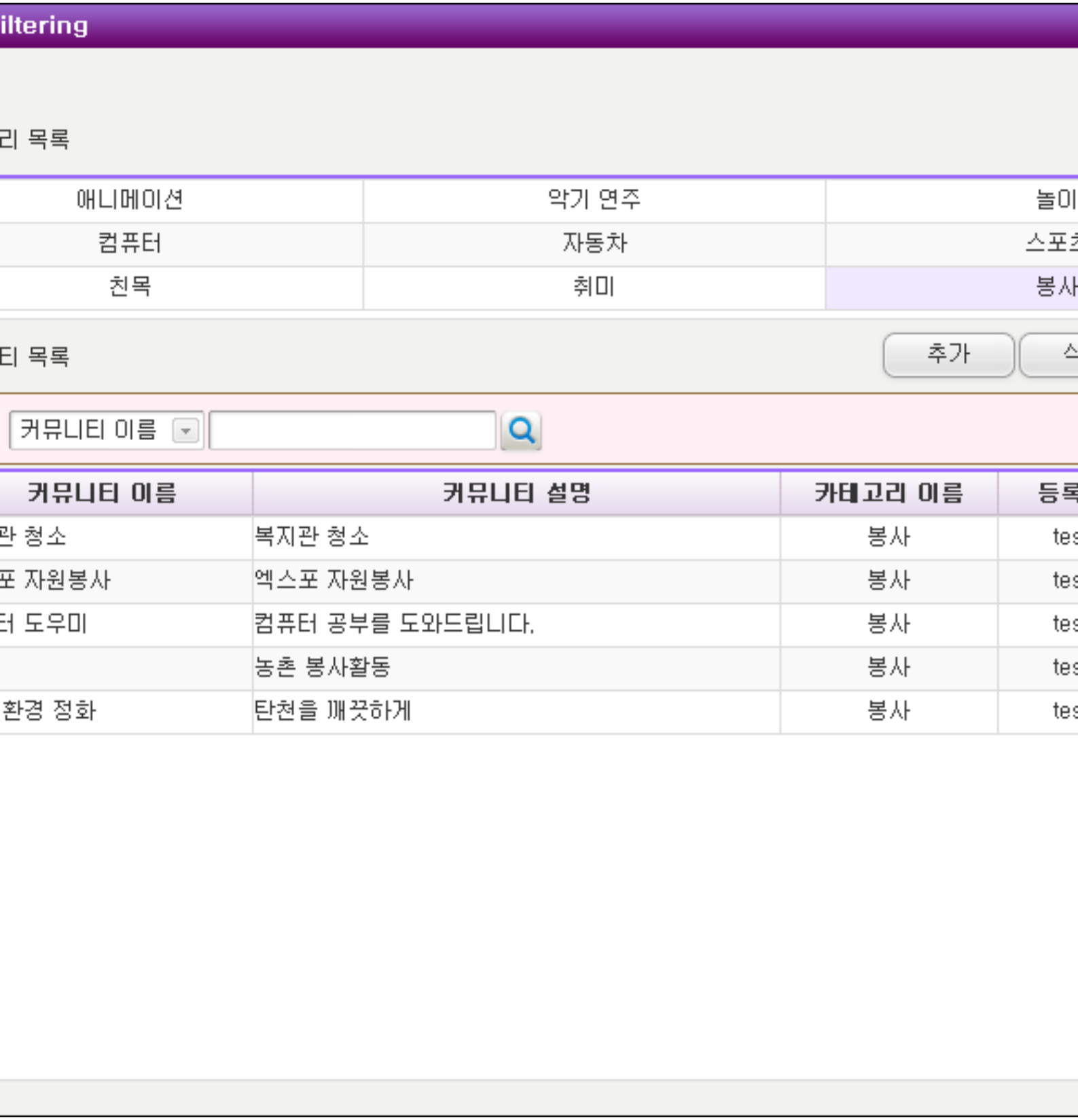


- 좌측 Tree에서 카테고리를 선택하면 우측에 카테고리 정보가 나타난다.
- 좌측 Tree에서 커뮤니티를 선택하면 우측에 커뮤니티 정보가 나타난다.
- 수정한 커뮤니티 정보는 저장버튼을 클릭했을 때 일괄 저장된다.

- Tree 검색 시 자동 완성 기능이 추가 되어 있지 않다. 사용자는 정확한 커뮤니티 이름을 검색어로 입력 해야 한다.

5.2.10.Grid Filtering(카테고리구분 + 검색 + Grid)

카테고리 별 커뮤니티 목록을 조회하는 화면이다. Grid를 이용하여 카테고리 구분표를 구성하는 방법을 제시하고 있다. 하단의 커뮤니티 목록은 상단의 카테고리를 클릭했을 때 해당 카테고리 ID를 가지고 XPLATFORM DataSet Method API인 filter()를 사용해서 출력한다.

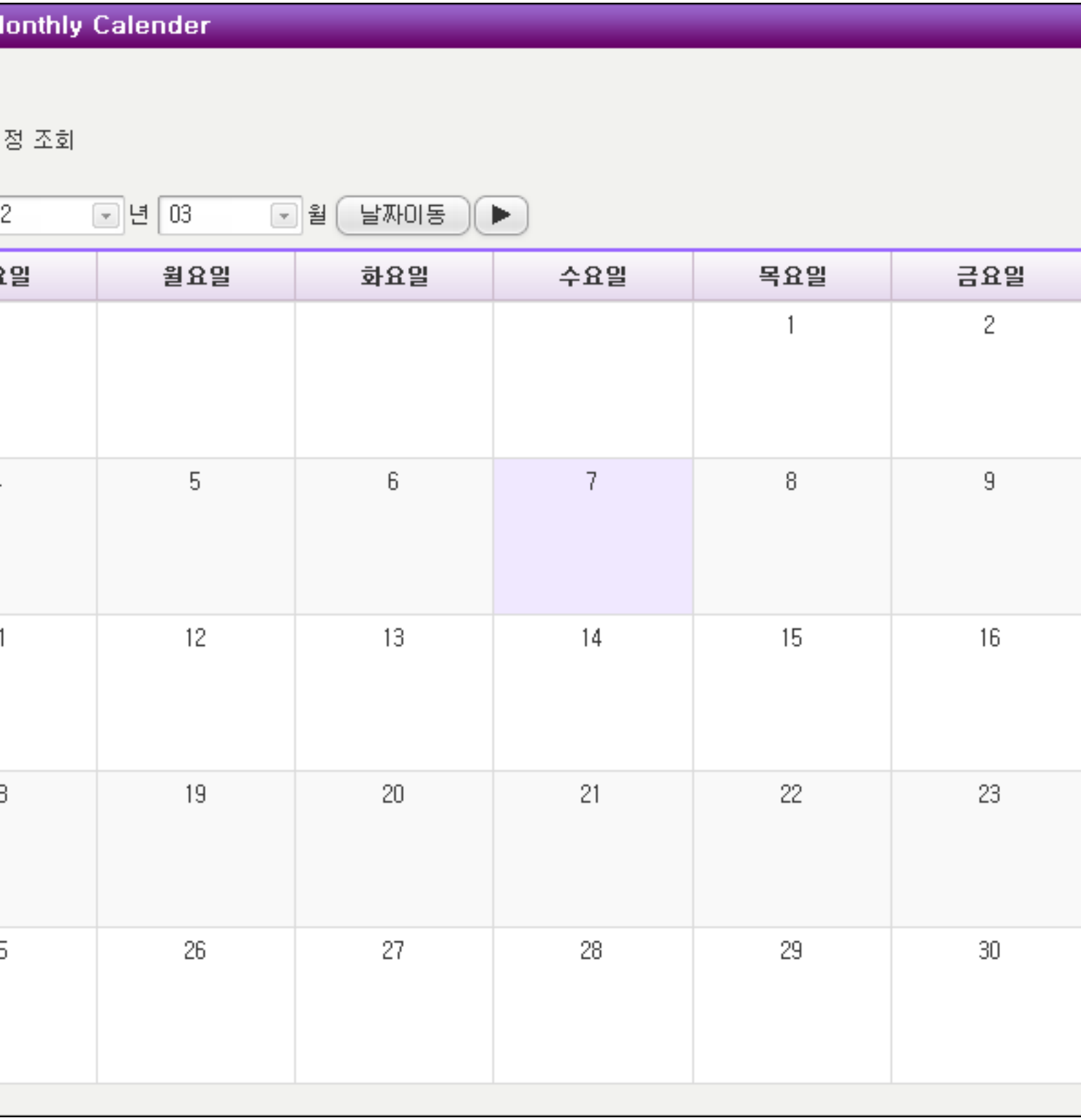


- 상단에서 카테고리를 선택하면 하단에 커뮤니티 목록이 나열된다.
- 추가 버튼을 클릭하면 Grid에 Row가 추가되고 신규 생성할 커뮤니티 정보를 입력할 수 있다.
- 기존 커뮤니티 정보를 Grid에서 직접 수정할 수 있다.

- 삭제할 커뮤니티의 Checkbox를 체크하고 삭제 버튼을 클릭하면 커뮤니티를 삭제할 수 있다. Checkbox를 이용하여 여러 개의 항목을 한꺼번에 삭제할 수 있다.
- 사용자가 입력/수정/삭제 등 편집한 내용은 저장 버튼을 클릭했을 때 한꺼번에 테이블에 저장된다.

5.2.11.Grid Monthly Calender(Grid를 이용한 월간 Calendar)

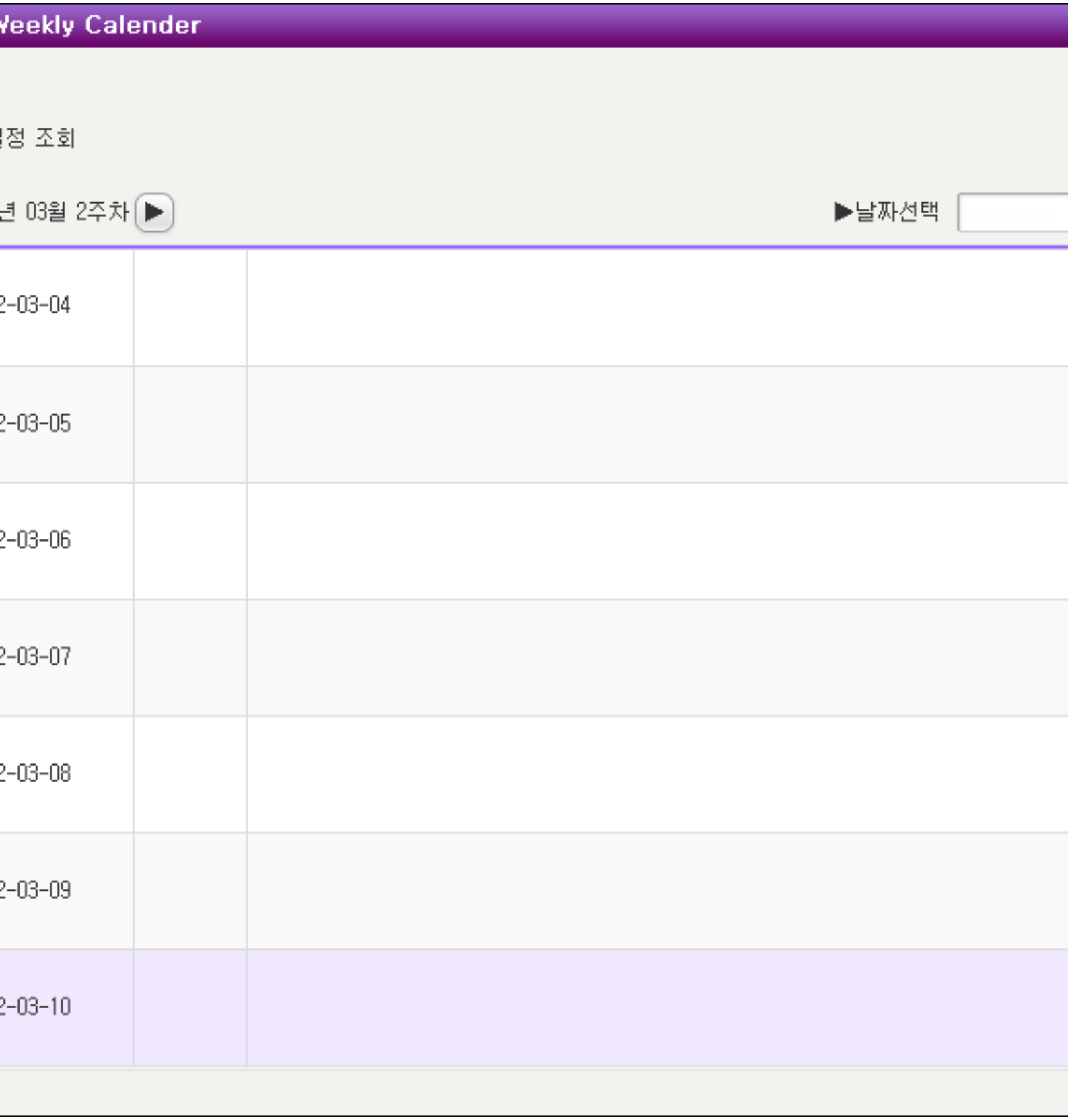
월간 일정 관리 화면이다. Grid를 이용하여 Calendar를 구성하는 방법을 보여준다.



- 버튼을 이용하여 월별로 이동하며 일정을 조회할 수 있다.
- 기존에 정의한 일정이 존재하는 경우 해당 날짜를 더블 클릭하면 일정 상세조회 팝업 창이 나타난다.
- 일정이 정의되지 않은 날짜를 더블 클릭하면 신규 일정을 추가할 수 있는 팝업 창이 나타난다.

5.2.12.Grid Weekly Calender(Grid를 이용한 주간 Calendar)

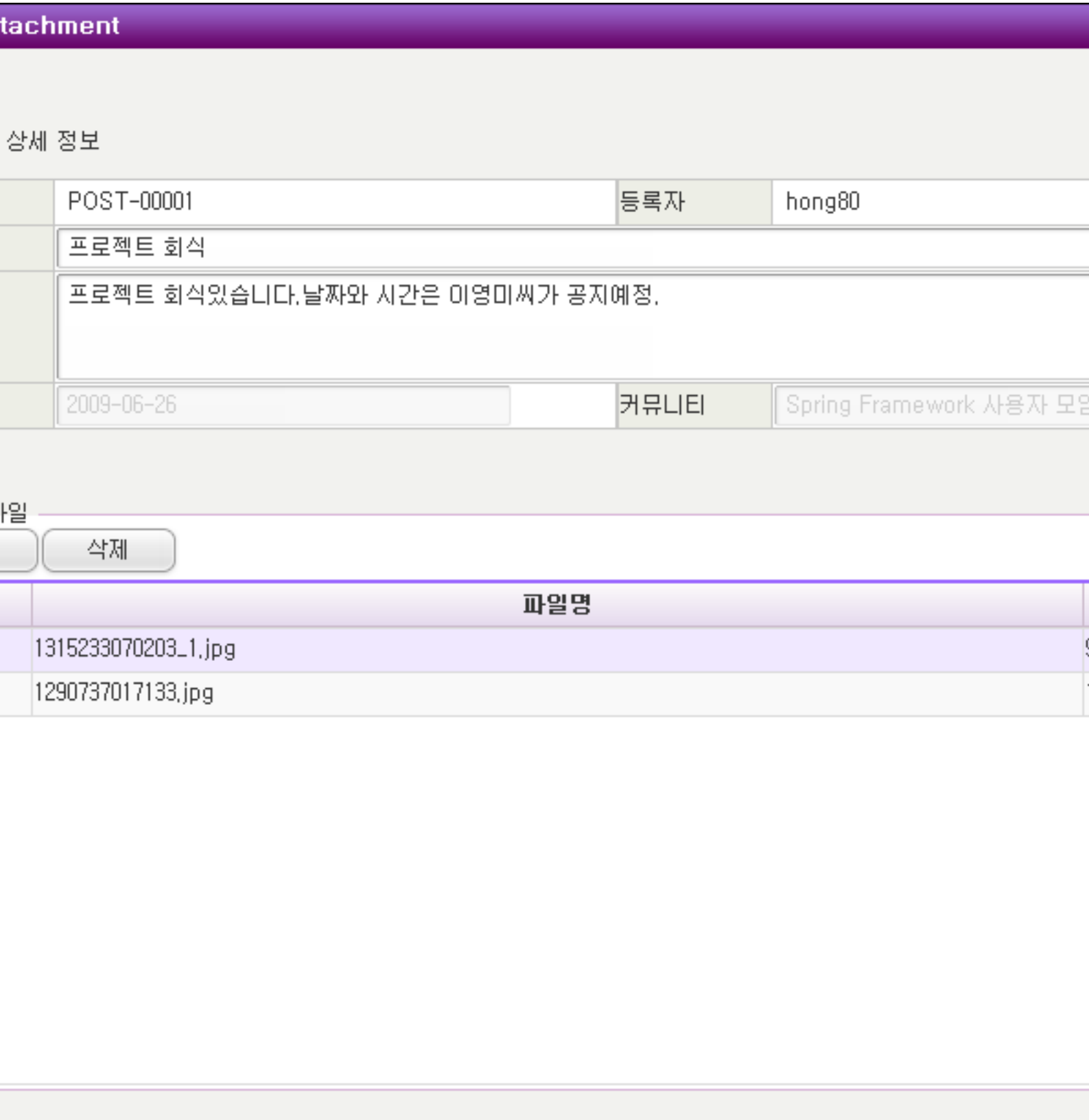
주간 일정 관리 화면이다. Grid를 이용하여 Calendar를 구성하는 방법을 보여준다.



- 버튼을 이용하여 주 별로 이동하며 일정을 조회할 수 있다.
- 기존에 정의된 일정이 존재하는 경우 해당 날짜를 더블 클릭하면 일정 상세조회 팝업 창이 나타난다.
- 일정이 정의되지 않은 날짜를 더블 클릭하면 신규 일정을 추가할 수 있는 팝업 창이 나타난다.

5.2.13.File Attachment(파일 첨부)

'POST-00001'이라는 ID의 게시글의 상세 정보 조회화면으로, File 컴포넌트와 File Dialog 컴포넌트를 사용하여 파일 첨부 기능을 구현한 샘플화면이다. 첨부된 파일을 context.properties 파일에 정의된 repository.path 상에 지정된 위치로 업로드를 한다.



- 추가 버튼을 클릭하면 파일을 하나씩 첨부할 수 있는 File Dialog 창이 나타난다.
- 첨부 파일 목록에서 첨부파일을 더블 클릭하면 첨부파일을 저장할 위치와 파일 이름을 지정할 수 있는 Dialog가 나타난다.
- 삭제할 첨부파일의 Checkbox를 체크하고 삭제버튼을 클릭하면 첨부파일을 삭제할 수 있다. Checkbox를 이용하여 여러 개의 항목을 한꺼번에 삭제할 수 있다.
- 사용자가 편집한 내용은 저장 버튼을 클릭했을 때 한꺼번에 테이블에 저장된다.

5.2.14.Grid Total Sum(총합, 평균)

Grid Cell의 속성 중 expr 속성을 이용하여 특정 Cell의 합을 구하는 Column을 만들고, Grid Summary 영역을 추가하여 특정 Column에 대한 총합, 평균을 표현하는 예제이다.

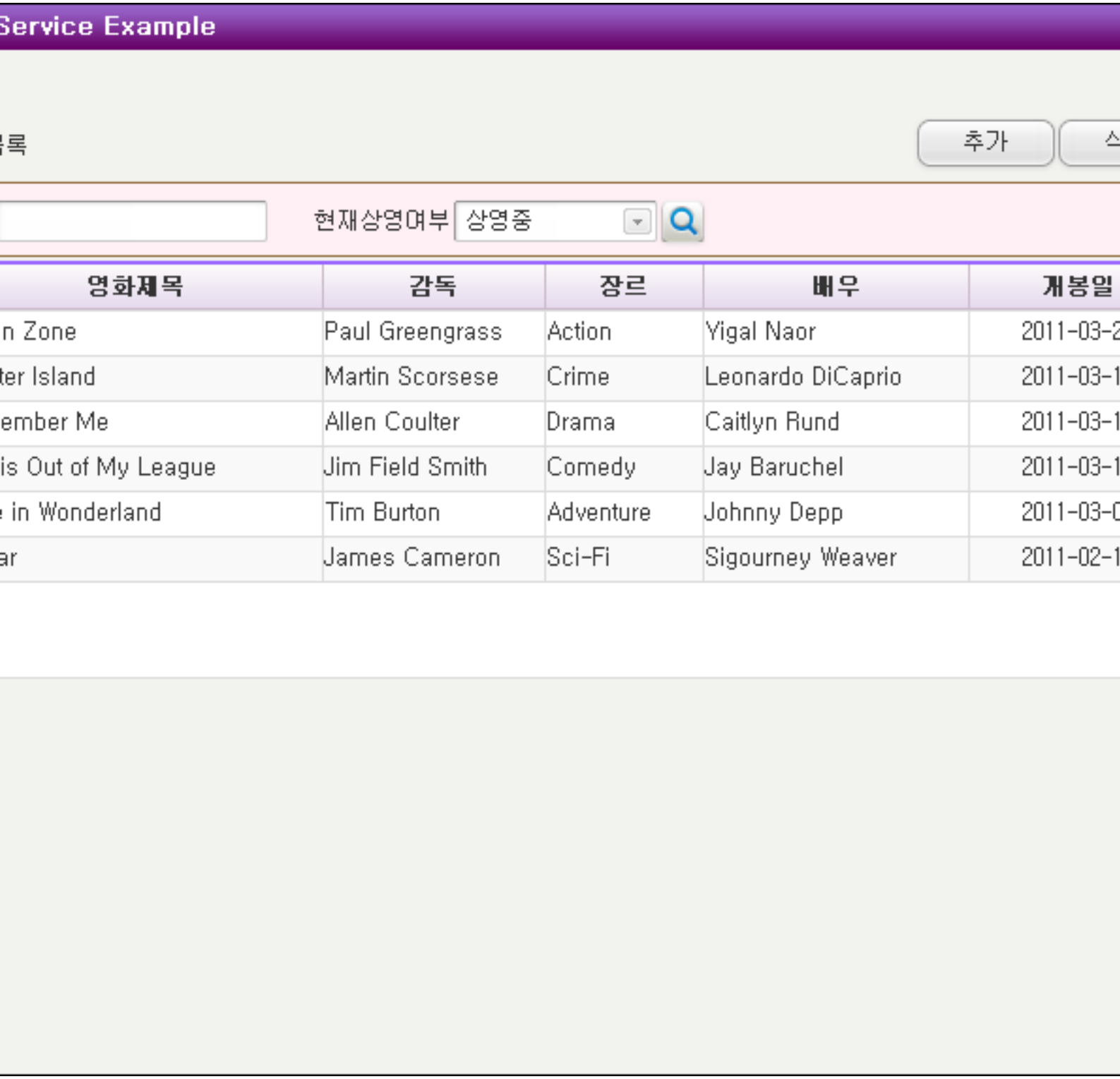
Total Sum

동 내역

회사	부서	이름	직급	상반기 봉사실적	하반기 봉사
성SDS	영업그룹	김미름		0	
		이동우	과장	8	
		김경호	부장	2	
		이경진	사원	3	
		박성욱	과장	6	
		이유리	사원	3	
	RnD그룹	김광현	사원	1	
		테스터	대리	2	
	총무그룹	김성수	대리	6	
	국내영업1팀	윤동남	차장	11	
		홍길동	책임	8	
	국내영업2팀	김철수	사원	10	
	해외영업팀	이영미	대리	3	
	디자인개발팀	김순이	부장	5	
	제품개발팀	이지현	사원	7	
		박정수	과장	6	
		최상현	과장	4	
	RnD2팀	김유나	사원	3	
누계				93	
평균				4.9	

5.2.15.User Service Example(사용자 정의 서비스 예제)

Anyframe XP Query에서 제공하는 공통 Flow를 이용하지 않고 Controller, Service Class를 직접 생성해서 호출하는 예제이다.



- 추가 버튼을 클릭하면 영화 상세 정보를 입력할 수 있는 팝업창이 나타난다. 팝업창에서 정보를 입력한 후 저장 버튼을 클릭하면 유효값 인증 절차를 거친 후 /xpQueryMovie.do 경로를 호출하여 MovieController의 create method를 호출하여 정보를 저장 하게 된다.
- 특정 영화를 더블클릭 하면 /xpQueryMovie.do 경로를 호출하여 MovieController의 get Method를 호출하여 해당 Movie ID를 가지는 영화 정보를 보여주는 팝업창이 생성된다. 팝업창에서 데이터를 변경 후 저장 버튼을 클릭하면 /xpQueryMovie.do 경로를 호출하여 MovieController의 update Method를 호출하여 변경 내역을 수정하게 된다.
- Grid 화면의 체크 박스로 영화를 선택한 후 삭제 버튼을 눌러 원하는 영화 정보를 삭제 할 수 있다. 삭제를 완료한 후 Grid 상단의 저장 버튼을 눌러서 변경된 데이터를 저장 할 수 있다. 저장 버튼을 클릭하면 /xpQueryMovie.do 경로를 호출하여 MovieController의 saveAll Method를 호출하여 변경 내역을 수정 하게 된다.

6.Standards

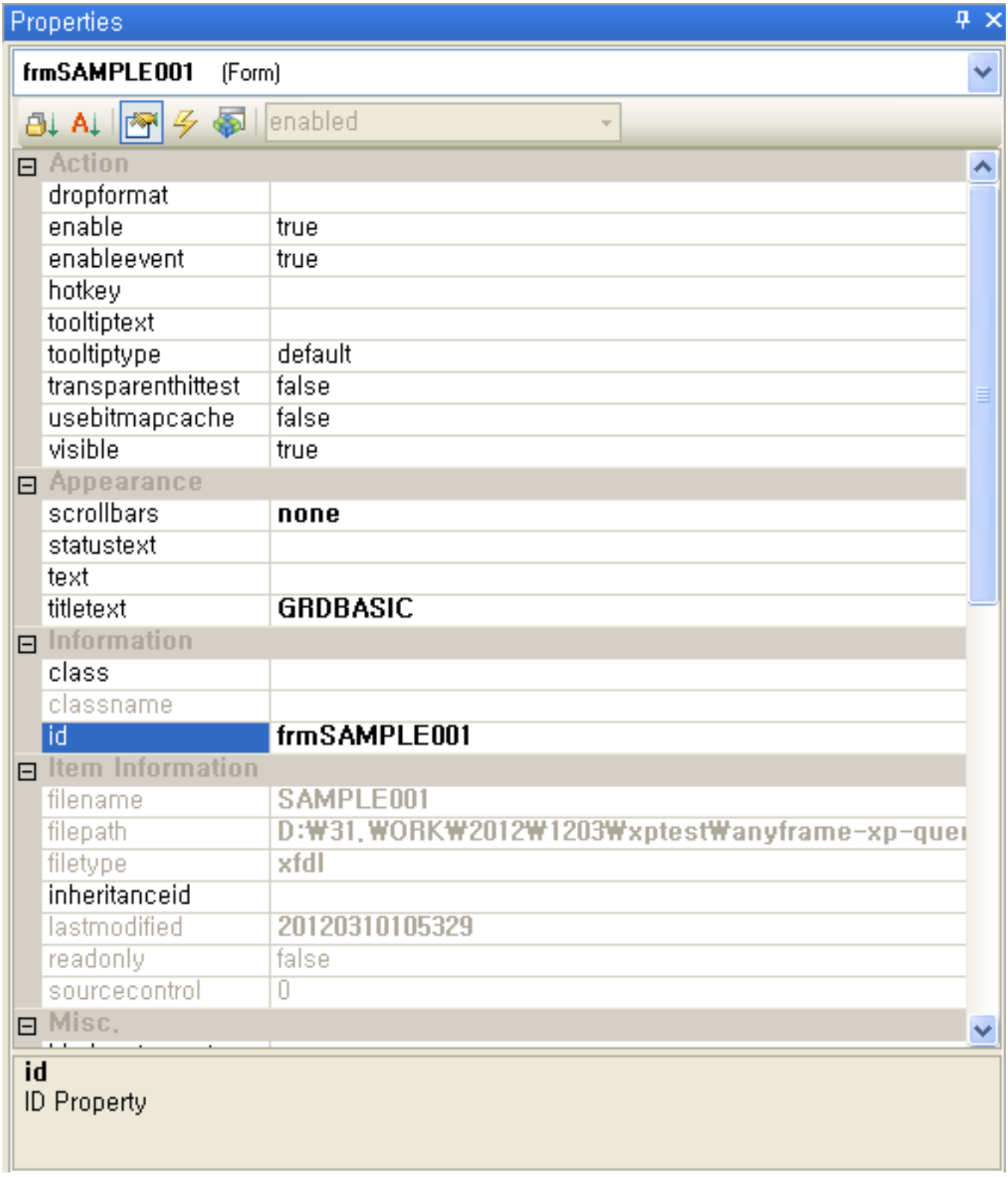
6.1.Naming Rules of Form

화면 파일 하나당 Form은 하나로 구성되어 있으며 Form의 ID는 "frm"+ 파일명 형태로 이루어진다.

표 6.1. 예)

File Name	Example of Form ID
01GRD.xml	frm01GRD
CategoryMgmt.xfdl	frmCategoryMgmt

UX Studio에서 Form ID는 Form의 속성으로 지정한다.



6.2.Naming Rules of UI Component

UI 컴포넌트를 나타내는 Prefix와 의미 있는 이름을 조합하여 정의한다.

- UI 컴포넌트 별 영문 약어를 Prefix로 사용
- 단어와 단어 사이는 단어 첫 알파벳을 대문자로 시작하여 구분

예) 사업장 ‘Combo’: "cmbPlantCd", 목록을 출력하는 ‘Grid’: "grdTitleList", 업체명을 수정하는 ‘Edit’: "edtClientNm", 조회 ‘Button’: "btSearch", 저장 ‘Button’: "btSave", 초기화 ‘Button’: "btReset",

- DataSet의 이름은 Prefix ds + ‘Dataset이 Binding되는 컴포넌트의 이름’으로 정의

  예) dsCmbRepairItem

- DataSet의 컬럼 ID는 DB Table 컬럼 명을 그대로 사용

UI 컴포넌트 별 Prefix는 다음과 같다.

표 6.2.

UI Component	Prefix	Example of Component ID
Button	bt	btSave, btSearch
Calendar	cal	calFromDt, calToDt
Checkbox	chk	chkAll
ComboBox	cmb	cmbAccItem
DataSet	ds	dsUser, dsMenu
Division	div	divUserInfo
Edit	edt	edtUserNm
File	file	fileUserImg
FileDialog	fdlg	fdlgUserImg
Grid	grd	grdClientList
Image	img	imgTitle
List	lst	lstMenu
MaskEdit	msk	mskAmount
MenuBar	mb	mbTopMenu
Pie	pie	pieChart
PopupDiv	pdiv	pdivMemo
Progressbar	pb	pbLoading
Radio	rdo	rdoYn
Shape	shp	shpBox
Spin	sp	spAddVal
Static	st	stName
Tab	tab	tabClient
TeeChart	tc	tcIncome
TextArea	txa	txaMemo

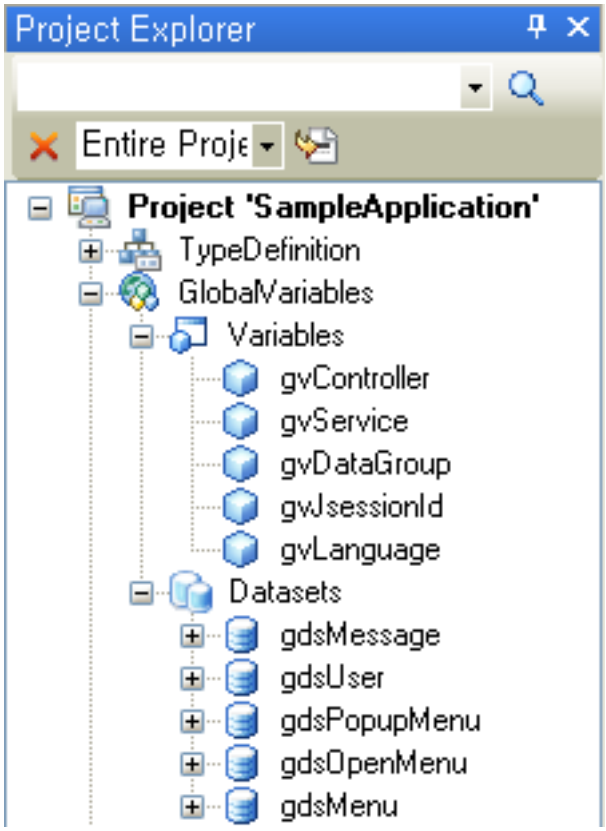
6.3.Naming Rules of Variable

6.3.1.Global Variable

XPLATFORM UX Studio의 Global variables 탭에서 작성하면 어플리케이션 전역에서 사용 가능하다.

- "gv" 로 시작하고 영문 약어 사용.
- 단어와 단어 사이는 단어 첫 알파벳을 대문자로 시작하여 구분

  예) gvLanguage, gvController, ...
- 예외: 'JSESSIONID' - Web Server의 JSESSIONID 값을 화면에서 사용하기 위해 Web Server의 Cookie 변수와 동일한 변수 명으로 정의



6.3.2.Common Script Variable

공통 스크립트(예: lib/commonScript.xjs) 내에 선언되며, 스크립트를 include하는 모든 Form 에서 사용 가능하다.

- Form 스크립트에 include 형태로 포함됨
- "g" 로 시작하고 영문 약어 사용
- 단어와 단어 사이는 단어 첫 알파벳을 대문자로 시작하여 구분

  예) gSysdate, gUserName, ·····
- 함수 안에서 선언되는 변수는 '일반 변수' 규칙을 따름

6.3.3.Local Variable

각 화면 Form 스크립트에서 사용하는 변수는 사용하고자 하는 데이터 타입에 따라 Prefix를 붙이고 UI 상의 의미를 정의한 영문 약어를 사용한다. 단어와 단어 사이는 단어 첫 알파벳을 대문자로 시작하여 구분한다.

표 6.3.

Data Type	Prefix	Example of Variable
문자형	str	strUserNm, strUserId, …
숫자형	n	nTotAmt, nTaxAmt, …
boolean형	b	bIsNull, bHasAuth, …
Array형	arr	arrCategoryIds, arrHolidays, …
DateTime형	dt	dtToday
Object	obj	objDataset, objGrid

6.4.Naming Rules of Function

6.4.1.Global Function

XPLATFORM UX Studio의 ADL 탭에서 작성하는 함수로, include 하지 않아도 어플리케이션 전역에서 사용 가능하다. 함수의 기능을 표현하는 영문 약어로 정의한다.

- "gfn" 으로 시작
- 단어와 단어 사이는 단어 첫 알파벳을 대문자로 시작하여 구분

  예) gfnSetUserInfo(strPara1) { … }, gfnSetAuth(strPara1, strPara2) { … }
- 이벤트 발생 시 작동하는 공통 함수명은 "gfn" + UI컴포넌트명 + "_" + 이벤트명

  예) gfnForm_OnUnLoadCompleted (){ … }, gfnForm_OnLoadCompleted{ … }

6.4.2.Common Script Function

공통 스크립트(예: lib/commonScript.xjs) 내에 선언되며, 스크립트를 include하는 모든 Form 에서 사용 가능하다. 함수의 기능을 표현하는 영문 약어로 정의한다.

- "gfn" 으로 시작
- 단어와 단어 사이는 단어 첫 알파벳을 대문자로 시작하여 구분

  예) gfnIsNull(strArg) { ... }, gfnIsNotNull(strArg) { ... }
- 이벤트 발생 시 작동하는 공통 함수명은 “gfn” + UI컴포넌트명 + "_" + 이벤트명

  예) gfnGrid_OnHeadClick(){…}, gfnForm_OnLoadCompleted{…}

6.4.3.Local Function

일반적으로 개발자들이 화면 개발 시에 Form 스크립트 안에 선언하는 함수로 UI 컴포넌트에서 이벤트 발생 시 작동하는 함수 명은 Design 탭에서 UI 컴포넌트를 더블 클릭했을 때 자동으로 생성되는 이름을 사용한다.

- UI 컴포넌트 ID + "_" + 이벤트명

  예) btSaveAll_onclick() { … }, divSearch_edtSearchKeyword_onkeydown() { … }

  그 외 개발자의 필요에 의해 직접 정의하는 경우 함수의 기능을 표현하는 영문약어를 사용한다.

- "fn"으로 시작, 단어와 단어 사이는 단어 첫 알파벳을 대문자로 시작하여 구분

  예) fnCallback() { … }, fnGetUserInfo() { … }

7.Working with Common Flow

본 절에서는 앞서 설명한 Anyframe XP Query를 이용하여 화면 개발 시 필요한 설정들에 대해 설명 한다.

7.1.Common Script

모든 화면의 Form 스크립트에는 반드시 최상위에 다음과 같은 공통 스크립트 추가 구문을 작성 해야 한다. **include "lib::commonScript.xjs";** commonScript.xjs 파일에는 Anyframe XP Query Plugin에서 공통적으로 사용되는 스크립트 함수들이 정의되어 있고, 또한 그 외 제공되는 스크립트 파일들(*.xjs)이 추가 되어 있다. commonScript.xjs 파일에 정의된 함수 중 Request를 발생시켜 서버를 호출하는 함수는 다음과 같이 정의 되어 있다.

7.1.1.Service Call

gfnService() 함수는 서버측에 사용자가 원하는 Service bean을 호출하기 위한 함수로 내부적으로 XPLATFORM에서 자체적으로 제공하는 transaction() 함수를 호출하도록 작성되어 있다.

7.1.1.1.Syntax

gfnService(strServiceId, strArgument)

예) gfnService("getListCommunity"), gfnService("saveAllBoard")

7.1.1.2.Parameters

Parameter	Description
strServiceId	사용자가 임의로 정하는 Service에 대한 고유한 ID 접두어는 아래 정의된 내용으로 한정함. • **getList: 조회** • **getPagingList: 페이지조회** • **get: 단건 조회** • **create: 등록** • **update: 수정** • **remove: 삭제** • **saveAll: 등록/수정/삭제 모두 수행 (하나의 Transaction)** • **execute: DBMS의 Stored Procedure 실행** 예) getListCommunity
strArgument	Service 호출 시 HTTP GET 방식으로 전달할 파라미터 Syntax: “name1=value1 name2=value2”

7.1.2.Callback

Anyframe XP Query Plugin에서 제공되는 gfnService() 함수를 이용해서 서버에 리퀘스트를 보낸 경우, 요청에 대한 응답이 도착했을 때 **gfnCallback()** 함수를 디폴트로 호출한다. gfnCallback() 함수는 내부적으로 화면 스크립트 내의 **fnCallback()** 함수를 호출하도록 정의되어 있다. 따라서 Service 호출이 완료된 후에 메세지 처리나 컴포넌트 reload 등의 기능을 수행하기 위해서 화면에 정의된 스크립트 내에 fnCallback() 함수를 적절하게 정의 하면 된다.

7.1.2.1.Syntax

fnCallback(strServiceId, strErrorCode, strErrorMsg)

예)

```
// gfnService 처리 후 callback 처리
function fnCallback(strServiceId, strErrorCode, strErrorMsg){
 if(strErrorCode == -1){
  gfnAlertMsg(strErrorMsg, "ERR");
 } else {
  if(strServiceId == "saveAllCategory"){
   gfnAlertMsg("msg.save.success");
   SAMPLE001_onload();
  } else if(strServiceId == "getPagingListCategory"){
   dsSearch.setColumn("SEARCH_CONDITION", searchCondition);
   divPage.objListDataset = dsGrdCategory;
   divPage.objPageDataset = dsSearch;
   divPage.fnMakePage();
  }
 }
}
```

7.1.2.2.Parameters

Parameter	Description
strServiceId	gfnService()로 Service 호출 시 입력했던 strServiceId와 동일한 값
strErrorCode	에러 코드, '-1' 인 경우 Error
strErrorMsg	호출한 Service에서 넘겨준 에러 메세지

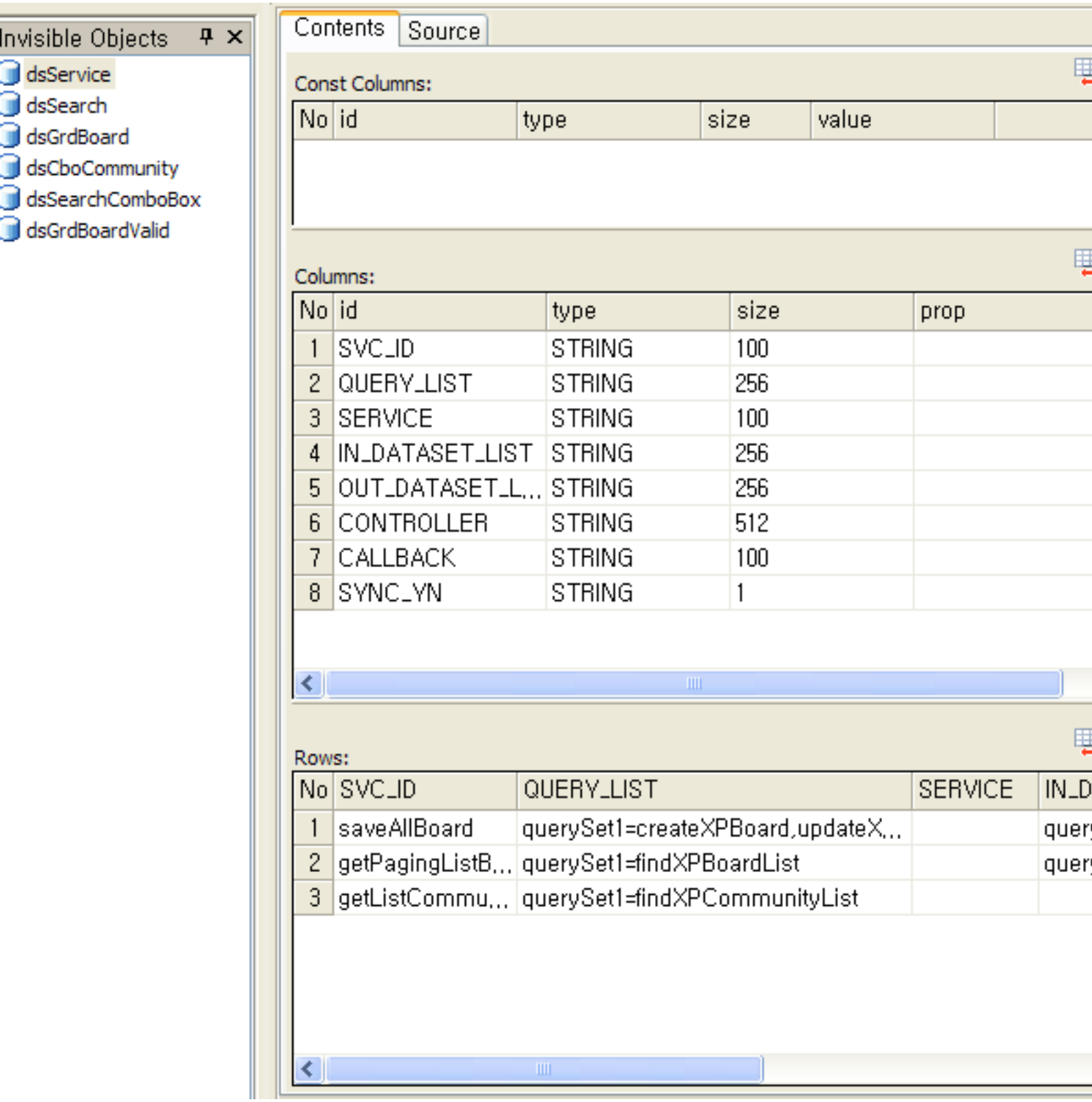
7.2.Common DataSet

7.2.1.DataSet For Service

데이터를 가져오기 위해서 gfnService() 함수를 통해 Service를 호출 할 경우, 호출할 Seervice에 대한 여라가지 파라미터 정보를 DataSet에 정의해야 한다. 따라서 모든 화면에 다음과 같은 값들이 정의된 **dsService** 라는 이름의 DataSet이 반드시 포함 되어야 한다.

Column	Description
SVC_ID	gfnService()의 strServiceId와 동일한 값을 지님 예) getListCommunity

Column	Description
QUERY_LIST	Service에서 실행해야 할 쿼리ID (space로 구분하여 다중입력 가능) Syntax: "querySet" + 순번 + "=" + 쿼리ID 예) querySet1=getListMethodCode querySet1=getListMethodCode querySet2=getListCategory SVC_ID가 'saveAll'로 시작하는 경우(Grid 일괄 저장 시)는 comma(,)를 사용하여 **create, update, remove 쿼리를 한번에 지정** 예) querySet1=createCategory, updateCategory, removeCategory
SERVICE	공통Service를 사용하지 않을 경우 호출할 Service 정보 미지정시, 디폴트로 'gvService'에 설정된 공통Service인 'xpService' 가 호출됨. Syntax: Service명 + "." + Method명 예) categoryMgmtService.getListCategory, categoryMgmtService.createCategory
IN_DATASET_LIST	쿼리 실행 시, 파라미터로 사용될 Dataset ID (space로 구분하여 다중입력 가능) Syntax: "querySet"+ 순번 + "=" + DatasetID 예) querySet1=dsSearch querySet1=dsSearch querySet2=dsParam
OUT_DATASET_LIST	쿼리 수행 결과로 받을 Dataset 이름 목록 (다중입력 가능 - space로 구분) Syntax: DatasetID + "=" + "querySet" + 순번 예) dsSearch=querySet1 dsSearch=querySet1 dsParam=querySet2 Procedure를 호출한 경우 다음 명명 규칙을 따름 "querySet" + 순번 + "_" + Out Parameter Name 예) Mapping Sql의 Out Parameter의 이름 : outVal1 OUT_DATASET_LIST : resultDs1=querySet1_outVal1
CALLBACK	Service로부터 응답을 받았을 때 실행될 Callback 함수 이름 gfnCallback() 내에서 호출. 미지정시, 디폴트로 'fnCallback' 호출.
SYNC_YN	Y: Sync 호출 N: Async 호출 (default) – 권장
CONTROLLER	공통Controller를 사용하지 않을 경우 호출할 Controller 정보 미지정시, 디폴트로 'gvController'에 설정된 공통Controller인 'xpController.do' 가 호출됨. 공통Controller를 사용하지 않는 경우, 호출 할 Service에 대한 정보는 신규 작성한 Controller 에서 포함하고 있으므로 SERVICE 속성을 따로 정의하지 않아도 됨.

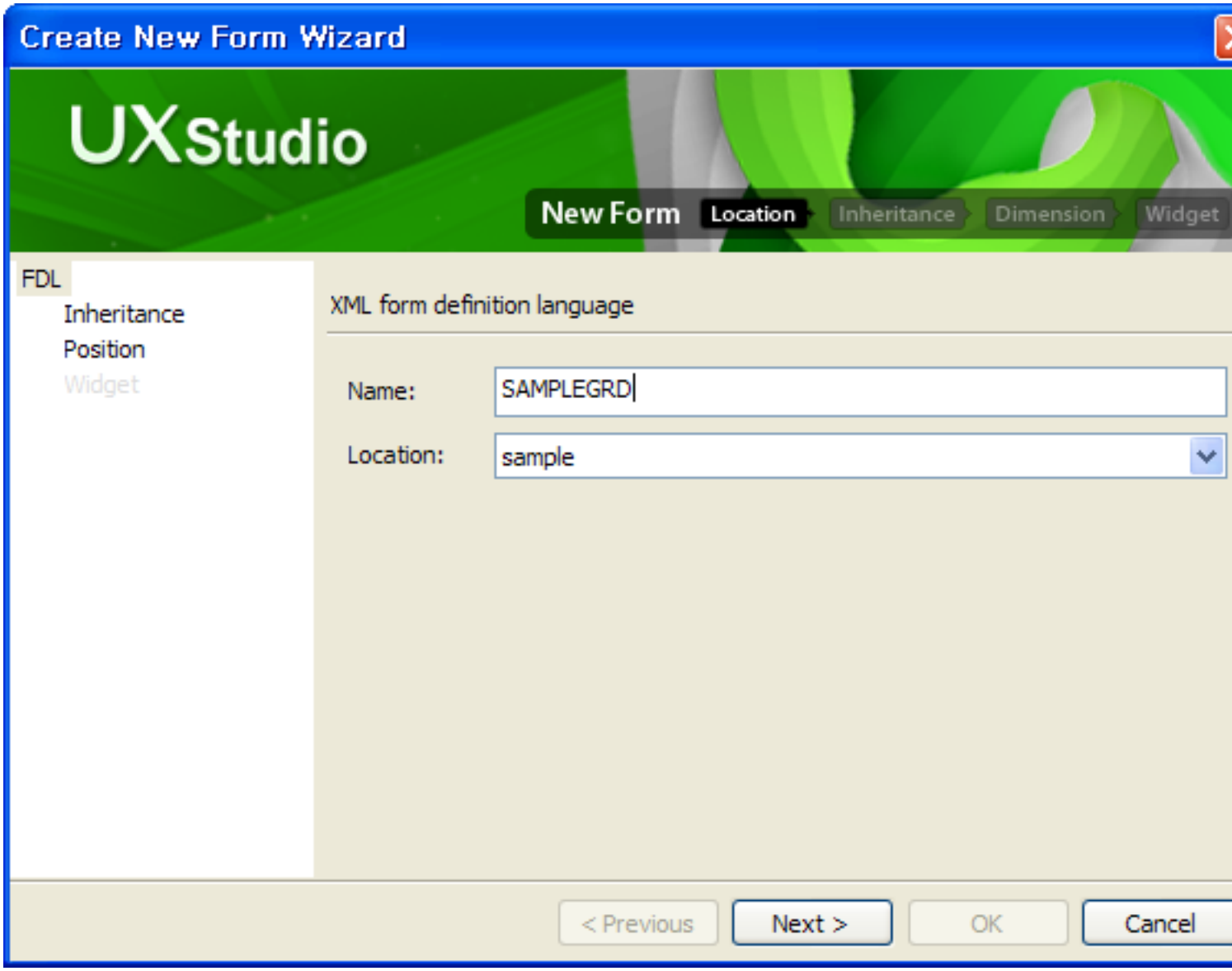


7.3.Example

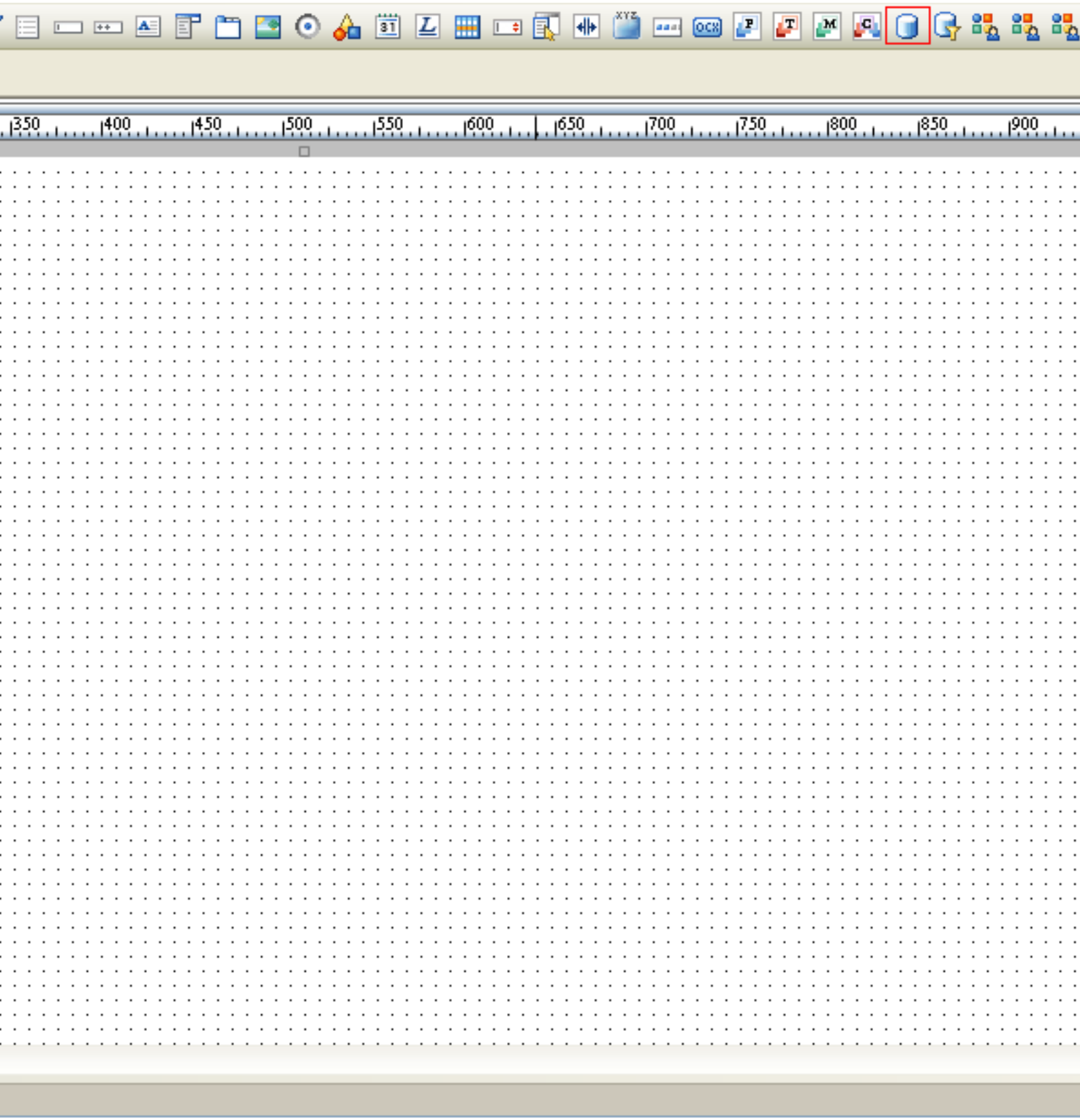
본 절에서는 앞서 설명한 내용들을 바탕으로 공통 Flow를 통해 커뮤니티 목록을 조회해서 Grid로 출력하는 단순 조회 화면 생성 예제를 설명한다.

- XPLATFORM UXStudio의 Project Explorer에서 File - New - Item - Form을 클릭한다.

- Create New Form Wizard 창에서 적절한 Form name을 지정하고 Next 버튼을 클릭한다.



- 이후의 Form size 설정 등을 정의하고 Finish 버튼을 클릭하면 새로운 Form이 생성된다.



- 새로 만든 Form 편집 창에서 상단의 DataSet 아이콘을 클릭한 후 하단의 Invisible Objects 영역을 다시 클릭 하는 것으로 새로운 DataSet을 생성한다. 그 후 DataSet 이름을 dsService로 수정하고, 아래에 나열된 Column을 입력한다. (혹은 다른 샘플 화면에서 dsService DataSet을 복사하여 붙여넣기 해도 무방하다.) 마찬가지 방법으로 'dsGrdCommunity' DataSet을 생성한다.

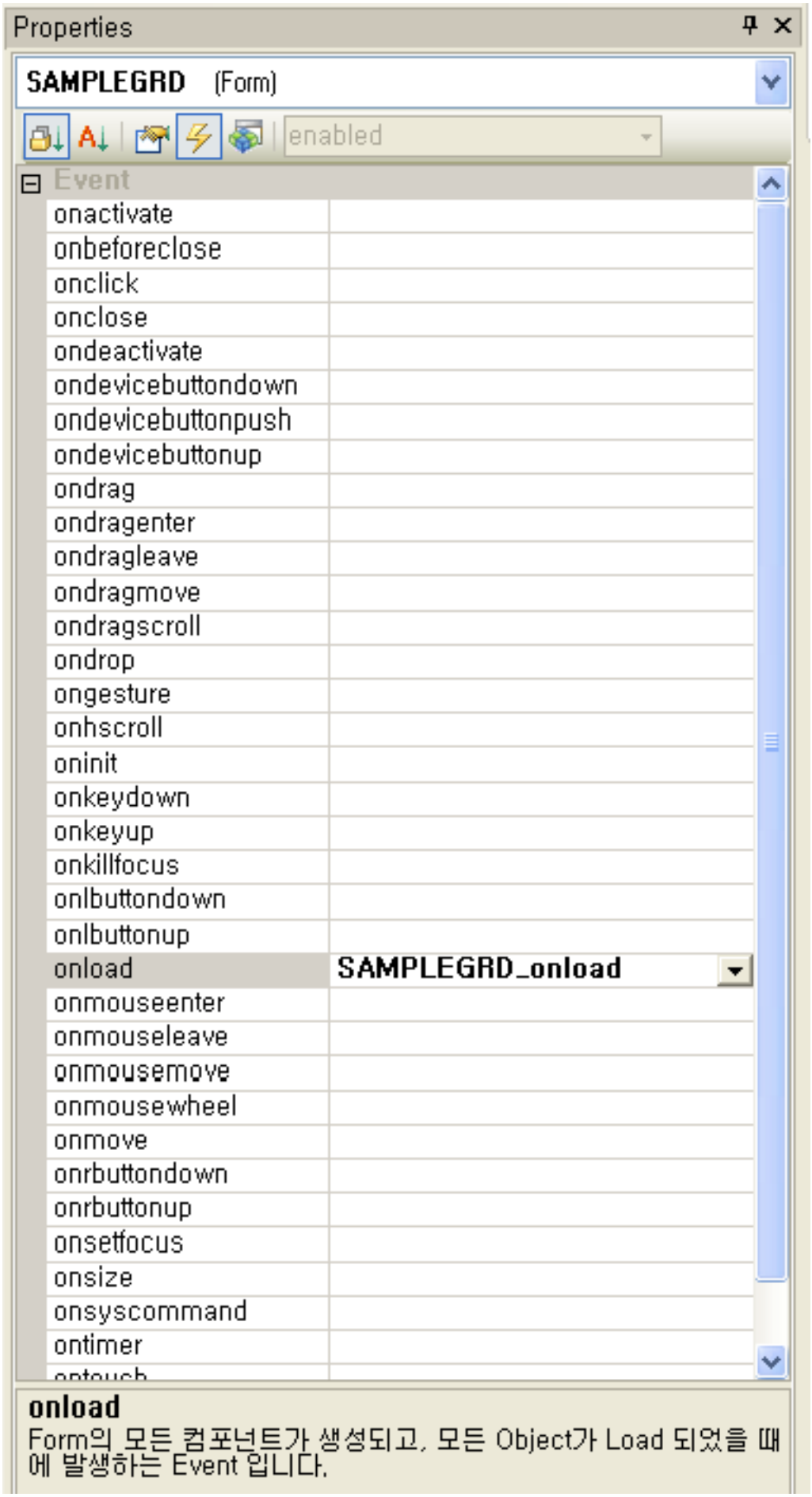
  dsService DataSet은 아래와 같은 컬럼 데이터를 가지도록 설정한다.

Column	Description
SVC_ID	getListCommunity
QUERY_LIST	querySet1=findXPCommunityList
SERVICE	
IN_DATASET_LIST	
OUT_DATASET_LIST	dsGrdCommunity=querySet1
CALLBACK	
SYNC_YN	
CONTROLLER	

  dsGrdCommunity DataSet Column 정보는 아래와 같다.

Column	Type	Size
COMMUNITY_ID	STRING	16
COMMUNITY_NAME	STRING	256
COMMUNITY_DESC	STRING	256
CATEGORY_ID	STRING	16
REG_ID	STRING	256
REG_DATE	STRING	10

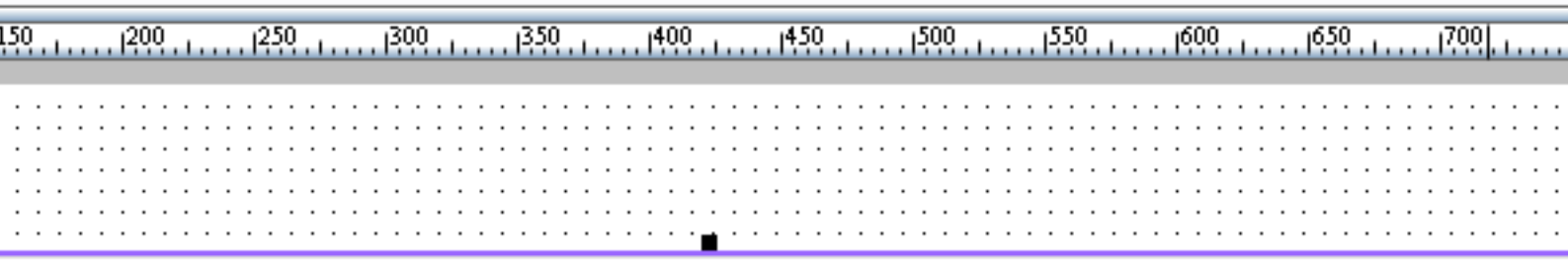
- Form 화면을 클릭 한 후, 오른쪽의 Properties 탭에서 이벤트 보기 아이콘을 클릭하고 Form의 onload 이벤트의 CheckBox 부분을 더블클릭 하여 새로운 이벤트 핸들러를 등록한다.


Properties
SAMPLEGRD (Form)
enabled
Event
onactivate
onbeforeclose
onclick
onclose
ondeactivate
ondevicebuttondown
ondevicebuttonpush
ondevicebuttonup
ondrag
ondragenter
ondragleave
ondragmove
ondragscroll
ondrop
ongesture
onhscroll
oninit
onkeydown
onkeyup
onkillfocus
onlbuttondown
onlbuttonup
onload
SAMPLEGRD_onload
onmouseenter
onmouseleave
onmousemove
onmousewheel
onmove
onrbuttondown
onrbuttonup
onsetfocus
onsize
onsyscommand
ontimer
onload
Form의 모든 컴포넌트가 생성되고, 모든 Object가 Load 되었을 때에 발생하는 Event 입니다.

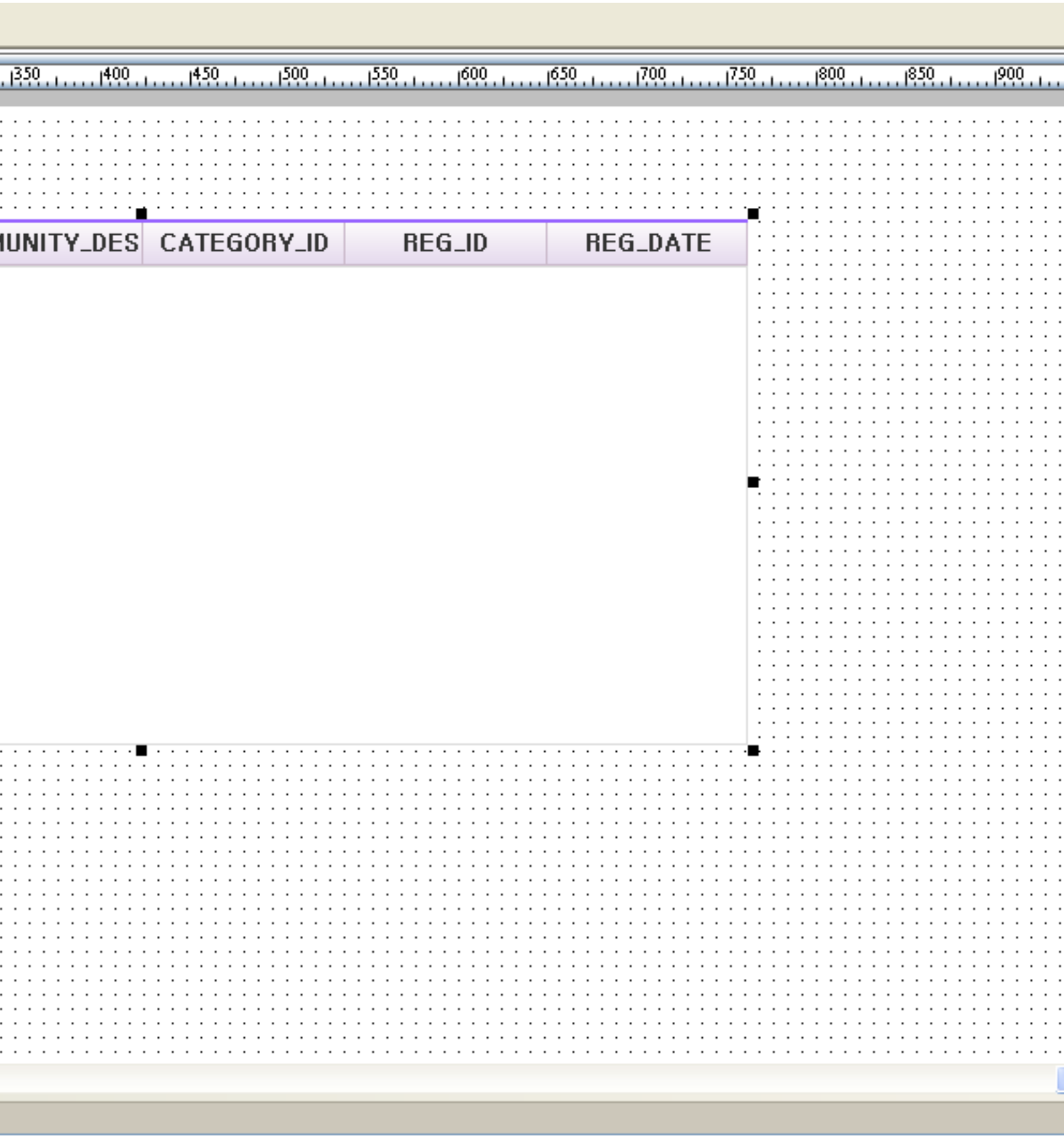
- Script 편집 창으로 이동하여 자동 생성된 onload 이벤트 처리 함수를 완성 한다. 이때 Script 최상단에 commonScript.xjs 파일을 추가하는 구문을 반드시 작성 해야 한다. 아래의 예제 코드를 참고하여 스크립트를 완성한다.

```
include "lib::commonScript.xjs";
function SAMPLEGRD_onload(obj:Form, e:LoadEventInfo)
{
     gfnService("getListCommunity");
}
```

- Design 탭을 클릭하여 Form 편집 창을 열고, 메뉴바 하단에 Grid 아이콘을 클릭하여 적당한 크기의 Grid Component를 그린다.


150
200
250
300
350
400
450
500
550
600
650
700

- 하단의 Invisible Objects 창에 앞서 만든 dsGrdCommunity DataSet을 드래그해서 Grid Component 위에 드롭 한다. 그 후 Grid Component가 선택 된 상태에서 오른쪽의 Properties 창에서 autofittype 속성을 col 로 변경한다.


350
400
450
500
550
600
650
700
750
800
850
900
UNITY_DES
CATEGORY_ID
REG_ID
REG_DATE

• Tool Bar에서 Quick View 아이콘을 클릭하여 조회 화면이 출력 되는것을 확인 한다.

D	OMMUNITY_NAM	OMMUNITY_DES	CATEGORY_ID	REG_ID	REG_DATE
9	연원조기축구	연원조기축구	CATEGORY-0003	test	2009-09-26
0	엑스포 자원봉사	엑스포 자원봉사	CATEGORY-0006	test	2009-09-26
09	레몬테라스	오는 집처럼 예쁘게	CATEGORY-0005	test	2009-09-26
7	복지관 청소	복지관 청소	CATEGORY-0006	test	2009-09-26
03	온갖 놀이	는 모든 놀이를 다 해	CATEGORY-1009	test	2009-09-25
20	목공 나들이	손으로 만들어 쓸 수	CATEGORY-0005	test	2009-08-21
9	컴퓨터 도우미	터 공부를 도와드립L	CATEGORY-0006	test	2009-08-21
5	농활	농촌 봉사활동	CATEGORY-0006	test	2009-08-21
8	동기모임	동기야 반갑다.	CATEGORY-0004	hong80	2009-08-06
7	K리그 봐요	I 있는 분들이면 누구	CATEGORY-0003	test	2009-08-06
4	탄천 환경 정화	탄천을 깨끗하게	CATEGORY-0006	test	2009-08-06

8.Validation

XPLATFORM을 이용하여 화면을 개발할 때 사용자 입력 정보에 대한 Validation 처리는 어떻게 할 수 있는지 알아본다. 기본적으로 XPLATFORM Component에서 제공하는 속성을 이용하여 유효값 검증이 가능하지만, 본 샘플에서는 Script를 이용한 유효값 검증에 대해 설명 한다.

8.1.Using Script

본 샘플에서는 화면에서 사용자가 입력한 정보에 대한 Validation 처리를 일괄적으로 할 수 있도록 공통 Script 함수(gfnDsCheckValid())를 제공한다.

8.1.1.Check Validity

gfnDsCheckValid() 는 입력 파라미터로 지정된 DataSet에 대해서 DataSet에 속한 모든 Row에 대해 유효값 검증 작업을 수행한다. 체크 기준은 "입력 파라미터로 지정된 DataSet Name" + "Valid" 라는 이름의 DataSet에 정의된 각종 유효값 속성에 따른다.

8.1.1.1.Syntax

gfnDsCheckValid(objDataSet)

예) if(gfnDsCheckValid(dsGrdCategory)) {…}

8.1.1.2.Parameters

Property	Description
objDataSet	Validation을 수행할 값이 포함된 DataSet

8.1.2.Check List for Validation

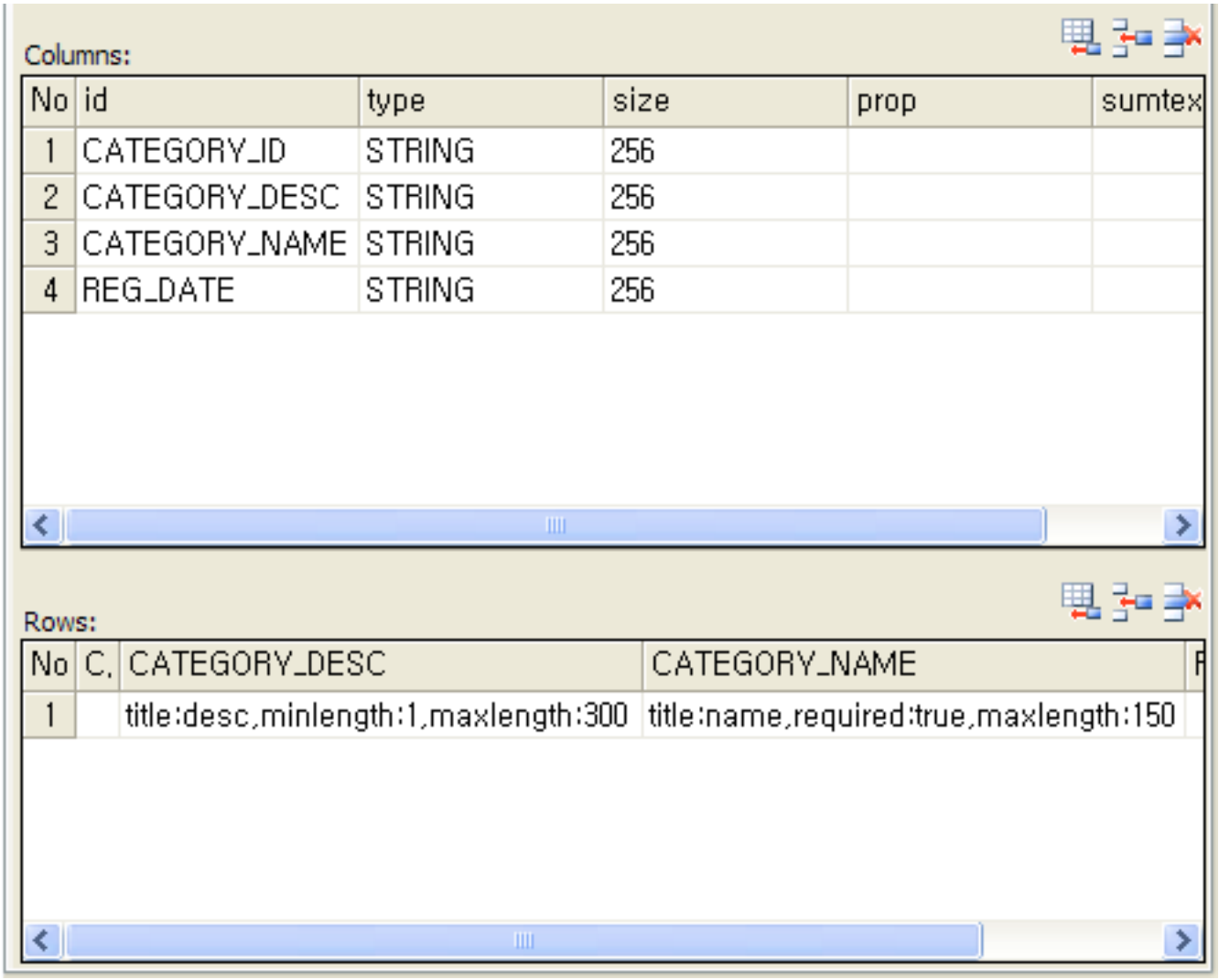
Validation 처리를 위한 Check List 작성 시 각 항목은 comma(,)로 구분한다.

8.1.2.1.Notice

Check Item	Description
title:항목명	오류 메세지 출력 시 사용될 항목 명
required:true,false	컬럼에 대한 필수항목 체크
minLength:값	컬럼에 대한 최소길이 체크 Byte 단위가 아니므로 입력될 문자길이로 지정할 것
maxLength:값	컬럼에 대한 최대 길이 체크 Byte 단위가 아니므로 입력될 문자길이로 지정할 것
num:f,i,n	컬럼 값이 실수, 정수, 자연수 인지 체크
fromNum:값	컬럼에 대한 최소값 체크
toNum:값	컬럼에 대한 최대값 체크
format=mail	컬럼에 대한 email 형식 체크
format:mail	컬럼에 대한 e-mail 형식 체크
format:phone	컬럼에 대한 전화번호 형식 체크

Check Item	Description
format:url	컬럼에 대한 URL 형식 체크
format:date	컬럼에 대한 날짜(YYYYMMDD) 형식 체크
format:date10	컬럼에 대한 날짜(YYYY-MM-DD, YYYY.MM.DD)시간 형식 체크
format:zipcode	컬럼에 대한 우편번호 형식 체크

아래는 Grid의 BindDataset(dsGrdCategory)에서 대한 vaildation check용 DataSet(dsGrdCategoryValid)의 속성 정의 예시이다.

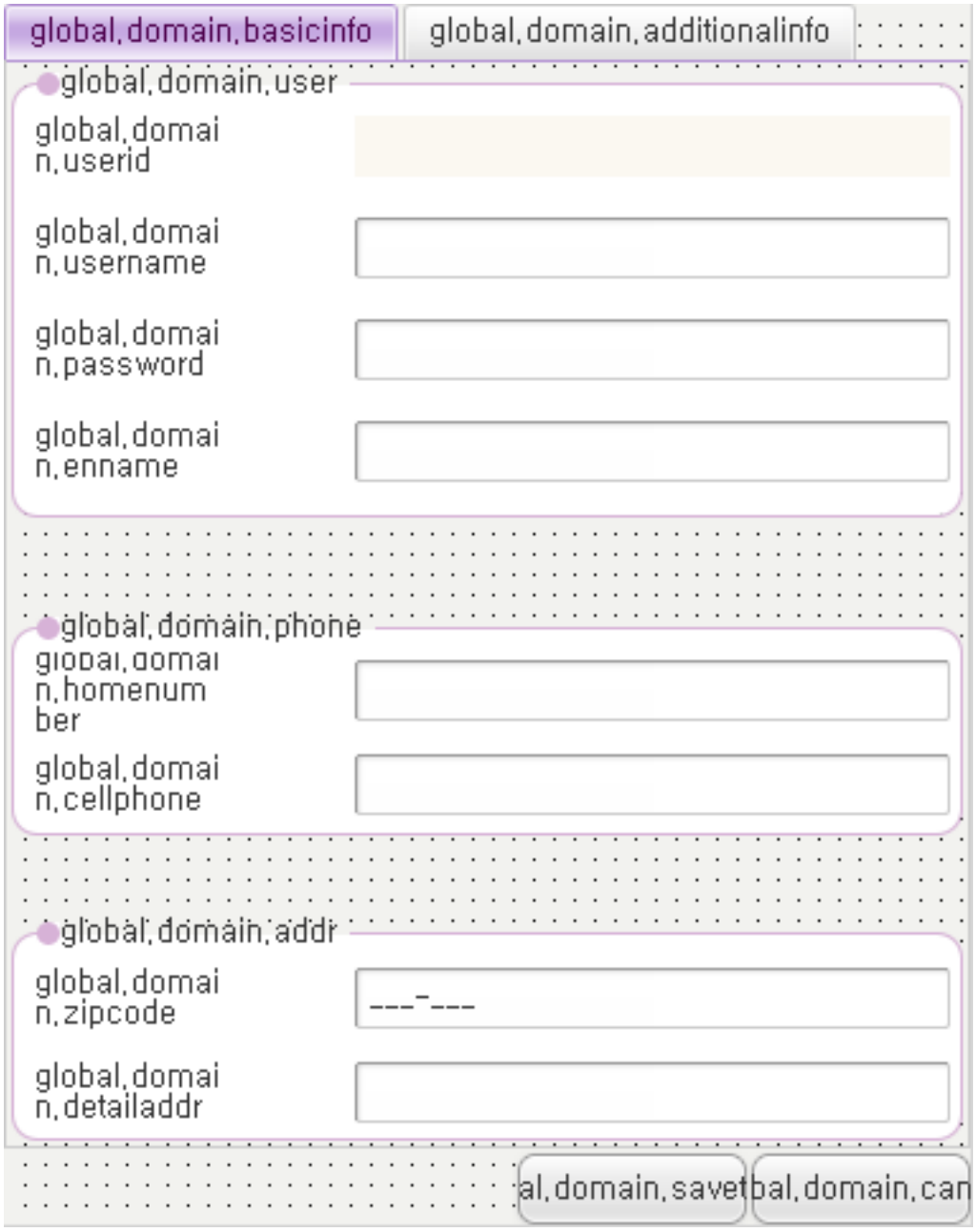


9.Internationalization (i18n)

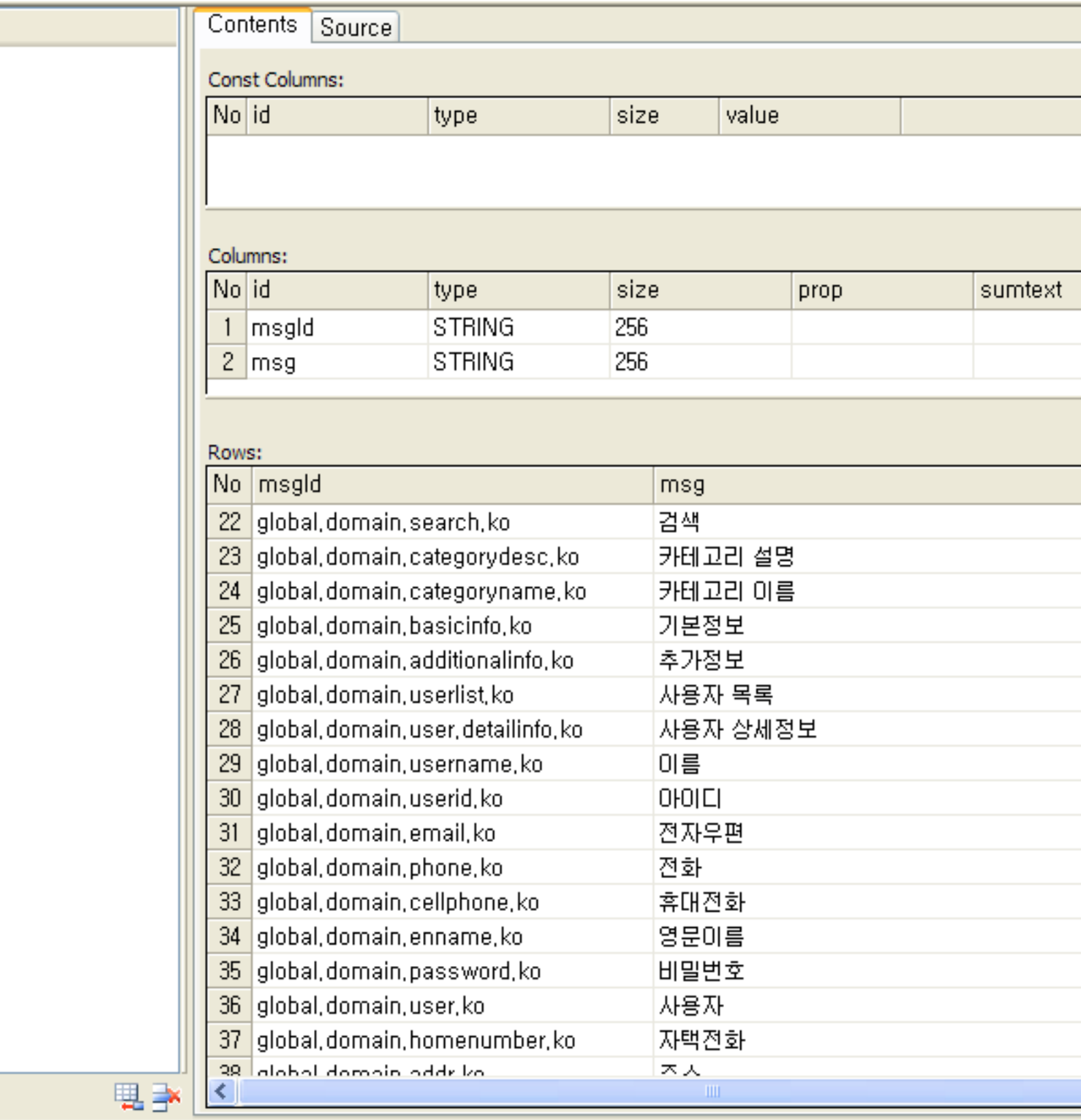
9.1.국제화

본 절에서는 XPLATFORM을 이용하여 어플리케이션 개발 시 국제화 기능 구현 예제를 설명한다.

XPLATFORM 기반의 UI 개발시 국제화 기능을 구현 하기 위한 여러 방법 중 본 샘플에서 제공하는 방법은 global DataSet에 필요한 메세지를 저장해두고 메세지 id값에 따라 해당하는 key값으로 컴포넌트 text를 대체하는 스크립트를 만들어서 사용하는 방법이다. 이 방법 외에 메세지 정보를 서버 properties 파일로 만들거나 DB에 저장하는 방법도 존재하는데, 추후 버전에서 이런 방법들에 대한 가이드를 제공할 예정이다.



위 그림에서 각 컴포넌트들의 Title 속성은 message id로 작성 되어 있다.



Contents Source

Const Columns:

No	id	type	size	value

Columns:

No	id	type	size	prop	sumtext
1	msgId	STRING	256		
2	msg	STRING	256		

Rows:

No	msgId	msg
22	global.domain.search.ko	검색
23	global.domain.categorydesc.ko	카테고리 설명
24	global.domain.categoryname.ko	카테고리 이름
25	global.domain.basicinfo.ko	기본정보
26	global.domain.additionalinfo.ko	추가정보
27	global.domain.userlist.ko	사용자 목록
28	global.domain.user.detailinfo.ko	사용자 상세정보
29	global.domain.username.ko	이름
30	global.domain.userid.ko	아이디
31	global.domain.email.ko	전자우편
32	global.domain.phone.ko	전화
33	global.domain.cellphone.ko	휴대전화
34	global.domain.enname.ko	영문이름
35	global.domain.password.ko	비밀번호
36	global.domain.user.ko	사용자
37	global.domain.homenumber.ko	자택전화

각 msgId에 해당하는 msg key값을 저장하고 있는 변수는 Global DataSet으로 선언된 gdsMessage이다. 위 그림처럼 "msgId" + ".locale" 형태로 msgId를 지정하고, msg값으로 실제 msg값을 저장하면 된다.

각 컴포넌트들의 표현 text를 message 처리 하는 함수는 gfnSetTitle() 함수이다. Form onload 이벤트 처리 함수에 gfnSetTitle() 메소드를 호출하면 Form에 속해있는 각각의 component들에 접근해서 해당

컴포넌트의 title text에 해당하는 message key값이 있는 경우 이 값으로 대체 해준다. 사용자의 Locale 정보는 로그인 시 layout/Login.xfdl의 스크립트에서 서버에서 전달한 Language 값을 gvLanguage에 저장하도록 구현 되어 있다.