OZ Framework Manual

OZ Java Framework 연동 개요

OZR, OZA 등의 폼 파일 뿐만 아니라 데이터도 오즈 서버에서 가져오는 기존 방식과는 달리 OZ Framework을 연동하면 OZR, OZA 등의 폼 파일은 오즈 서버에서 가져오고 데이터는 오즈 서버가 아닌 일반적인 웹 애플리케이션 서버 또는 사용자가 지정한 특정 서버와 연동하여 데이터를 생성하고 생성된 데이터를 가져옵니다.

데이터를 가져올 때는 오즈 서버와 마찬가지로 데이터 셋 또는 데이터 모듈 단위로 생성하고 가져올 수 있으며, 이 때 데이터를 생성하고 가져오는 단위를 패치 단위(Fetch Unit)라고 합니다.
예를 들어 하나의 ODI 파일에 3개의 데이터셋이 있을 경우 패치 단위를 데이터셋으로 설정(fetchunit=dm_per_dataset)하면 해당 데이터셋의 데이터만 가져오고, 패치 단위를 데이터 모듈로 설정(fetchunit=dm_per_datamodule)하면 ODI 파일에 추가되어있는 모든 데이터셋의 데이터를 가져옵니다.

만일 데이터를 특정 URL로부터 데이터를 받아오는 경우에는 ODI명, 데이터셋명 등의 데이터 생성 정보를 POST 패러미터로 전달하여 URL을 호출합니다.
또한 특정 URL을 이용하여 행 삽입, 삭제, 변경 등의 DataAction을 하는 경우 DataAction 내용을 POST 패러미터로 전달하여 URL을 호출합니다. DataAction이 성공한 경우에는 "ok" 또는 "success"로 결과가 리턴됩니다. 설정 가능한 POST 패러미터는 아래 "POST 패러미터" 부분을 참조하시기 바랍니다.

OZ Framework을 이용할 경우에는 ODI의 DB 관련 정보는 사용되지 않기 때문에 사용자 데이터(UDS) 스토어를 이용하거나 오즈 애플리케이션의 경우 FXDataModule을 이용하여 구현할 수 있습니다.

POST 패러미터

클라이언트로 데이터를 전송하거나 클라이언트로부터 행 삽입, 삭제, 변경 정보를 받아 처리할 경우 POST 패러미터를 이용하거나 POST 패러미터에서 정보를 추출하여 처리합니다. Post 패러미터는 ODI 등과 연동되는 기본 DataModule용 POST 패러미터와 FXDataModule에서만 사용되는 FXDataModule용 POST 패러미터로 나뉩니다.

기본 DataModule용 POST 패러미터

■ 서버에 데이터를 조회할 때 서버로 보내지는 POST 패러미터

패러미터 이름	설명
_OZ_ODIFetchType_	패치 타입 DM_BATCH_FETCH 또는 DM_CONCURRENT_FETCH FetchUnit이 DM_PER_DATASET인 경우에 패치 타입은 항상 DM_CONCURRENT_FETCH임.
_OZ_ODIITEM_	ODI 파일명
_OZ_ODICATEGORY_	ODI 카테고리명
_OZ_DATASET_	데이터셋명 FetchUnit이 DM_PER_DATASET인 경우에만 전송됨.
_OZ_DEBUG_	Debug 모드 설정 여부 (true / false) ※ 참고사항 : XML 형태의 SDM을 생성하기 위해서는 반드시 Debug 모드로 설정해야 합니다.
MyParam	ODI 패러미터 ODI 패러미터는 같은 이름으로 전송됨

■ 서버에 DataAction 내용을 보낼 때 서버로 보내지는 POST 패러미터

패러미터 이름	설명
MyParam	ODI 패러미터 ODI 패러미터는 같은 이름으로 전송됨
_OZ_DAC_CNT	DataAction의 개수 <INDEX>와 연관됨

<INDEX>.DATASET	DataAction의 데이터셋명
<INDEX>.TYPE	DataAction의 CUD 타입 Insert, Delete, RowUpdate 중 하나
<INDEX>.EXT	DataAction의 Extra 값
<INDEX>.SRC_CNT	DataAction의 Source 항목 개수 <INDEX2>와 연관됨
<INDEX>.SF_<INDEX2>	DataAction의 <INDEX2>번째 SourceName
<INDEX>.SV_<INDEX2>	DataAction의 <INDEX2>번째 SourceValue
<INDEX>.TRG_CNT	DataAction의 Target 항목 개수 <INDEX2>와 연관됨
<INDEX>.DF_<INDEX2>	DataAction의 <INDEX2>번째 TargetName
<INDEX>.DV_<INDEX2>	DataAction의 <INDEX2>번째 TargetValue

※ 참고사항 : 3개의 ODI 패러미터가 정의되어 있고, DataAction 3개를 Commit한 경우 전송되는 POST 패러미터의 예

```
paramA=somevalue
paramB=somevalue
paramC=somevalue
_OZ_DAC_CNT=3
0.DATASET=dataset1
0.TYPE=Insert
0.EXT=
0.SRC_CNT=2
0.SF_0=FieldName1
0.SV_0=a
0.SF_1=FieldName2
0.SV_1=b
1.DATASET=dataset1
1.TYPE=Delete
1.EXT=
1.TRG_CNT=1
1.DF_0=FieldName1
1.DV_0=a
2.DATASET=dataset1
2.TYPE=RowUpdate
2.EXT=
2.SRC_CNT=2
2.SF_0=FieldName1
2.SV_0=newvalue
2.SF_1=FieldName2
```

```
2.SV_1=oldvalue
2.TRG_CNT=2
2.DF_0=FieldName1
2.DV_0=newvalue
2.DF_1=FieldName2
2.DV_1=oldvalue
```

FXDataModule용 POST 패러미터와 Custom 태그

■ 서버에 DataAction 내용을 보낼 때 서버로 보내지는 POST 패러미터

패러미터 이름	설명
_OZ_DAC_CNT_	DataAction 개수
<DAC_INDEX>.DATASET	데이터셋 이름
<DAC_INDEX>.TYPE	DataAction 타입 (Insert, RowUpdate, Delete)
<DAC_INDEX>.EXT	DataAction의 ext 값
<DAC_INDEX>.SRC_CNT	DataAction의 Source 필드 개수
<DAC_INDEX>.SF_<SRC_INDEX>	DataAction의 Source 필드 이름
<DAC_INDEX>.SFT_<SRC_INDEX>	DataAction의 Source 필드 타입
<DAC_INDEX>.SV_<SRC_INDEX>	DataAction의 Source 필드 값 (값이 없으면 null로 처리됨)
<DAC_INDEX>.TRG_CNT	DataAction의 Target 필드 개수
<DAC_INDEX>.DF_<TRG_INDEX>	DataAction의 Target 필드 이름
<DAC_INDEX>.DFT_<TRG_INDEX>	DataAction의 Target 필드 타입
<DAC_INDEX>.DV_<TRG_INDEX>	DataAction의 Target 필드 값 (값이 없으면 null로 처리됨)

■ 필드 타입별 상수 - 오즈 자바 서버 연동 시

Type	Object	Const	
FX_DT_BIT	Boolean	-7	"BIT"
FX_DT_TINYINT	Integer	-6	"TINYINT"

FX_DT_SMALLINT	Integer	5	"SMALLINT"
FX_DT_INTEGER	Integer	4	"INTEGER"
FX_DT_BIGINT	Long	-5	"BIGINT"
FX_DT_FLOAT	Double	6	"FLOAT"
FX_DT_REAL	Float	7	"REAL"
FX_DT_DOUBLE	Double	8	"DOUBLE"
FX_DT_NUMERIC	String	2	"NUMERIC"
FX_DT_DECIMAL	String	3	"DECIMAL"
FX_DT_CHAR	String	1	"CHAR"
FX_DT_VARCHAR	String	12	"VARCHAR"
FX_DT_LONGVARCHAR	String	-1	"LONGVARCHAR"
FX_DT_DATE	Date	91	"DATE"
FX_DT_TIME	Time	92	"TIME"
FX_DT_TIMESTAMP	Timestamp	93	"TIMESTAMP"
FX_DT_BINARY	byte[]	-2	"BINARY"
FX_DT_VARBINARY	byte[]	-3	"VARBINARY"
FX_DT_LONGVARBINARY	byte[]	-4	"LONGBINARY"
FX_DT_BLOB	byte[]	2004	"BLOB"
FX_DT_CLOB	byte[]	2005	"CLUB"

※ 참고사항 : BINARY, VARBINARY, LONGVARBINARY, BLOB, CLOB 데이터 타입은 BASE64로 저장합니다.

■ 필드 타입별 상수(enum FX_DataTypes) - 오즈 닷넷 서버 연동 시

Type	Object
BIT	Boolean
TINYINT	Integer
SMALLINT	Integer
INTEGER	Integer
BIGINT	Long
FLOAT	Double
REAL	Float
DOUBLE	Double

NUMERIC	String
DECIMAL	String
CHAR	String
VARCHAR	String
LONGVARCHAR	String
DATE	Date
TIME	Time
TIMESTAMP	Timestamp
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]
BLOB	byte[]
CLOB	byte[]

※ 참고사항 : BINARY, VARBINARY, LONGVARBINARY, BLOB, CLOB 데이터 타입은 BASE64로 저장합니다.

■ Custom 태그

- DataModule (오즈 자바 서버 연동 시)

Definition	데이터 모듈
Child	Parameter, DataSetMeta, DataSet

- DataModule (오즈 닷넷 서버 연동 시)

Definition	데이터 모듈	
Attribute	*runat*	동작되는 서버로 반드시 "server"로 설정해야 함
Child	Parameter, DataSetMeta, DataSet	

- Parameter

Definition	패러미터	
Attribute	*name*	패러미터 이름
	fieldType	패러미터 타입
Text	패러미터 값	

- DataSetMeta

Definition	데이터셋 메타 정보	
Attribute	*name*	데이터 셋 이름
	masterSetName	마스터 셋 이름
Child	DataFieldMeta	

- DataFieldMeta

Definition	데이터 필드 메타 정보	
Attribute	*fieldName*	필드 이름
	fieldType	필드 타입

- DataSet

Definition	데이터셋	
Attribute	*name*	데이터 셋 이름
Child	Record	

- Record

Definition	데이터셋의 레코드
Child	Column, DataSet

- Column

Definition	데이터셋의 필드	
Attribute	*Name*	필드 이름
Text	필드 값	

- Send

Definition	데이터 모듈 전송 태그	
Attribute	*isCompress*	압축 여부(gzip)
	isSDM	데이터 모듈 형태(true : SDM, false : XML)
Child	Error	

- Error

Definition	데이터 모듈 생성 실패 메시지

Text	에러 메시지

※ 주의사항

① 태그 라이브러리는 Servlet 2.3이상 버전부터 지원합니다.

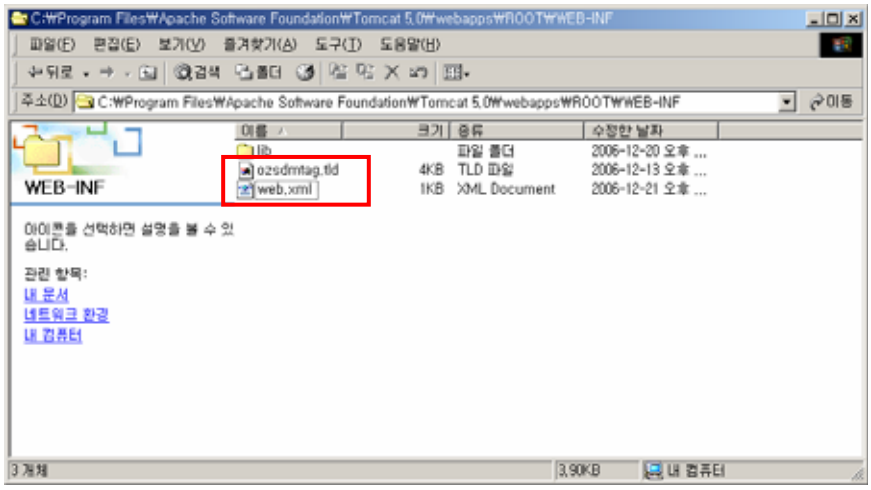
② DataAction은 태그 형태로 지원하지 않습니다.

③ 오즈 닷넷 서버와 연동 시 DataModule 태그에 runat="server"로 반드시 입력되어야 합니다.

■ Custom 태그 사용 방법

- JSP

구동할 WAS의 web.xml 파일에 Custom 태그를 설정합니다. 이때 location으로 지정한 위치에 ozsdmtag.tld 파일이 있어야합니다.



web.xml 파일에 아래와 같은 형식으로 매핑시켜야 합니다.

```
...

<!-- JSPC servlet mappings start -->
...
<taglib>
    <taglib-uri>http://www.forcs.com/oz/fxsdmapi/taglib</taglib-uri>
    <taglib-location>/WEB-INF/ozsdmtag.tld</taglib-location>
  </taglib>
...
<servlet-mapping>
      <servlet-name>sample.Test </servlet-name>
      <url-pattern>/Test </url-pattern>
</servlet-mapping>
```

```
...
<!-- JSPC servlet mappings end -->
```

태그를 사용할 jsp의 HTML 코드 상단에 아래와 같이 입력합니다. URL은 web.xml에 지정한 URL과 동일해야 합니다.

```
<%@ taglib uri="http://www.forcs.com/oz/fxsdmapi/taglib" prefix="oz" %>
```

- ASP.NET

웹 애플리케이션 프로젝트에 OZSDMAPI 어셈블리를 참조 추가한 후 태그를 사용할 aspx의 HTML 코드 상단에 아래와 같이 입력합니다.

```
<%@ Register TagPrefix="oz" Namespace="oz.fxapi.custom.tag"
Assembly="OZSDMAPI" %>
```

사용 예는 아래를 참조하시기 바랍니다.

```
<oz:DataModule runat="server">

<oz:DataSetMeta name="IDs" >
  <oz:DataFieldMeta fieldName="CarID" fieldType="VARCHAR" />
</oz:DataSetMeta>
<oz:DataSetMeta name="Informations" masterSetName="IDs" >
  <oz:DataFieldMeta fieldName="Maker" fieldType="VARCHAR" />
  <oz:DataFieldMeta fieldName="CarName" fieldType="CLOB" />
  <oz:DataFieldMeta fieldName="ECarName" fieldType="VARCHAR" />
</oz:DataSetMeta>
<oz:Parameter name="param1" fieldType="INTEGER">35</oz:Parameter>
<oz:DataSet name="IDs">
  <oz:Record>
    <oz:Column name="CarID" >08</oz:Column>
      <oz:DataSet name="Informations" >
        <oz:Record>
        <oz:Column name="Maker" >HYUNDAI</oz:Column>
        <oz:Column name="CarName" >EF소나타</oz:Column>
        <oz:Column name="ECarName" >EFSONATA</oz:Column>
      </oz:Record>
    </oz:DataSet>
  </oz:Record>
  <oz:Record>
    <oz:Column name="CarID" >05</oz:Column>
    <oz:DataSet name="Informations" >
      <oz:Record>
        <oz:Column name="Maker" >HYUNDAI</oz:Column>
        <oz:Column name="CarName" >다이너스티</oz:Column>
```

```
        <oz:Column name="ECarName" >DYNASTY</oz:Column>
      </oz:Record>
    </oz:DataSet>
  </oz:Record>
  <oz:Record>
    <oz:Column name="CarID" >07</oz:Column>
    <oz:DataSet name="Informations" >
      <oz:Record>
        <oz:Column name="Maker" >HYUNDAI</oz:Column>
        <oz:Column name="CarName" >그랜져</oz:Column>
        <oz:Column name="ECarName" >GRANDEUR</oz:Column>
      </oz:Record>
    </oz:DataSet>
  </oz:Record>
  <oz:Record>
    <oz:Column name="CarID" >04</oz:Column>
    <oz:DataSet name="Informations" >
      <oz:Record>
        <oz:Column name="Maker" >HYUNDAI</oz:Column>
        <oz:Column name="CarName" >에쿠스</oz:Column>
        <oz:Column name="ECarName" >EQUUS</oz:Column>
      </oz:Record>
    </oz:DataSet>
  </oz:Record>
  <oz:Record>
    <oz:Column name="CarID" >09</oz:Column>
    <oz:DataSet name="Informations" >
      <oz:Record>
        <oz:Column name="Maker" >HYUNDAI</oz:Column>
        <oz:Column name="CarName" >베르나</oz:Column>
        <oz:Column name="ECarName" >VERNA</oz:Column>
      </oz:Record>
    </oz:DataSet>
  </oz:Record>
</oz:DataSet>

<oz:Send isCompressed="false" isSDM="true">
<oz:Error>Not implemented</oz:Error>
</oz:Send>

</oz:DataModule>
```

Servlet API 함수 (for OZ Java Server)

DataModuleFactory 관련 함수

■ **DataModuleFactory 클래스 함수**

- getDataModule

Prototype	`public static DataModule getDataModule(String FetchType) throws OZSDMException`	
Definition	지정한 패치 타입의 데이터 모듈을 가져옵니다. 예를들어 패치 타입을 "DM_CONCURRENT_FETCH"으로 설정하여 getDataModule 함수를 호출하면 패치 타입이 CONCURRENT 타입인 데이터 모듈을 가져옵니다.	
Argument	*FetchType*	패치 타입 DM_CONCURRENT_FETCH / DM_BATCH_FETCH / DM_PER_DATASET

※ 참고사항 : 패치 타입

패치 타입이란, 오즈 서버에 보고서의 데이터 모듈을 요청할 때 오즈 서버가 데이터 모듈을 처리하는 방식을 말합니다. OZ Framework에서는 세가지 타입으로 설정할 수 있습니다.

① DM_CONCURRENT_FETCH : 서버에서 데이터 모듈을 생성함과 동시에 클라이언트로 전송하는 방식입니다.

② DM_BATCH_FETCH : 서버에서 전체 데이터에 대한 데이터 모듈의 생성을 모두 완료한 후 클라이언트로 전송하는 방식입니다.

③ DM_PER_DATASET : 데이터셋 방식입니다. 데이터 모듈을 요청하지 않고, 데이터셋 단위로 요청할 때 사용합니다.

DataModule 관련 함수

■ DataModule 클래스 함수

- init

Prototype	`public void init(OutputStream out) throws IOException`
Definition	데이터 모듈 바인딩을 설정한 Stream으로 초기화합니다.
Argument	*out* Response의 OutputStream

- startBinding

Prototype	`public void startBinding() throws IOException, SQLException`
Definition	데이터 모듈의 바인딩을 시작합니다. 바인딩 시작 전에 반드시 호출해야 합니다.

- endBinding

Prototype	`public void endBinding() throws IOException, SQLException`
Definition	데이터 모듈의 바인딩을 종료합니다. 바인딩이 종료된 후 반드시 호출해야 합니다.

- startSet

Prototype	`public void startSet(String DatasetName) throws IOException`
Definition	해당 데이터셋의 바인딩을 시작합니다. 데이터셋 시작 전에 반드시 호출해야 합니다.
Argument	*DatasetName* 데이터셋 이름

- endSet

Prototype	`public void endSet(String DatasetName) throws IOException`
Definition	해당 데이터셋의 바인딩을 종료합니다.
Argument	*DatasetName* 데이터셋 이름

- addParameter

Prototype	`public void addParameter(String ParamName, int FieldType, Object Value)`

Definition	패러미터 값을 설정합니다. ※ 주의사항 : 패러미터 값은 데이터 모듈을 가져올 때만 설정할 수 있습니다. 즉, getDataModule() 함수에서 패치 타입을 DM_CONCURRENT_FETCH 또는 DM_BATCH_FETCH로 설정한 경우에만 패러미터 값을 설정할 수 있습니다.	
Argument	*ParamName*	패러미터 이름
	FieldType	패러미터 필드 타입
	Value	설정할 패러미터 값

- addSetInfo

Prototype	`public void addSetInfo(String DatasetName, String MasterDatasetName, String[] FieldNames, int[] FieldTypes) throws IllegalArgumentException`	
Definition	데이터 모듈에 해당되는 데이터셋 정보를 설정합니다. 즉, 패치 타입을 DM_CONCURRENT_FETCH 또는 DM_BATCH_FETCH로 설정한 경우에 가져올 데이터 셋 정보를 설정합니다.	
Argument	*DatasetName*	데이터셋 이름
	MasterDatasetName	데이터셋의 마스터 데이터셋 이름 단, 마스터 데이터셋이 없는 경우 ""로 설정
	FieldNames	데이터셋의 필드 이름
	FieldTypes	데이터셋의 필드의 타입

- addSetInfo

Prototype	`public void addSetInfo(String DatasetName, String[] FieldNames, int[] FieldTypes) throws IllegalArgumentException`	
Definition	데이터셋에 해당되는 데이터셋 정보를 설정합니다. 즉, 패치 타입을 DM_PER_DATASET으로 설정한 경우에 가져올 데이터셋 정보를 설정합니다.	
Argument	*DatasetName*	데이터셋 이름
	FieldNames	데이터셋의 필드 이름
	FieldTypes	데이터셋의 필드의 타입

- addRow

Prototype	`public void addRow(String DatasetName, HashMap hMap) throws IOException, SQLException`

Definition	ResultSet의 Row을 설정한 데이터셋에 추가합니다.	
Argument	*DatasetName*	데이터셋 이름

	hMap	HashMap 형태로 지정된 추가할 Row 내용
		예)
		…
		rs = stmt.executeQuery(query);
		while(rs.next()) {
		HashMap hMap = new HashMap();
		hMap.put(_FN[0], rs.getString(_FN[0]));
		hMap.put(_FN[1], rs.getString(_FN[1]));
		hMap.put(_FN[2], rs.getString(_FN[2]));
		hMap.put(_FN[3], rs.getString(_FN[3]));
		hMap.put(_FN[4], rs.getString(_FN[4]));
		hMap.put(_FN[5], rs.getString(_FN[5]));
		module.addRow("SET_1", hMap);
		}
		…

- addRow

Prototype	`public void addRow(String DatasetName, List list) throws IOException, SQLException`	
Definition	ResultSet의 Row을 설정한 데이터셋에 추가합니다.	
Argument	*DatasetName*	데이터셋 이름
	list	ArrayList 형태로 지정된 추가할 Row 내용

- addRow

Prototype	`public void addRow(String DatasetName, String[] arr) throws IOException, SQLException`	
Definition	ResultSet의 Row을 설정한 데이터셋에 추가합니다.	
Argument	*DatasetName*	데이터셋 이름
	arr	String[] 형태로 지정된 추가할 Row 내용

- makeSDM_SET

Prototype	`public void makeSDM_SET(String DatasetName, ResultSet rs, OutputStream out) throws OZSDMException, IOException`	
Definition	DM_PER_DATASET 단위로 바인딩하여 SDM을 만듭니다. 단, 패치 타입이 Batch 방식인 경우에는 지원하지 않습니다.	
Argument	*DatasetName*	데이터셋 이름
	rs	ResultSet
	out	Response의 OutputStream

- sendBindErrorMessage

Prototype	`public void sendBindErrorMessage(String msg) throws IOException`	
Definition	바인딩시에 에러 발생 시 클라이언트로 에러 메시지를 전송합니다. ※ 주의사항 : 바인딩 시 에러가 발생하면 반드시 메세지를 처리해줘야 합니다.	
Argument	*msg*	에러 메시지

- sendErrorMessage

Prototype	`public void sendErrorMessage(String msg, OutputStream out) throws IOException`	
Definition	바인딩 외의 에러 발생 시 클라이언트로 에러 메시지를 전송합니다. ※ 주의사항 : 바인딩 외의 에러가 발생하면 반드시 메세지를 처리해줘야 합니다.	
Argument	*msg*	에러 메시지
	out	Response의 OutputStream

- setDebug

Prototype	`public void setDebug(boolean isDebug, HttpServletResponse response) throws IOException`	
Definition	바인딩되는 요청을 Debug 모드로 설정합니다. ※ 참고사항 : XML 형태의 SDM을 생성하기 위해서는 반드시 Debug 모드로 설정해야 합니다. ※ 주의사항 : 이 함수는 init 함수 이전에 호출하여야하며, 한 번만 호출할 수 있습니다. 만약 이 이외의 상황에서 호출 시 "setDebug method can't call after init method calling and call only one time" 이라는 에러 메시지가 발생합니다.	

Argument	*isDebug*	Debug 모드 설정 여부 - true : Debug 모드로 설정함 - false : Debug 모드로 설정하지 않음(기본값)
	response	사용자 응답 객체

■ **Servlet API 함수 호출 순서**

- 데이터 모듈을 가져오는 경우

(1) 데이터 모듈에 있는 데이터셋의 필드 이름과 필드 타입 정의
(2) 데이터 모듈을 가져옴 → ***getDataModule();***
(3) 데이터 모듈 초기화 → ***init();***
(4) ODI 패러미터가 있을 경우 패러미터 설정 → ***addParameter();***
(5) 데이터셋 정보 설정 → ***addSetInfo();***
(6) 데이터 모듈 바인딩 시작 → ***startBinding();***
(7) 데이터셋 바인딩 시작 → ***startSet();***
(8) 데이터셋의 바인딩 로직 구현
(9)* 데이터를 레코드 단위로 추가 → ***addRow();***
(10) 데이터셋 바인딩 종료 → ***endSet();***
(11) 데이터 모듈 바인딩 종료 → ***endBinding();***

※ 참고사항 : (9)번 과정은 해당 데이터셋의 데이터 개수만큼 반복하여 실행합니다.

- 마스터-디테일 구조의 데이터 모듈을 가져오는 경우

(1) 데이터 모듈에 있는 데이터셋의 필드 이름과 필드 타입 정의
(2) 데이터 모듈을 가져옴 → ***getDataModule();***
(3) 데이터 모듈 초기화 → ***init();***
(4) ODI 패러미터가 있을 경우 패러미터 설정 → ***addParameter();***
(5) 데이터셋 정보 설정 → ***addSetInfo();***
(6) 데이터 모듈 바인딩 시작 → ***startBinding();***
(7) 마스터 데이터셋 바인딩 시작 → ***startSet();***
(8) 마스터 데이터셋의 바인딩 로직 구현

(9)* 마스터 데이터셋의 데이터를 레코드 단위로 추가 → ***addRow();***

(10)* 디테일 데이터셋 바인딩 시작 → ***startSet();***

(11)* 디테일 데이터셋의 바인딩 로직 구현

(12)* 디테일 데이터셋의 데이터를 레코드 단위로 추가 → ***addRow();***

(13)* 디테일 데이터셋 바인딩 종료 → ***endSet();***

(14) 마스터 데이터셋 바인딩 종료 → ***endSet();***

(15) 데이터 모듈 바인딩 종료 → ***endBinding();***

※ 참고사항 : (9)번부터 (13)번 과정은 마스터 데이터셋의 데이터 개수만큼 반복하여 실행하고, (12)번 과정은 디테일 데이터셋의 데이터 개수만큼 반복하여 실행합니다.

- 데이터셋만 가져오는 경우

(1) 데이터 모듈에 있는 데이터셋의 필드 이름과 필드 타입 정의

(2) 데이터 모듈을 가져옴 → ***getDataModule();***

(3) 데이터 모듈 초기화 → ***init();***

(4) 데이터셋 정보 설정 → ***addSetInfo();***

(5) 데이터 모듈 바인딩 시작 → ***startBinding();***

(6) 데이터셋 바인딩 시작 → ***startSet();***

(7) 데이터셋의 바인딩 로직 구현

(8)* 데이터를 레코드 단위로 추가 → ***addRow();***

(9) 데이터셋 바인딩 종료 → ***endSet();***

(10) 데이터 모듈 바인딩 종료 → ***endBinding();***

※ 참고사항 : (8)번 과정은 데이터셋의 데이터 개수만큼 반복하여 실행합니다.

FXDataModule 관련 함수

■ FX_DataModule 클래스 함수

- FX_DataModule

Prototype	`public FX_DataModule()`
Argument	FXDataModule을 생성합니다.

- addParameter

Prototype	`public void addParameter(FX_Parameter param)`	
Definition	패러미터 값을 설정합니다.	
Argument	*Param*	패러미터 이름

- addParameters

Prototype	`public void addParameters(FX_Parameter[] param)`	
Definition	패러미터 값을 설정합니다.	
Argument	*Params*	패러미터 이름

- addDataSetMeta

Prototype	`public void addDataSetMeta(FX_DataSetMeta meta)`	
Definition	데이터셋 메타 정보를 설정합니다.	
Argument	*Meta*	데이터셋 메타 정보

- write

Prototype	`public void write(OutputStream out, booleam isSDM, boolen isCompress)`	
Definition	설정된 DataModule의 값을 SDM 또는 XML 파일 형태로 변환합니다. ※ 참고사항 : 압축하지 않은 XML 파일로 저장할 경우 Response의 ContentType은 사용자가 지정해야 합니다. (Ex : "text/xml")	
Argument	*out*	파일을 변환할 OutStream
	isSDM	SDM 형태로 저장 여부 (false일 경우 XML로 저장)

	isCompress	압축 여부 ※ 참고사항 : true일 경우 GZip으로 압축해서 저장하고 클라이언트에서 Unzip을 해야합니다.
Return	파일 변환 성공 여부	

- sendErrorMessage

Prototype	`public void sendErrorMessage(OutputStream out, booleam isSDM, boolen isCompress, String error)`	
Definition	설정된 DataModule의 값을 SDM 또는 XML 파일 형태로 변환하고, 에러 발생 시 클라이언트로 에러 메시지를 전송합니다.	
Argument	*out*	파일을 변환할 OutStream
	isSDM	SDM 형태로 저장 여부 (false일 경우 XML로 저장)
	isCompress	압축 여부 ※ 참고사항 : true일 경우 GZip으로 압축해서 저장하고 클라이언트에서 Unzip을 해야합니다.
	error	에러 발생 시 전송할 메시지
Return	파일 변환 성공 여부	

- getDataAction

Prototype	`public FX_DataAction getDataAction(HttpServletRequest request)`	
Definition	Request로부터 패러미터를 파싱하여 FX_DataAction 객체를 리턴합니다.	
Argument	*request*	Request 객체
	error	에러 발생 시 전송할 메시지
Return	DataAction 객체	

■ **FX_DataSetMeta 클래스 함수**

- FX_ DataSetMeta

Prototype	`public FX_DataSetMeta(String name)`	
Definition	FXDataModule의 데이터셋을 생성합니다.	
Argument	*name*	데이터셋 이름

- FX_DataSetMeta

Prototype	public FX_DataSetMeta(String name, String mSetName)	
Definition	FXDataModule의 디테일 데이터셋을 생성합니다.	
Argument	*name*	데이터셋 이름
	mSetName	마스터 데이터셋 이름

- setMasterSetName

Prototype	public void setMasterSetName(String mSetName)	
Definition	해당 데이터셋의 마스터 데이터셋 이름을 설정합니다.	
Argument	*mSetName*	마스터 데이터셋 이름

- addDataFieldMeta

Prototype	public void addDataFieldMeta(FX_DataFieldMeta field)	
Definition	데이터셋에 필드를 추가합니다.	
Argument	*field*	추가할 필드 정보

- addDataFieldMeta

Prototype	public void addDataFieldMeta(FX_DataFieldMeta[] fields)	
Definition	데이터셋에 여러 개의 필드를 추가합니다.	
Argument	*fields*	추가할 필드 정보

■ **FX_DataFieldMeta 클래스 함수**

- FX_DataFieldMeta

Prototype	public FX_DataFieldMeta(String name, int type)	
Definition	데이터셋의 필드를 생성합니다.	
Argument	*name*	필드 이름
	type	필드 타입

- FX_ DataFieldMeta

Prototype	public FX_DataFieldMeta(String name)	
Definition	데이터셋의 필드를 기본형인 VARCHAR 타입으로 생성합니다.	
Argument	*name*	필드 이름

- setType

Prototype	`public void setType(int type)`
Definition	필드의 타입을 설정합니다.
Argument	*type* 필드 타입

■ FX_DataSet 클래스 함수

- FX_DataSet

Prototype	`public FX_DataSet(String name)`
Definition	FXDataModule의 데이터셋을 생성합니다.
Argument	*name* 데이터셋 이름

- addRecord

Prototype	`public void addRecord(FX_Record record)`
Definition	데이터셋에 레코드 즉, 행 단위의 데이터를 추가합니다.
Argument	*record* 행 데이터

■ FX_Record 클래스 함수

- FX_Record

Prototype	`public FX_Record()`
Definition	레코드 즉, 행을 생성합니다.

- addColumn

Prototype	`public void addColumn(String name, Object value)`	
Definition	행에 필드 값을 추가합니다.	
Argument	*name*	필드 이름
	value	필드 값

- addDetailDataSet

Prototype	`public void addDetailDataSet(FX_DataSet detail)`
Definition	디테일 데이터셋의 값을 추가합니다.
Argument	*detail* 디테일 데이터셋의 값

■ FX_DataAction 클래스 함수

- FX_DataAction

Prototype	`public FX_DataAction(String name, String actionType)`	
Definition	DataAcion을 생성합니다.	
Argument	*name*	데이터셋 이름
	actionType	DataAction 타입

- FX_DataAction

Prototype	`public FX_DataAction(String name, String actionType, String ext)`	
Definition	DataAcion을 생성합니다.	
Argument	*name*	데이터셋 이름
	actionType	DataAction 타입
	ext	ext 값

- getDataSetName

Prototype	`public String getDataSetName()`
Definition	데이터셋 이름을 리턴합니다.
Return	데이터셋 이름

- getActionType

Prototype	`public String getActionType()`
Definition	DataAction 타입을 리턴합니다.
Return	DataAction 타입 (Insert, RowUpdate, Delete)

- getExt

Prototype	`public String getExt()`
Definition	ext 값을 리턴합니다.
Return	ext 값

- setExt

Prototype	`public String setExt(String ext)`
Definition	ext 값을 설정합니다.

Argument	*ext*	ext 값

- getSourceFields

Prototype	`public FX_Parameter[] getSourceFields()`
Definition	Source 필드 정보를 리턴합니다.
Return	Source 필드 정보

- setSourceFields

Prototype	`public void setSourceFields(FX_Parameter[] fields)`	
Definition	Source 필드 정보을 설정합니다.	
Argument	*fields*	Source 필드 정보

- getTargetFields

Prototype	`public FX_Parameter[] getTargetFields()`
Definition	Target 필드 정보을 리턴합니다.
Return	Target 필드 정보

- setTargetFields

Prototype	`public void setTargetFields(FX_Parameter[] fields)`	
Definition	Target 필드 정보를 설정합니다.	
Argument	*fields*	Target 필드 정보

■ FX_Parameter 클래스 함수

- FX_Parameter

Prototype	`public FX_Parameter(String name, int type, String value)`	
Definition	FXDataModule의 패러미터를 생성합니다.	
Argument	*name*	패러미터 이름
	type	패러미터 데이터 타입
	value	패러미터 데이터 값

- FX_Parameter

Prototype	`public FX_Parameter(String name, String value)`

Definition	FXDataModule의 패러미터를 기본형인 VARCHAR 타입으로 생성합니다.	
Argument	*name*	패러미터 이름
	value	패러미터 데이터 값

- getName

Prototype	`public String getName()`
Definition	패러미터 이름을 리턴합니다.
Return	패러미터 이름

- getType

Prototype	`public int getType()`
Definition	패러미터 데이터 타입을 리턴합니다.
Return	패러미터 데이터 타입

- setType

Prototype	`public void setType(int type)`	
Definition	패러미터 데이터 타입을 설정합니다.	
Definition	*type*	패러미터 데이터 타입

- getValue

Prototype	`public String getType()`
Definition	패러미터 값을 리턴합니다.
Return	패러미터 값

- isNull

Prototype	`public boolean isNull()`
Definition	값의 null을 허용 여부를 리턴합니다.
Return	null 허용 여부

Servlet API 함수 (for OZ .Net Server)

※ 주의사항 : 오즈 닷넷 서버의 Servlet API 함수는 Framework 1.1.4322 이상에서 사용 가능합니다.

FXDataModule 관련 함수

■ **FX_DataModule 클래스 함수**

- FX_DataModule

Prototype	`public FX_DataModule()`
Argument	FXDataModule을 생성합니다.

- addParameter

Prototype	`public void addParameter(FX_Parameter param)`
Definition	패러미터 값을 설정합니다.
Argument	*Param* 패러미터 이름

- addParameters

Prototype	`public void addParameters(FX_Parameter[] param)`
Definition	패러미터 값을 설정합니다.
Argument	*Params* 패러미터 이름

- addDataSetMeta

Prototype	`Public void addDataSetMeta(FX_DataSetMeta meta)`
Definition	데이터셋 메타 정보를 설정합니다.
Argument	*Meta* 데이터셋 메타 정보

- write

Prototype	`public void write(OutputStream output, booleam isSDM, boolen isCompress)`

Definition	설정된 DataModule의 값을 SDM 또는 XML 형태로 변환합니다. ※ 참고사항 : 압축하지 않은 XML 파일로 저장할 경우 Response의 ContentType은 사용자가 지정해야 합니다. (Ex : "text/xml")	
Argument	*output*	값을 저장할 OutStream
	isSDM	SDM 형태로 저장 여부 (false일 경우 XML로 저장)
	isCompress	값의 압축 여부 ※ 참고사항 : true일 경우 GZip으로 압축해서 저장하고 클라이언트에서 Unzip을 해야합니다.
Return	파일 변환 성공 여부	

- sendErrorMessage

Prototype	`public void sendErrorMessage(OutputStream output, booleam isSDM, boolen isCompress, string error)`	
Definition	설정된 DataModule의 값을 SDM 또는 XML 파일 형태로 변환하고, 에러 발생 시 클라이언트로 에러 메시지를 전송합니다.	
Argument	*output*	파일을 변환할 OutStream
	isSDM	SDM 형태로 저장 여부 (false일 경우 XML로 저장)
	isCompress	값의 압축 여부 ※ 참고사항 : true일 경우 GZip으로 압축해서 저장하고 클라이언트에서 Unzip을 해야합니다.
	error	에러 발생 시 전송할 메시지
Return	파일 변환 성공 여부	

- getDataAction

Prototype	`public FX_DataAction getDataAction(HttpServletRequest request)`	
Definition	Request로부터 패러미터를 파싱하여 FX_DataAction 객체를 리턴합니다.	
Argument	*request*	Request 객체
	error	에러 발생 시 전송할 메시지
Return	DataAction 객체	

■ FX_DataSetMeta 클래스 함수

- FX_ DataSetMeta

Prototype	`public FX_DataSetMeta(String name)`

Definition	FXDataModule의 데이터셋을 생성합니다.	
Argument	*name*	데이터셋 이름

- FX_DataSetMeta

Prototype	`public FX_DataSetMeta(String name, String mSetName)`	
Definition	FXDataModule의 디테일 데이터셋을 생성합니다.	
Argument	*name*	데이터셋 이름
	mSetName	마스터 데이터셋 이름

- MasterSetName

Prototype	`public string MasterSetName{get; set;}`
Definition	해당 데이터셋의 마스터 데이터셋 이름을 설정합니다.

- addDataFieldMeta

Prototype	`public void addDataFieldMeta(FX_DataFieldMeta[] field)`	
Definition	데이터셋에 여러 개의 필드를 추가합니다.	
Argument	*field*	추가할 필드 정보

■ **FX_DataFieldMeta 클래스 함수**

- FX_DataFieldMeta

Prototype	`public FX_DataFieldMeta(string name, int type)`	
Definition	데이터셋의 필드를 생성합니다	
Argument	*name*	필드 이름
	type	필드 타입

- FX_ DataFieldMeta

Prototype	`public FX_DataFieldMeta(String name)`	
Definition	데이터셋의 필드를 기본형인 VARCHAR 타입으로 생성합니다.	
Argument	*name*	필드 이름

- Type

Prototype	`public FX_DataTypes Type{get; set;}`
Definition	필드의 타입을 설정합니다.

■ FX_DataSet 클래스 함수

- FX_DataSet

Prototype	`public FX_DataSet(string name)`
Definition	FXDataModule의 데이터셋을 생성합니다
Argument	*name* 데이터셋 이름

- addRecord

Prototype	`public void AddRecord(FX_Record record)`
Definition	데이터셋에 레코드 즉, 행 단위의 데이터를 추가합니다.
Argument	*record* 행 데이터

■ FX_Record 클래스 함수

- FX_Record

Prototype	`public FX_REcord()`
Definition	레코드 즉, 행을 생성합니다.

- addColumn

Prototype	`public void addColumn(string name, Object value)`	
Definition	행에 필드 값을 추가합니다.	
Argument	*name*	필드 이름
	value	필드 값

- addDetailDataSet

Prototype	`public void addDetailDataSet(FX_DataSet detail)`
Definition	디테일 데이터셋의 값을 추가합니다.
Argument	*detail* 디테일 데이터셋의 값

■ FX_DataAction 클래스 함수

- FX_DataAction

Prototype	`public FX_DataAction(string name, string actionType)`
Definition	DataAcion을 생성합니다.
Argument	*name* 데이터셋 이름

	actionType	DataAction 타입

- FX_DataAction

Prototype	`public FX_DataAction(string name, string actionType, string ext)`	
Definition	DataAcion을 생성합니다.	
Argument	*name*	데이터셋 이름
	actionType	DataAction 타입
	ext	ext 값

- DataSetName

Prototype	`public string DataSetName{get;}`
Definition	데이터셋 이름을 리턴합니다.

- ActionType

Prototype	`public string ActionType{get;}`
Definition	DataAction 타입을 리턴합니다. (Insert, RowUpdate, Delete)

- ExtraArgument

Prototype	`public string ExtraArgument{get; set;}`
Definition	ext 값을 리턴하거나 설정합니다.

- SourceFields

Prototype	`public FX_Parameter[] SourceFields{get; set;}`
Definition	Source 필드 정보를 리턴하거나 설정합니다.

- TargetFields

Prototype	`public FX_Parameter[] TargetFields{get; set;}`
Definition	Target 필드 정보를 리턴하거나 설정합니다.

■ **FX_Parameter 클래스 함수**

- FX_Parameter

Prototype	`public FX_Parameter(string name, int type, string value)`

Definition	FXDataModule의 패러미터를 생성합니다.	
Argument	*name*	패러미터 이름
	type	패러미터 데이터 타입
	value	패러미터 데이터 값

- FX_Parameter

Prototype	`public FX_Parameter(string name, string value)`	
Definition	FXDataModule의 패러미터를 기본형인 VARCHAR 타입으로 생성합니다.	
Argument	*name*	패러미터 이름
	value	패러미터 데이터 값

- Name

Prototype	`public string Name{get;}`
Definition	패러미터 이름을 리턴합니다.

- Type

Prototype	`public FX_DataTypes Type{get; set;}`
Definition	패러미터 데이터 타입을 리턴하거나 설정합니다.

- value

Prototype	`public string value{get;}`
Definition	패러미터 값을 리턴합니다.

- IsNull

Prototype	`public string getType{get; set;}`
Definition	값의 null 허용 여부를 리턴하거나 설정합니다.

Servlet API 적용 예

예제 1 : Application - 데이터 검색

Servlet API를 이용하여 오즈 애플리케이션의 Table을 이용하여 데이터를 표시하는 예제를 살펴보겠습니다.

Servlet API를 이용하여 데이터를 검색하기 위해서는 먼저 ozsdmapi.jar 파일과 데이터를 검색하는 클래스 파일을 WAS에 등록한 후 UDS 스토어를 이용한 ODI 파일을 만들어 오즈 애플리케이션의 프로젝트에 추가하여야 합니다.

Step 1 ozsdmapi.jar 파일을 WAS에 등록하기

Servlet API 연동 모듈인 ozsdmapi.jar 파일을 WAS의 라이브러리에 등록합니다. 본 예제에서는 Tomcat 5.0의 webapps\ROOT\WEB-INF\lib 폴더에 ozsdmapi.jar 파일을 복사하였습니다.

Step 2 데이터 검색용 클래스 파일 만들기

데이터를 검색하는 클래스 파일을 만듭니다. 본 예제에서는 아래와 같은 java 소스 파일을 이용하여 DataModuleSampleServlet.class 파일로 만들었습니다.

```
package sample;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;

import java.util.*;

import oz.framework.api.DataModule;
import oz.uds.rs.ListMapResultSet;
import oz.sdm.DataModuleFactory;
import oz.uds.rs.ListMapResultSet;


/**
 * <p>Title: OZ SDM API</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: </p>
 * @author Forcs
 * @version 1.0
 */

public class DataModuleSampleServlet extends HttpServlet
{
```

```
    private static final String _KEY_ODI_FETCH_TYPE = "_OZ_ODIFetchType_";
    private static final String _KEY_ODIITEM            = "_OZ_ODIITEM_";
    private static final String _KEY_ODICATEGORY       = "_OZ_ODICATEGORY_";
    private static final String _KEY_OZ_DATASET_       = "_OZ_DATASET_";

    private Connection m_conn = null;

    public void init(ServletConfig config)
        throws ServletException
    {
        super.init(config);

        System.out.println("init...");
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        process(request, response);
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        process(request, response);
    }

    private void process(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        // init할때 커낵션을 정보를 설정하여 connection을 가져온다.
        try {
            String _URL = "jdbc:odbc:ozdemokr30";
            Properties prop = new Properties();
            prop.put("user", "");
            prop.put("password", "");

            Driver                    driver                    =                    (Driver)
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
            m_conn = driver.connect(_URL, prop);
            System.out.println("connection ok...");
        }
        catch(Exception e) {
            System.out.println("connection error...");
            DataModule.sendErrorMessage("connection error...",response.getOutputStream());

            if (m_conn != null) {
                try {
                    m_conn.close();
                }
                catch (Exception e1) {
                }
            }

            throw new ServletException(e.getMessage());
        }

        try {
            System.out.println("start----------");
            Enumeration enum = request.getParameterNames();
            while (enum.hasMoreElements()) {
                String temp = (String) enum.nextElement();
                System.out.println("name=" + temp + "     value=" +
                                   getEncode(request.getParameter(temp)));
            }
```

```
                // ------------------------------------------------------------------
                // fetch parameters
                // "DM_BATCH_FETCH|DM_CONCURRENT_FETCH"
                String odiFetchType = getEncode(request.getParameter(_KEY_ODI_FETCH_TYPE));
                String item = getEncode(request.getParameter(_KEY_ODIITEM));
                String category = getEncode(request.getParameter(_KEY_ODICATEGORY));
                String dataset = getEncode(request.getParameter(_KEY_OZ_DATASET_));
                // ------------------------------------------------------------------

                if (odiFetchType == null) odiFetchType = "DM_CONCURRENT_FETCH";

                if ( (dataset == null) || (dataset.length() == 0)) {
                    // ODI FetchUnit이 DM_PER_DATAMODULE 방식이다.
                    if (item.equals("JASMIN_SAMPLE3.odi")) {
                        // Master-Detail간 Sample이다.
                        Master_Detail_Style(request, response, odiFetchType);
                    }
                    else if (item.equals("JASMIN_SAMPLE2.odi") ||
                            (item.equals("JASMIN_SAMPLE4.odi"))) {
                        // 일반적인 SET하나인 Sample이다.
                        default_Style(request, response, odiFetchType);
                    }
                    else {
                        System.out.println("unsupported odi error...");
                        DataModule.sendErrorMessage("unsupported odi error...",
                                                    response.getOutputStream());
                        throw new ServletException("Unknown Item :" + item);
                    }
                }
                else {
                    // ODI FetchUnit이 DM_PER_DATASET 방식이다.
                    // 이 방식일 경우에는 odiFetchType은 무조건 DM_PER_DATASET 또는
DM_CONCURRENT_FETCH 이어야 한다.
                    // 아닐 경우 지원하지 않는다. (batch 타입은 지원하지 않는다.)
                    if (odiFetchType.equalsIgnoreCase("DM_CONCURRENT_FETCH")) {
                        // set 만드는 방법 #1
                        setDefaultStyle(request, response, "DM_PER_DATASET");
                        // set 만드는 방법 #2
//                        setResultSetQueryStyle(request, response, "DM_CONCURRENT_FETCH");
                    }
                    else {
                        System.out.println("unsupported error...");
                        DataModule.sendErrorMessage("unsupported error...",
                                                    response.getOutputStream());
                        throw new ServletException("unsupported");
                    }
                }
            }
            catch(Exception ex){
                ex.printStackTrace();
                // 바인딩 외의 에러는 모두 sendErrorMessage메소드로 통해서 전송한후
                // throw를 해줘야 한다. 그렇지 않으면 client가 다운될 수 있음
                DataModule.sendErrorMessage(ex.getMessage(),
                                            response.getOutputStream());
                throw new ServletException(ex.getMessage());
            }finally{

                if (m_conn != null) {
                    try {
                        m_conn.close();
                    }
                    catch (Exception e) {
                    }
```

```
            }
            System.out.println("end--------------------");
        }

    }

    private void Master_Detail_Style(HttpServletRequest request, HttpServletResponse response,
String fetchType)
        throws ServletException, IOException
    {
        Statement stmt1 = null;
        Statement stmt4 = null;
        ResultSet rs1 = null;
        ResultSet rs4 = null;

        DataModule module = null;

            // jasmin/JASMIN_SAMPLE1.odi
            // SET_1
            String[] _FN1 = {"FirstName" };
            int[]    _FT1 = {java.sql.Types.VARCHAR };
            // SET_4
            String[] _FN4 = {"ContactId", "FirstName", "LastName", "Phone", "Fax", "Email",
"OrgUnitId", "UserName" };
            int[]    _FT4 = {4, 12, 12, 12, 12, 12, 5, 12 };
        try {
            module = DataModuleFactory.getDataModule(fetchType);

            module.init(response.getOutputStream());

            // set SET Info.
            module.addSetInfo("SET_1", "", _FN1, _FT1);
            module.addSetInfo("SET_4", "SET_1", _FN4, _FT4);
        }
        catch(Exception ex) {
            throw new ServletException(ex.getMessage());
        }
        try {
            module.startBinding();

            String query1 = "select distinct FirstName from contact";

            module.startSet("SET_1");
            stmt1 = m_conn.createStatement();
            rs1 = stmt1.executeQuery(query1);
            while(rs1.next()) {
                String f1 = rs1.getString(_FN1[0]);
                HashMap map1 = new HashMap();
                map1.put(_FN1[0], f1);

                module.addRow("SET_1", map1);

                String query4 = "select * from contact where FirstName='" + f1 + "' order by
contactid";

                module.startSet("SET_4");
                stmt4 = m_conn.createStatement();
                rs4 = stmt4.executeQuery(query4);
                while(rs4.next()) {
                    HashMap map4 = new HashMap();
                    map4.put(_FN4[0], new Integer(rs4.getInt(_FN4[0])));
                    map4.put(_FN4[1], rs4.getString(_FN4[1]));
                    map4.put(_FN4[2], rs4.getString(_FN4[2]));
                    map4.put(_FN4[3], rs4.getString(_FN4[3]));
                    map4.put(_FN4[4], rs4.getString(_FN4[4]));
```

```
                    map4.put(_FN4[5], rs4.getString(_FN4[5]));
                    map4.put(_FN4[6], new Integer(rs4.getInt(_FN4[6])));
                    map4.put(_FN4[7], rs4.getString(_FN4[7]));

                    module.addRow("SET_4", map4);
                }
                rs4.close(); rs4 = null;
                stmt4.close(); stmt4 = null;

                module.endSet("SET_4");
            }

            rs1.close(); rs1 = null;
            stmt1.close(); stmt1 = null;

            module.endSet("SET_1");

            module.endBinding();
        }
        catch(Exception e) {
            e.printStackTrace();
            // 바인딩시 에러 전송은 sendBindErrorMessage메소드로 꼭 해야 한다.
            if(module != null) {
                module.sendBindErrorMessage(e.toString());
            }
        }
        finally {

            if(rs1 != null) {
                try { rs1.close(); } catch(Exception e) {}
            }
            if(stmt1 != null) {
                try { stmt1.close(); } catch(Exception e) {}
            }
            if(rs4 != null) {
                try { rs4.close(); } catch(Exception e) {}
            }
            if(stmt4 != null) {
                try { stmt4.close(); } catch(Exception e) {}
            }
        }
    }

    private void default_Style(HttpServletRequest request, HttpServletResponse response, String
fetchType)
        throws ServletException, IOException
    {
        System.out.println("select time=="+System.currentTimeMillis());
        Statement stmt = null;
        ResultSet rs = null;

        DataModule module = null;

        // jasmin/JASMIN_SAMPLE2.odi
        // SET_1
        String[] _FN = {"CarID", "Maker", "EMaker", "CarName", "ECarName",
                        "CarImageFile"};
        int[] _FT = {
            java.sql.Types.VARCHAR, java.sql.Types.VARCHAR,
            java.sql.Types.VARCHAR, java.sql.Types.VARCHAR,
            java.sql.Types.VARCHAR, java.sql.Types.VARCHAR
        };

       try {
            module = DataModuleFactory.getDataModule(fetchType);
```

```
            module.init(response.getOutputStream());
//            module.init(output);

            // set Parameter Info.
            module.addParameter("PARAM1", java.sql.Types.VARCHAR,
                                getEncode(request.getParameter("PARAM1")));
            module.addParameter("PARAM2", java.sql.Types.VARCHAR,
                                getEncode(request.getParameter("PARAM2")));

            // set SET Info.
            module.addSetInfo("SET_1", "", _FN, _FT);
        }catch(Exception ex){
            throw new ServletException(ex.getMessage());
        }



        try {
            module.startBinding();

            String query = "select * from car";

            module.startSet("SET_1");
            stmt = m_conn.createStatement();
            rs = stmt.executeQuery(query);
            while(rs.next()) {
                HashMap map = new HashMap();
                String a = rs.getString(_FN[0]);
                String b = rs.getString(_FN[1]);
                String c = rs.getString(_FN[2]);
                String d = rs.getString(_FN[3]);
                String e = rs.getString(_FN[4]);
                String f = rs.getString(_FN[5]);

                map.put(_FN[0], a);
                map.put(_FN[1], b);
                map.put(_FN[2], c);
                map.put(_FN[3], d);
                map.put(_FN[4], e);
                map.put(_FN[5], f);

                module.addRow("SET_1", map);
                System.out.print("select FN0="+_FN[0]+"   name="+a);
                System.out.print("      FN1="+_FN[1]+"   name="+b);
                System.out.print("      FN2="+_FN[2]+"   name="+c);
                System.out.print("      FN3="+_FN[3]+"   name="+d);
                System.out.print("      FN4="+_FN[4]+"   name="+e);
                System.out.print("      FN5="+_FN[5]+"   name="+f);
                System.out.println("");

            }
            module.endSet("SET_1");
            module.endBinding();

            rs.close(); rs = null;
            stmt.close(); stmt = null;

        }
        catch(Exception e) {
            e.printStackTrace();
            // 바인딩시 에러 전송은 sendBindErrorMessage메소드로 꼭 해야 한다.
            module.sendBindErrorMessage(e.toString());
        }
        finally {
            OutputStream out = response.getOutputStream();
```

```
            if(rs != null) {
                try { rs.close(); } catch(Exception e) {}
            }
            if(stmt != null) {
                try { stmt.close(); } catch(Exception e) {}
            }
        }
    }


    private void setDefaultStyle(HttpServletRequest request, HttpServletResponse response, String
fetchType)
        throws ServletException, IOException
    {
        System.out.println("setDefaultStyle fetchType =="+fetchType);
        Statement stmt = null;
        ResultSet rs = null;

        DataModule module = null;

        // jasmin/JASMIN_SAMPLE4.odi
        // SET_1
        String[] _FN = {"CarID", "Maker", "EMaker", "CarName", "ECarName",
                        "CarImageFile"};
        int[] _FT = {
            java.sql.Types.VARCHAR, java.sql.Types.VARCHAR,
            java.sql.Types.VARCHAR, java.sql.Types.VARCHAR,
            java.sql.Types.VARCHAR, java.sql.Types.VARCHAR
        };

       try {
            module = DataModuleFactory.getDataModule(fetchType);
            module.init(response.getOutputStream());

            // set SET Info.
            module.addSetInfo("SET_1", _FN, _FT);
        }catch(Exception ex){
            throw new ServletException(ex.getMessage());
        }

        try {
            module.startBinding();

            String query = "select * from car";

            module.startSet("SET_1");
            stmt = m_conn.createStatement();
            rs = stmt.executeQuery(query);
            while(rs.next()) {
                HashMap map = new HashMap();
                String a = rs.getString(_FN[0]);
                String b = rs.getString(_FN[1]);
                String c = rs.getString(_FN[2]);
                String d = rs.getString(_FN[3]);
                String e = rs.getString(_FN[4]);
                String f = rs.getString(_FN[5]);

                map.put(_FN[0], a);
                map.put(_FN[1], b);
                map.put(_FN[2], c);
                map.put(_FN[3], d);
                map.put(_FN[4], e);
                map.put(_FN[5], f);
```

```
                    module.addRow("SET_1", map);
                    System.out.print("select FN0="+_FN[0]+"   name="+a);
                    System.out.print("      FN1="+_FN[1]+"   name="+b);
                    System.out.print("      FN2="+_FN[2]+"   name="+c);
                    System.out.print("      FN3="+_FN[3]+"   name="+d);
                    System.out.print("      FN4="+_FN[4]+"   name="+e);
                    System.out.print("      FN5="+_FN[5]+"   name="+f);
                    System.out.println("");

               }
               module.endSet("SET_1");
               module.endBinding();

               rs.close(); rs = null;
               stmt.close(); stmt = null;

          }
          catch(Exception e) {
               e.printStackTrace();
               // 바인딩시 에러 전송은 sendBindErrorMessage메소드로 꼭 해야 한다.
               module.sendBindErrorMessage(e.toString());
          }
          finally {
               OutputStream out = response.getOutputStream();

               if(rs != null) {
                    try { rs.close(); } catch(Exception e) {}
               }
               if(stmt != null) {
                    try { stmt.close(); } catch(Exception e) {}
               }
          }
     }


     private void setResultSetQueryStyle(HttpServletRequest request, HttpServletResponse response,
String fetchType)
          throws ServletException, IOException
     {
          System.out.println("setResultSetQueryStyle...");
          Statement stmt = null;
          ResultSet rs = null;

          DataModule module = null;
          try {
               // jasmin/JASMIN_SAMPLE4.odi
               // SET_1
               String[]   _FN   =   {"CarID",   "Maker",   "EMaker",   "CarName",   "ECarName",
"CarImageFile" };
               int[]      _FT = {java.sql.Types.VARCHAR, java.sql.Types.VARCHAR,
                                   java.sql.Types.VARCHAR, java.sql.Types.VARCHAR,
                                   java.sql.Types.VARCHAR,java.sql.Types.VARCHAR};

               module = DataModuleFactory.getDataModule(fetchType);

               String query = "select * from car";

               stmt = m_conn.createStatement();
               rs = stmt.executeQuery(query);

               module.makeSDM_SET("SET_1",rs,response.getOutputStream());

               rs.close(); rs = null;
               stmt.close(); stmt = null;
```

```
            }
            catch(Exception e) {
                e.printStackTrace();
                // set단위로 할 경우에 에러발생시 sendBindErrorMessage를 하지 않는다.
                // 왜냐하면 makeSDM_SET메소드 안에 처리가 되어 있다.
                throw new ServletException(e.getMessage());
            }
            finally {
                if(rs != null) {
                    try { rs.close(); } catch(Exception e) {}
                }
                if(stmt != null) {
                    try { stmt.close(); } catch(Exception e) {}
                }
            }

        }

        public void destroy()
        {
            super.destroy();
        }

        private String getEncode(String value) {
            try {
                return new String(value.getBytes("8859_1"), "KSC5601");
            }catch(Exception ex){
                return value;
            }
        }

}
```

Step 3 **클래스 파일 등록하기**

DataModuleSampleServlet.class 파일을 WAS에 등록합니다.

- DataModuleSampleServlet.class 파일을 웹 서버의 클래스 폴더에 복사합니다. 본 예제에서는 Tomcat 5.0의 webapps\ROOT\WEB-INF\classes 폴더에 sample 폴더를 새로 만든 후 클래스 파일을 복사하였습니다.

- DataModuleSampleServlet.class 파일을 WAS에 매핑합니다. 본 예제에서는 Tomcat 5.0의 webapps\ROOT\WEB-INF의 web.xml 파일을 편집기로 열어 아래와 같이 편집하여 복사한 클래스 파일을 매핑하였습니다.

```
...

<!-- JSPC servlet mappings start -->
...
<servlet>
    <servlet-name>sample.DataModuleSampleServlet</servlet-name>
    <servlet-class>sample.DataModuleSampleServlet</servlet-class>
</servlet>
```

```
...
<servlet-mapping>
        <servlet-name>sample.DataModuleSampleServlet</servlet-name>
        <url-pattern>/sample.DataModuleSampleServlet</url-pattern>
</servlet-mapping>

...
<!-- JSPC servlet mappings end -->
```

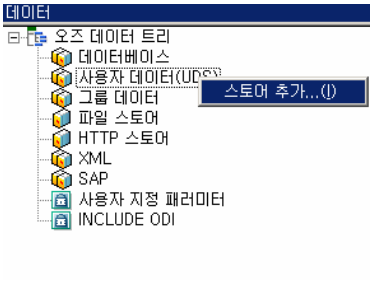
Step 4 ODI 만들기

Servlet API를 이용할 경우에는 사용자 데이터(UDS) 스토어를 이용하고, 데이터셋과 데이터 필드를 추가하여야 합니다.

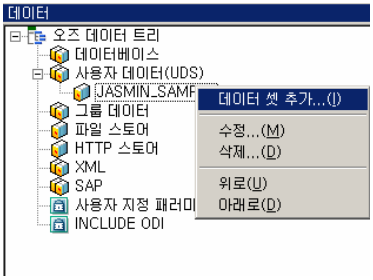
데이터셋과 데이터 필드를 추가할 때에는 수동으로 하나씩 추가하는 방법과 POST 패러미터 중 "_OZ_ODIITEM_"을 이용하여 자동으로 추가하는 방법이 있습니다.

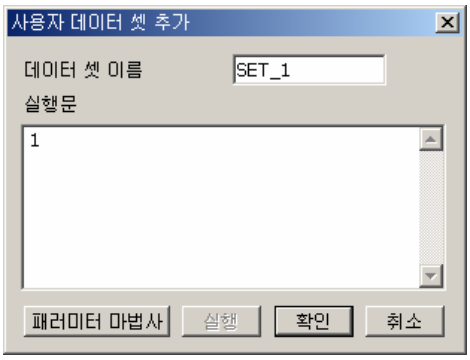
먼저 수동으로 데이터셋과 데이터 필드를 추가하는 방법은 다음과 같습니다.

- 오즈 쿼리 디자이너를 실행하여 아래와 같이 사용자 데이터(UDS) 스토어를 추가합니다.

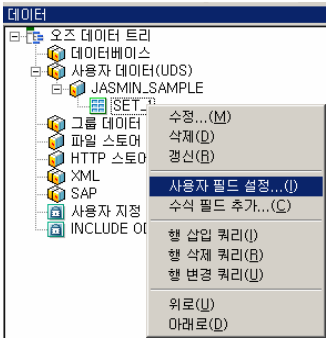


- 사용자 데이터 스토어에 데이터셋을 추가합니다.

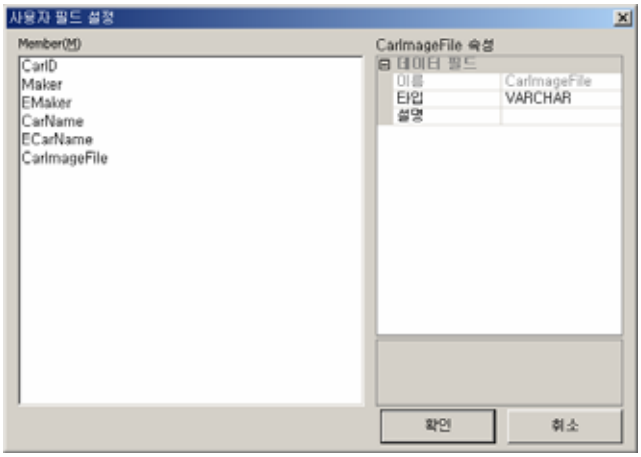




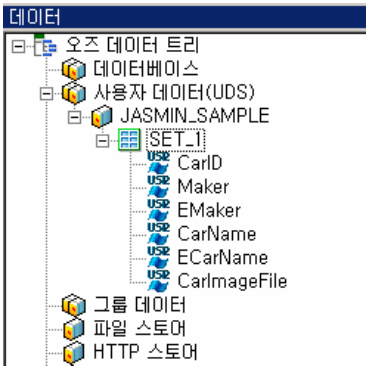
➢ 추가된 데이터셋에 사용자 필드를 추가합니다. 추가된 데이터셋을 마우스 오른쪽 버튼으로 클릭하여 나타나는 팝업 메뉴에서 [사용자 필드 설정] 메뉴를 클릭합니다.



➢ 사용자 필드 설정 다이얼로그에서 아래 그림처럼 사용자 필드를 추가한 후 [확인] 버튼을 누릅니다.

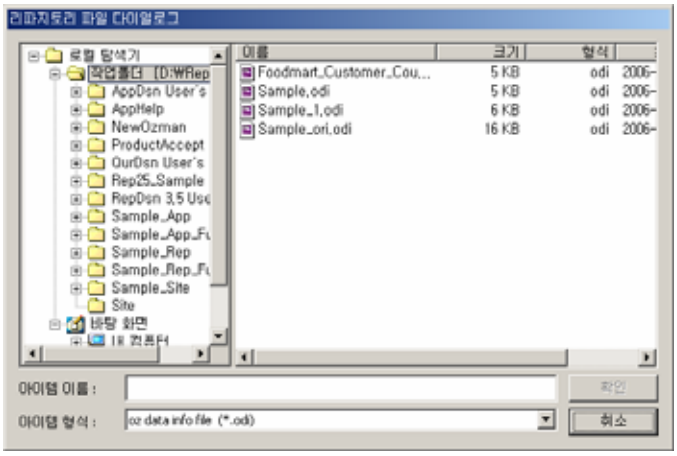


오즈 데이터 트리에 사용자 필드가 아래 그림처럼 추가되었는지 확인합니다.

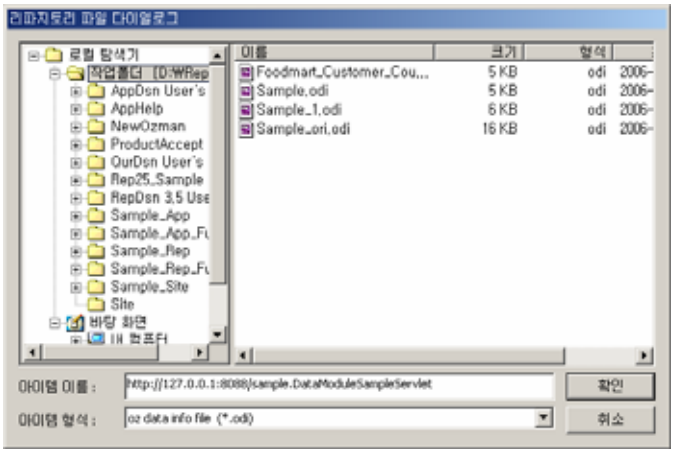


다음으로 자동으로 데이터셋과 데이터 필드를 추가하는 방법은 다음과 같습니다.

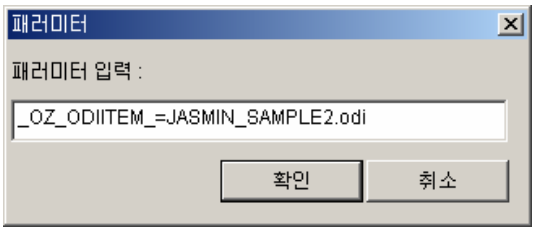
- 오즈 쿼리 디자이너를 실행한 후 툴바에서 열기( ) 아이콘을 클릭하여 리파지토리 열기 다이얼로그를 띄웁니다.



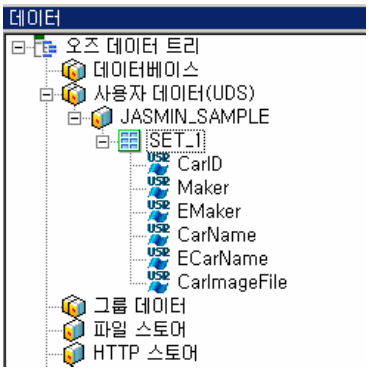
- 아이템 이름에 클래스 파일의 URL을 입력한 후 [확인] 버튼을 클릭합니다.



➢ 패러미터 입력 다이얼로그에 "_OZ_ODIITEM_=ODI이름"을 입력한 후 [확인] 버튼을 클릭합니다.



➢ 데이터 트리에 데이터셋과 데이터 필드가 추가되어 표시됩니다.



➢ 수동 또는 자동 방식으로 만들어진 ODI 파일을 "JASMIN_SAMPLE2.odi" 파일로 저장합니다.

Step 5 OZF와 프로젝트 만들기

※ 주의사항 : Servlet API를 이용하여 데이터 연동을 오즈 서버가 아닌 다른 서버와 연동할 경우 연동할 서버의 URL을 지정하기 위해서는 DataModule의 RegisterUserDataModule 함수를 이용하여 JavaScript 객체를 등록하여야 합니다. JavaScript 객체를 등록하는 함수는 반드시 전역 함수로 작성되어야 하므로, Document.GlobalFunction에 작성하거나 OZF로 작성한 후 해당 OZF 파일을 프로젝트에 추가하여 사용합니다. 본 예제에서는 OZF로 작성한 후 프로젝트에 추가하는 예로 설명하겠습니다.

➢ 오즈 애플리케이션 함수 편집기를 실행하여 OZF를 새로 만든 후 스크립트 편집창에 아래와 같이 입력한 후 "MyFrameworkURLUDS.ozf"로 저장합니다.

```
MyFrameworkURLUDS.prototype.GetFrameworkURL =
                        MyFrameworkURLUDS_GetFrameworkURL;
MyFrameworkURLUDS.prototype.GetCUDFrameworkURL =
```

```
                        MyFrameworkURLUDS_GetCUDFrameworkURL;
MyFrameworkURLUDS.prototype.GetFrameworkPostParam =
                        MyFrameworkURLUDS_GetFrameworkPostParam;
MyFrameworkURLUDS.prototype.GetCUDFrameworkPostParam =
                        MyFrameworkURLUDS_GetCUDFrameworkPostParam;

function MyFrameworkURLUDS(_url, _cud_url, _url_param, _cud_url_param){
    this.url = _url;
    this.cud_url = _cud_url;
    this.url_param = _url_param;
    if(this.url_param == null){
        this.url_param = "";
     }
    this.cud_url_param = _cud_url_param;
    if(this.cud_url_param == null){
        this.cud_url_param = "";
     }
}

function MyFrameworkURLUDS_GetFrameworkURL(dataset_name){
    return this.url;
}

function MyFrameworkURLUDS_GetCUDFrameworkURL(dataset_name){
    return this.cud_url;
}

function MyFrameworkURLUDS_GetFrameworkPostParam(dataset_name){
    return "default&"+this.url_param;
}

function MyFrameworkURLUDS_GetCUDFrameworkPostParam(dataset_name){
    return "default&"+this.cud_url_param;
}
```

※ 참고사항

① 객체에는 GetFrameworkURL과 GetCUDFrameworkURL 함수가 반드시 입력되어 있어야 합니다.

② GetFrameworkURL 함수는 서버로부터 데이터를 가져올 경우 사용되는 URL을 얻을 때 호출됩니다.

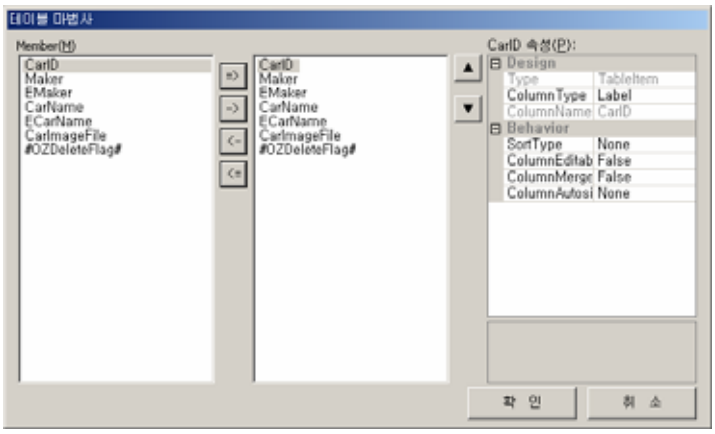
③ GetCUDFrameworkURL 함수는 서버에 DataAction을 보내는 경우 사용되는 URL을 얻을 때 호출됩니다.

④ 위 예제에서는 데이터셋 이름과 상관없이 항상 같은 URL을 사용하도록

구현한 예이며, 데이터셋 이름에 따라 URL을 다르게 사용할 수도 있습니다.

- 오즈 애플리케이션 디자이너를 실행하여 프로젝트를 새로 만든 후 프로젝트에 "JASMIN_SAMPLE2.odi" 파일과 "MyFrameworkURLUDS.ozf" 파일을 추가합니다.

- Board에 Table을 추가한 후 Table의 ODIKey 속성값을 "JASMIN_SAMPLE2"로, DataSet 속성값을 "SET_1"을 설정한 후 테이블 마법사를 실행하여, 아래 그림처럼 테이블에 표시할 필드를 설정합니다.



- Table의 OnInitialize 이벤트에 아래와 같이 스크립트를 입력합니다.

```
var uds = new MyFrameworkURLUDS("http://127.0.0.1:8088/sample.DataModuleSampleServlet","http://127.0.0.1:8088/sample.DataModuleSampleServlet");
var datamanager = _GetDataManager();
var datamodule = datamanager.GetDataModule("JASMIN_SAMPLE2");
datamodule.RegisterUserDataModule(uds);
```

Step 6 미리보기

WAS를 구동한 후 오즈 애플리케이션 디자이너 툴바에서 미리보기 아이콘을 클릭하여 미리보기합니다.

	CarID	Maker	EMaker	CarName	ECarName	CarImageFile
1	H04	현대자동차	HyunDai	EF소나타	EFSONATA	http://127.0.0.1/img/EF소나타.gif
2	H02	현대자동차	HyunDai	다이네스티	DYNASTY	http://127.0.0.1/img/다이네스티.gif
3	H03	현대자동차	HyunDai	그랜져	GRANDEUR	http://127.0.0.1/img/그랜져.gif
4	H01	현대자동차	HyunDai	에쿠우스	EQUUS	http://127.0.0.1/img/에쿠우스.gif
5	H05	현대자동차	HyunDai	베르나	VERNA	http://127.0.0.1/img/베르나.gif
6	H06	현대자동차	HyunDai	아토스	ATOZ	http://127.0.0.1/img/아토스.gif
7	D01	대우자동차	DaeWoo	라노스	RANOS	http://127.0.0.1/img/라노스.gif
8	D02	대우자동차	DaeWoo	누비라	NUBIRA	http://127.0.0.1/img/누비라.gif
9	D03	대우자동차	DaeWoo	매그너스	MAGNUS	http://127.0.0.1/img/매그너스.gif
10	K01	기아자동차	KIA	세피아 I	SEPHIA	http://127.0.0.1/img/세피아.jpg
11	K03	기아자동차	KIA	세피아 II	SEPHIA II	http://127.0.0.1/img/세피아2.jpg

예제 2 : Application - DataAction

다음은 Servlet API를 이용하여 Table에 표시된 데이터를 삽입, 삭제, 변경하는 예제를 살펴보겠습니다.

Step 1 ozsdmapi.jar 파일을 WAS에 등록하기

Servlet API 연동 모듈인 ozsdmapi.jar 파일을 WAS의 라이브러리에 등록합니다. 본 예제에서는 Tomcat 5.0의 webapps\ROOT\WEB-INF\lib 폴더에 ozsdmapi.jar 파일을 복사하였습니다.

Step 2 DataAction용 클래스 파일 만들기

DataAction을 실행하는 클래스 파일을 만듭니다. 본 예제에서는 아래와 같은 java 소스 파일을 이용하여 DataActionSampleServlet.class 파일로 만들었습니다.

```
package sample;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;

import java.util.*;

/**
 * <p>Title: OZ SDM API</p>
```

```
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: </p>
 * @author Forcs
 * @version 1.0
 */

public class DataActionSampleServlet extends HttpServlet
{
    private static final String _KEY_ODI_FETCH_TYPE = "_OZ_ODIFetchType_";
    private static final String _KEY_ODIITEM        = "_OZ_ODIITEM_";
    private static final String _KEY_ODICATEGORY    = "_OZ_ODICATEGORY_";
    private static final String _KEY_OZ_DATASET_    = "_OZ_DATASET_";

    private static final String _KEY_OZ_DAC_CNT = "_OZ_DAC_CNT";

    private static String TABLENAME = "car";

    private String dac_insert_query = "";
    private String dac_update_query = "";
    private String dac_delete_query = "";

    private PreparedStatement p_insert_stmt = null;
    private PreparedStatement p_update_stmt = null;
    private PreparedStatement p_delete_stmt = null;

    private Connection m_conn = null;

    public void init(ServletConfig config)
        throws ServletException
    {
        super.init(config);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        process(request, response);
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        process(request, response);
    }

    private void process(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {

        try {
            String _URL = "jdbc:odbc:ozdemokr30";
            Properties prop = new Properties();
            prop.put("user", "");
            prop.put("password", "");

            Driver driver = (Driver)
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
            m_conn = driver.connect(_URL, prop);
        }
        catch(Exception ex) {
            response.getOutputStream().write(ex.getMessage().getBytes());
            throw new ServletException(ex.getMessage());
        }
```

```
try {
    dac_insert_query = "";
    dac_update_query = "";
    dac_delete_query = "";

    Enumeration enum = request.getParameterNames();
    while (enum.hasMoreElements()) {
        String temp = (String) enum.nextElement();
        System.out.println("name=" + temp + "     value=" +
                           getEncode(request.getParameter(temp)));
    }

    // ----------------------------------------------------------------------
    // fetch parameters
    // "DM_BATCH_FETCH|DM_CONCURRENT_FETCH"
    String odiFetchType = getEncode(request.getParameter(
        _KEY_ODI_FETCH_TYPE));
    String item = getEncode(request.getParameter(_KEY_ODIITEM));
    String category = getEncode(request.getParameter(_KEY_ODICATEGORY));
    String dataset = getEncode(request.getParameter(_KEY_OZ_DATASET_));
    // ----------------------------------------------------------------------
    System.out.println("--------------------");
    if (odiFetchType == null) odiFetchType = "DM_CONCURRENT_FETCH";

    m_conn.setAutoCommit(false);

    //데이터Action의 카운트를 우선 계산한다.
    int dac_cnt = Integer.parseInt(request.getParameter(_KEY_OZ_DAC_CNT));

    for (int i = 0; i < dac_cnt; i++) {
        String type = request.getParameter(i + ".TYPE");
        if (type.equalsIgnoreCase("insert")) {
            insert(request, response, i);
        }
        else if (type.equalsIgnoreCase("rowupdate")) {
            update(request, response, i);
        }
        else if (type.equalsIgnoreCase("delete")) {
            delete(request, response, i);
        }
        System.out.println(type+" time=="+System.currentTimeMillis());
    }
    m_conn.commit();
    response.getOutputStream().write(new String("OK").getBytes());
}catch(Exception ex){
    ex.printStackTrace();
    try { m_conn.rollback(); } catch(Exception e) {}
    response.getOutputStream().write(ex.getMessage().getBytes());
    throw new ServletException(ex.getMessage());

}finally {
    if(p_insert_stmt != null) {
        try { p_insert_stmt.close(); } catch(Exception e) {}
    }
    if(p_update_stmt != null) {
        try { p_update_stmt.close(); } catch(Exception e) {}
    }
    if(p_delete_stmt != null) {
        try { p_delete_stmt.close(); } catch(Exception e) {}
    }

    if (m_conn != null) {
        try { m_conn.close(); } catch(Exception e) {}
    }
```

```
        }
    }

    private void insert(HttpServletRequest request, HttpServletResponse response, int cnt)
        throws Exception
    {
        int insert_source_fieldCnt = Integer.parseInt(request.getParameter(cnt + ".SRC_CNT"));

        if (dac_insert_query.equalsIgnoreCase("")) {
            //맨 처음 호출한 경우 쿼리문을 만들어서 prepared화 시킨다.
            dac_insert_query = "INSERT INTO "+TABLENAME+" (";
            for (int i = 0; i < insert_source_fieldCnt; i++) {
                String s_fieldName = getEncode(request.getParameter(cnt + ".SF_" + i));
                if (i == insert_source_fieldCnt - 1) {
                    dac_insert_query += s_fieldName;
                }
                else {
                    dac_insert_query += s_fieldName + ",";
                }
            }
            dac_insert_query += ") VALUES ( ";

            for (int i = 0; i < insert_source_fieldCnt; i++) {
                if (i == insert_source_fieldCnt - 1) {
                    dac_insert_query += "?";
                }
                else {
                    dac_insert_query += "? ,";
                }
            }

            dac_insert_query += ")";
            System.out.println("dac_insert_query="+dac_insert_query);
            p_insert_stmt = m_conn.prepareStatement(dac_insert_query);
        }

        for (int i = 0; i < insert_source_fieldCnt; i++) {
            String value = getEncode(request.getParameter(cnt + ".SV_" + i));
            p_insert_stmt.setString(i + 1, value);
        }

        p_insert_stmt.execute();

    }

    private void update(HttpServletRequest request, HttpServletResponse response, int cnt)
        throws Exception
    {

        int update_source_fieldCnt = Integer.parseInt(request.getParameter(cnt + ".SRC_CNT"));
        int update_target_fieldCnt = 0;

        try {
            update_target_fieldCnt = Integer.parseInt(request.getParameter(cnt + ".TRG_CNT"));
        }catch(Exception ex){}

        if (dac_update_query.equalsIgnoreCase("")) {
            //맨 처음 호출한 경우 쿼리문을 만들어서 prepared화 시킨다.

            dac_update_query = "UPDATE "+TABLENAME+" set ";

            for (int i = 0; i < update_source_fieldCnt; i++) {
                String s_fieldName = getEncode(request.getParameter(cnt + ".SF_" + i));

                if (i == update_source_fieldCnt - 1) {
```

```
                    dac_update_query += s_fieldName +" = ? ";
                }
                else {
                    dac_update_query += s_fieldName +" = ?, ";
                }
            }
            dac_update_query += " WHERE ";

            for (int i = 0; i < update_target_fieldCnt; i++) {
                String t_fieldName = getEncode(request.getParameter(cnt + ".DF_" + i));

                if (i == update_target_fieldCnt - 1) {
                    dac_update_query += t_fieldName +" = ? ";
                }
                else {
                    dac_update_query += t_fieldName +" = ? AND ";
                }
            }


            System.out.println("dac_update_query="+dac_update_query);
            p_update_stmt = m_conn.prepareStatement(dac_update_query);
        }

        int i = 0;
        for (i = 0; i < update_source_fieldCnt; i++) {
            String value = getEncode(request.getParameter(cnt + ".SV_" + i));
            p_update_stmt.setString(i + 1, value);
        }

        for (int j = 0; j < update_source_fieldCnt; j++) {
            String value = getEncode(request.getParameter(cnt + ".DV_" + j));
            p_update_stmt.setString(i + 1, value);
            i++;
        }
        p_update_stmt.execute();

    }

    private void delete(HttpServletRequest request, HttpServletResponse response, int cnt)
            throws Exception
     {

            int delete_target_fieldCnt = Integer.parseInt(request.getParameter(cnt + ".TRG_CNT"));

            if (dac_delete_query.equalsIgnoreCase("")) {
                //맨 처음 호출한 경우 쿼리문을 만들어서 prepared화 시킨다.

                dac_delete_query = "DELETE FROM "+TABLENAME+" WHERE ";

                for (int i = 0; i < delete_target_fieldCnt; i++) {
                    String t_fieldName = getEncode(request.getParameter(cnt + ".DF_" + i));

                    if (i == delete_target_fieldCnt - 1) {
                        dac_delete_query += t_fieldName +" = ? ";
                    }
                    else {
                        dac_delete_query += t_fieldName +" = ? AND ";
                    }
                }

                System.out.println("dac_delete_query="+dac_delete_query);
                p_delete_stmt = m_conn.prepareStatement(dac_delete_query);
            }
```

```
            int i = 0;
            for (i = 0; i < delete_target_fieldCnt; i++) {
                String value = getEncode(request.getParameter(cnt + ".DV_" + i));
                p_delete_stmt.setString(i + 1, value);
            }

            p_delete_stmt.execute();

        }

    public void destroy()
    {
        super.destroy();
    }

    private String getEncode(String value) {
        try {
            return new String(value.getBytes("8859_1"), "KSC5601");
        }catch(Exception ex){
            return value;
        }
    }

}
```

Step 3 **클래스 파일 등록하기**

"예제 1"에서 만든 데이터 검색용 클래스 파일인 DataModuleSampleServlet.class 파일과 DataActionSampleServlet.class 파일을 WAS에 등록합니다.

- DataModuleSampleServlet.class, DataActionSampleServlet.class 파일을 WAS의 클래스 폴더에 복사합니다. 본 예제에서는 Tomcat 5.0의 webapps\ROOT\WEB-INF\classes폴더에 sample 폴더를 새로 만든 후 두 개의 클래스 파일을 복사하였습니다.

- DataModuleSampleServlet.class, DataActionSampleServlet.class 파일을 WAS에 매핑합니다. 본 예제에서는 Tomcat 5.0의 webapps\ROOT\WEB-INF의 web.xml 파일을 편집기로 열어 아래와 같이 편집하여 복사한 클래스 파일을 매핑하였습니다.

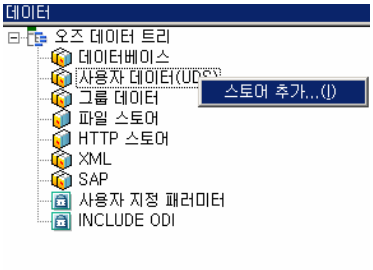
```
...

<!-- JSPC servlet mappings start -->
...
<servlet>
    <servlet-name>sample.DataModuleSampleServlet</servlet-name>
    <servlet-class>sample.DataModuleSampleServlet</servlet-class>
</servlet>

<servlet>
    <servlet-name>sample.DataActionSampleServlet</servlet-name>
```

```
    <servlet-class>sample.DataActionSampleServlet</servlet-class>
</servlet>

...
<servlet-mapping>
        <servlet-name>sample.DataModuleSampleServlet</servlet-name>
        <url-pattern>/sample.DataModuleSampleServlet</url-pattern>
</servlet-mapping>

<servlet-mapping>
        <servlet-name>sample.DataActionSampleServlet</servlet-name>
        <url-pattern>/sample.DataActionSampleServlet</url-pattern>
</servlet-mapping>

...
<!-- JSPC servlet mappings end -->
```

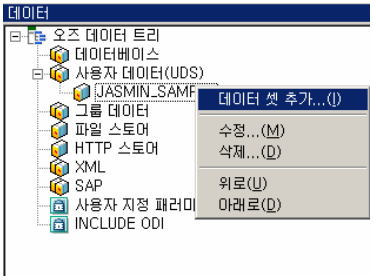
Step 4 ODI 만들기

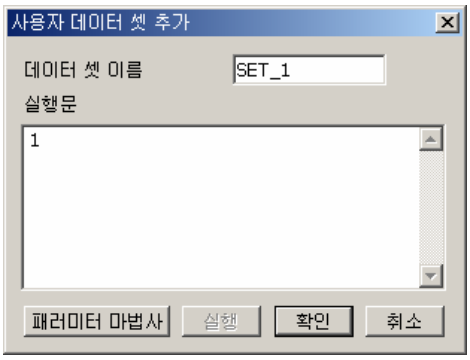
Servlet API를 이용할 경우에는 사용자 데이터(UDS) 스토어를 이용하고, 데이터셋에 사용자 정의 필드를 수동으로 추가하여야 합니다.

- 오즈 쿼리 디자이너를 실행하여 아래와 같이 사용자 데이터(UDS) 스토어를 추가합니다.

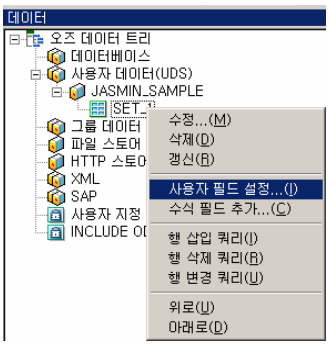


- 추가된 스토어에 데이터셋을 추가합니다.

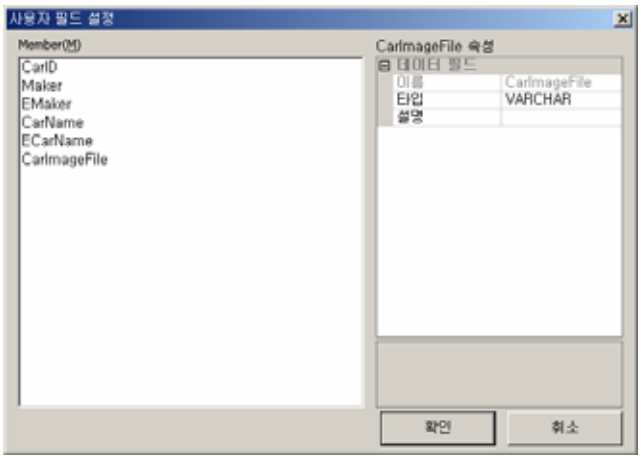




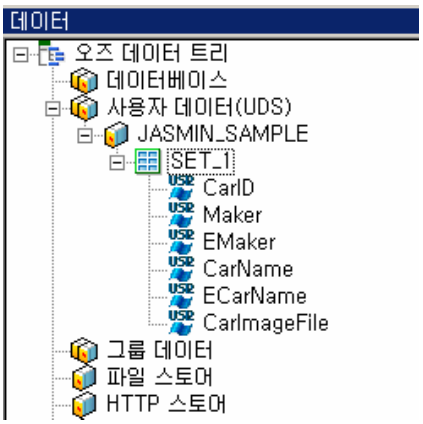
- 추가된 데이터셋에 사용자 필드를 추가합니다. 추가된 데이터셋을 마우스 오른쪽 버튼으로 클릭하여 나타나는 팝업 메뉴에서 [사용자 필드 설정] 메뉴를 클릭합니다.



- 사용자 필드 설정 다이얼로그에서 아래 그림처럼 사용자 필드를 추가한 후 [확인] 버튼을 누릅니다.



데이터 정보 트리에 사용자 필드가 아래 그림처럼 추가되었는지 확인합니다.



- SET_1을 마우스 오른쪽 버튼으로 클릭하여 각각 [행 삽입 쿼리], [행 삭제 쿼리], [행 변경 쿼리] 메뉴를 눌러 각각 행 삽입 쿼리, 행 삭제 쿼리, 행 변경 쿼리문을 입력합니다.

입력한 행 삽입 쿼리문은 다음과 같습니다.

```
INSERT INTO car(#@ARG_SF1#, #@ARG_SF2#, #@ARG_SF3#, #@ARG_SF4#,
#@ARG_SF5#, #@ARG_SF6#)
VALUES('#@ARG_SV1#','#@ARG_SV2#', '#@ARG_SV3#','#@ARG_SV4#',
'#@ARG_SV5#', '#@ARG_SV6#')
```

입력한 행 삭제 쿼리문은 다음과 같습니다.

```
DELETE FROM car WHERE [#@ARG_DF1#] = '#@ARG_DV1#';
```

입력한 행 변경 쿼리문은 다음과 같습니다.

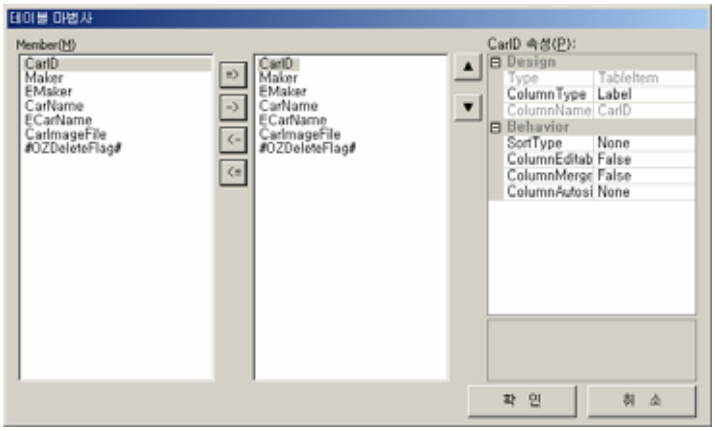
```
UPDATE car SET
[#@ARG_SF1#] = '#@ARG_SV1#', [#@ARG_SF2#] = '#@ARG_SV2#',
[#@ARG_SF3#] = '#@ARG_SV3#', [#@ARG_SF4#] = '#@ARG_SV4#',
[#@ARG_SF5#] = '#@ARG_SV5#', [#@ARG_SF6#] = '#@ARG_SV6#'
WHERE [#@ARG_DF1#] = '#@ARG_DV1#'
```

- ODI 파일을 "JASMIN_SAMPLE4.odi" 파일로 저장합니다.

Step 5 프로젝트 만들기

- 오즈 애플리케이션의 새로운 프로젝트에 "JASMIN_SAMPLE4.odi" 파일과 "예제 1"에서 만든 "MyFrameworkURLUDS.ozf" 파일을 추가합니다.

- Board에 Table을 추가한 후 Table의 ODIKey 속성값을 "JASMIN_SAMPLE4"로, DataSet 속성값을 "SET_1"을 설정한 후 테이블 마법사를 실행하여, 아래 그림처럼 테이블에 표시할 필드를 설정합니다.



- Table에서 데이터를 삽입, 삭제, 변경하기 위해서 Table의 속성 중 "AllowInsert", "AllowDelete", "AllowUpdate" 속성값을 모두 "True"로 변경합니다.

- Table의 OnInitialize 이벤트에 아래와 같이 스크립트를 입력합니다.

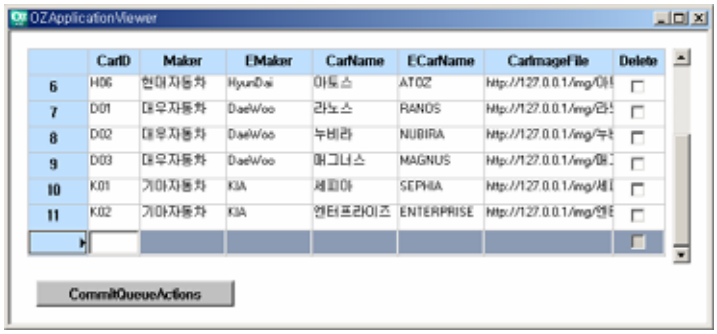
```
var uds = new MyFrameworkURLUDS("http://127.0.0.1:8088/sample.DataModuleSampleServlet","http://127.0.0.1:8088/sample.DataActionSampleServlet");
var datamanager = _GetDataManager();
var datamodule = datamanager.GetDataModule("JASMIN_SAMPLE4");
datamodule.RegisterUserDataModule(uds);
```

➢ Board에 Button을 추가한 후 Button의 OnClick 이벤트에 아래와 같이 스크립트를 입력합니다.

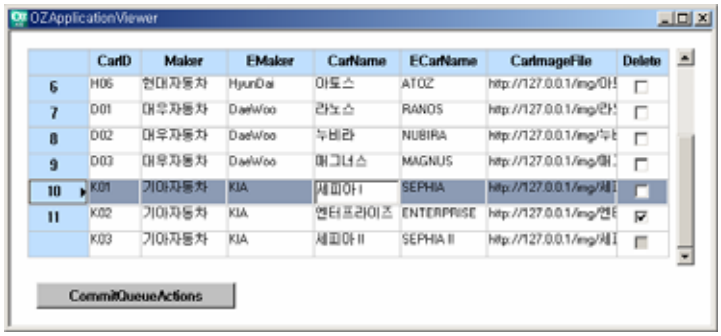
```
var result = Table1.CommitQueuedActions();
if(result == "") {
    Table1.GetDataModule().RefreshAllDataSet();
} else {
    _MessageBox(result);
}

//var myODIObject =
  _GetDataManager().GetDataModule("odiName").GetUserDataModule();
//Select할때 기본 패러미터에 다른 패러미터를 추가하고자 할 때 사용
//        myODIObject.url_param = "key1=value1&key2=value2";
//DataAction할때 기본 패러미터에 다른 패러미터를 추가하고자 할 때 사용
//      myODIObject.cud_url_param = "key1=value1&key2=value2";
//_GetDataManager().GetDataModule("odiName").RefreshAllDataSet();
```

Step 7 미리보기

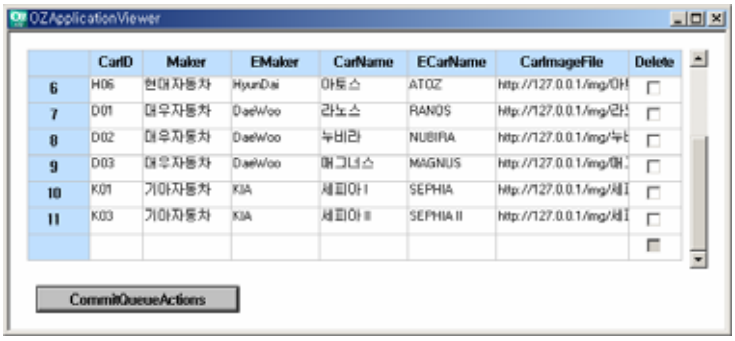
웹 서버를 구동한 후 오즈 애플리케이션 디자이너 툴바에서 미리보기 아이콘을 클릭하여 미리보기 합니다.



맨 마지막에 비어있는 행에 추가할 데이터 내용을 입력합니다. 삭제할 행은 Delete 컬럼의 체크 박스를 체크합니다. 변경할 데이터 내용을 직접 입력하여 변경합니다.



입력, 삭제, 변경한대로 DataAction을 실행하려면 버튼을 클릭하여 적용합니다. [CommitQueueActions] 버튼을 클릭하여 적용한 모습은 아래 그림과 같습니다.



예제 3 : Report - 데이터 검색

다음으로 Servlet API를 이용하여 테이블에 데이터를 표시하는 보고서 파일을 웹 브라우저에서 호출하는 예제를 살펴보겠습니다. Servlet API를 연동할 때에는 뷰어 패러미터 중 framworkurl 패러미터를 이용하여 클래스 파일의 경로(HTTP URL)를 설정하여야 합니다. 관련된 뷰어 패러미터는 odi.odi명.framworkurl, odi.framworkurl, connection.framworkurl 입니다.

※ 주의사항 : Servlet API를 이용하여 보고서에 데이터를 표시할 경우에는 오즈 리포트 디자이너에서는 미리보기할 수 없으며, 서버 리파지토리에 보고서를 등록한 후 웹 페이지를 통해서만 미리보기할 수 있습니다.

Step 1 ozsdmapi.jar 파일을 WAS에 등록하기

Servlet API 연동 모듈인 ozsdmapi.jar 파일을 WAS의 라이브러리에 등록합니다. 본 예제에서는 Tomcat 5.0의 webapps\ROOT\WEB-INF\lib 폴더에 ozsdmapi.jar 파일을 복사하였습니다.

Step 2 클래스 파일 등록하기

"예제 1"에서 만든 DataModuleSampleServlet.class 파일을 WAS에 등록합니다.

- DataModuleSampleServlet.class 파일을 WAS의 클래스 폴더에 복사합니다. 본 예제에서는 Tomcat 5.0의 webapps\ROOT\WEB-INF\classes\sample 폴더에 클래스 파일을 복사하였습니다.

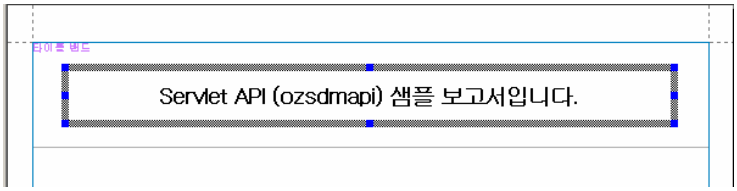
- DataModuleSampleServlet.class 파일을 WAS에 매핑합니다. 본 예제에서는 Tomcat 5.0의 webapps\ROOT\WEB-INF의 web.xml 파일을 편집기로 열어 아래와 같이 편집하여 복사한 클래스 파일을 매핑하였습니다.

```
...

<!-- JSPC servlet mappings start -->
...
<servlet>
    <servlet-name>sample.DataModuleSampleServlet</servlet-name>
    <servlet-class>sample.DataModuleSampleServlet</servlet-class>
</servlet>
...
<servlet-mapping>
      <servlet-name>sample.DataModuleSampleServlet</servlet-name>
      <url-pattern>/sample.DataModuleSampleServlet</url-pattern>
</servlet-mapping>
...
<!-- JSPC servlet mappings end -->
```

Step 3 보고서 만들기

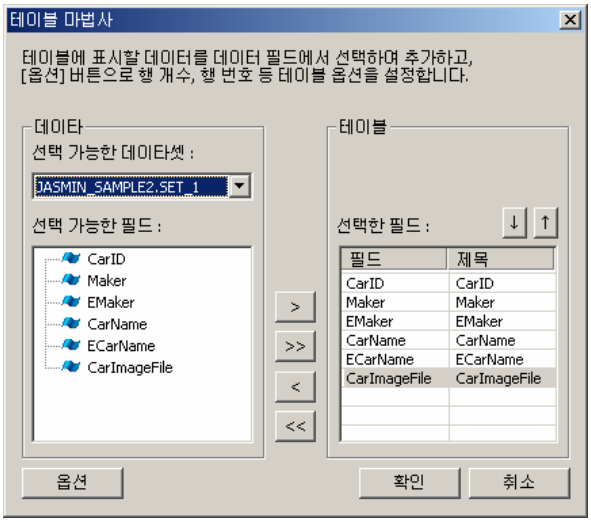
- 오즈 리포트 디자이너를 실행하여 보고서를 새로 만든 후 "예제 1"에서 만든 JASMIN_SAMPLE2.odi 파일을 추가합니다.

- 보고서에 타이틀 밴드를 추가하고, 타이틀 밴드에 라벨을 추가하여 보고서 타이틀을 만듭니다.

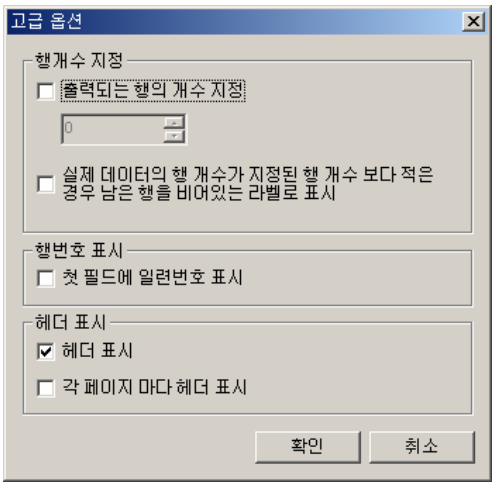


- 보고서에 데이터 밴드를 추가한 후 데이터 밴드의 속성 중 "ODI 이름" 속성값을 "JASMIN_SAMPLE2"로, "데이터셋" 속성값을 "SET_1"으로 설정합니다.

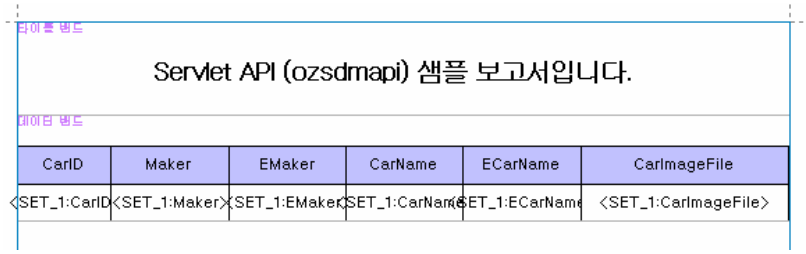
- 툴바에서 테이블 아이콘을 선택한 후 데이터 밴드에 원하는 테이블 크기만큼 드래그&드롭하면 자동으로 테이블 마법사가 실행됩니다. 테이블 마법사에서 테이블에 표시할 필드를 추가합니다.



➢ 테이블의 헤더를 표시하기 위해 테이블 마법사의 [옵션] 버튼을 클릭하여 나타나는 테이블 옵션 다이얼로그에서 "헤더 표시" 옵션을 체크한 후 [확인] 버튼을 클릭하여 옵션을 적용합니다.



➢ 옵션 다이얼로그가 닫힌 후에 테이블 마법사의 [확인] 버튼을 클릭하여 데이터 밴드에 테이블을 추가한 후 각 셀의 너비, 글꼴, 배경색 등을 디자인합니다.



➢ 디자인된 보고서를 "Sample_ServletAPI.ozr" 파일로 저장합니다.

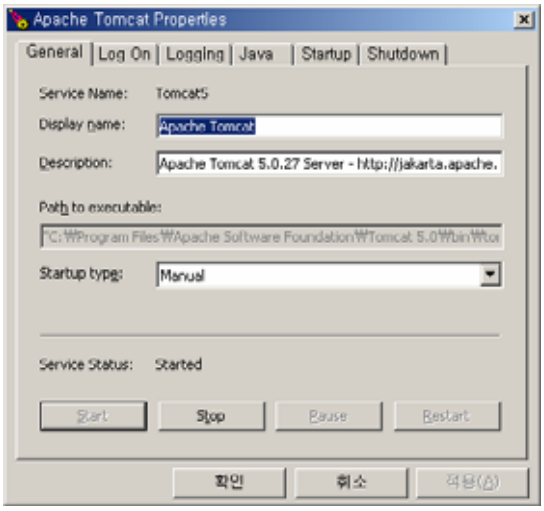
Step 4 보고서 미리보기

➢ 작성된 보고서를 웹에서 호출하는 웹 페이지를 작성합니다. 본 예제에서는 아래와 같이 입력한 후 웹 서버의 루트 경로에 Sample_ServletAPI.htm 파일로 저장하였습니다.

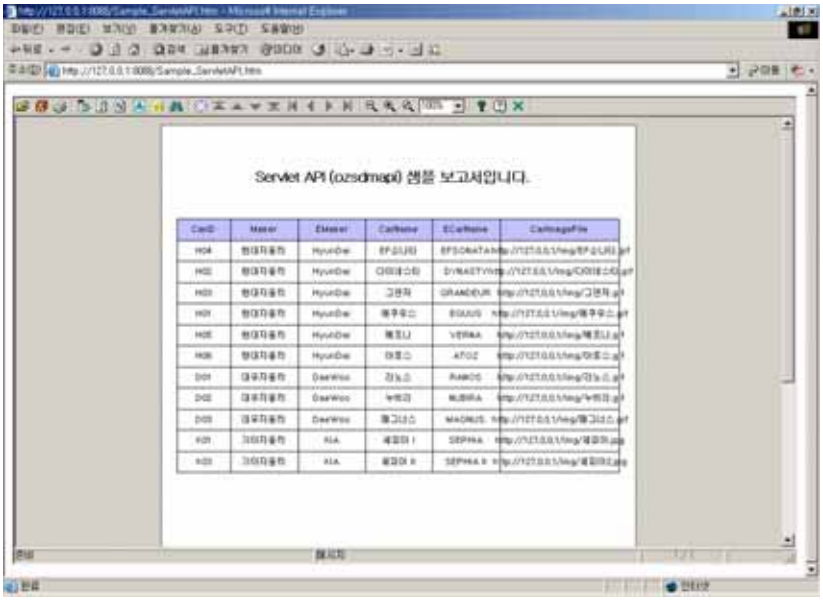
```
<HTML> <BODY>

<OBJECT width = "0" height = "0" ID="ZTransferX" CLASSID="CLSID:C7C7225A-9476-47AC-B0B0-FF3B79D55E67"
codebase="127.0.0.1:8088/ozrviewer/ZTransferX.cab#version=2,0,1,2">
    <PARAM NAME="download.Server" VALUE="http://127.0.0.1/ozrviewer">
    <PARAM NAME="download.Port" VALUE="8088">
    <PARAM NAME="download.Instruction" VALUE="ozrviewer.idf">
    <PARAM NAME="install.Base" VALUE="<PROGRAMS>/Forcs">
    <PARAM NAME="install.Namespace" VALUE="Sample_ServletAPI">
</OBJECT>

<OBJECT id = "ozrviewer" CLASSID="CLSID:0DEF32F8-170F-46f8-B1FF-4BF7443F5F25" width="100%" height="100%">
    <param name="connection.servlet" value="http://127.0.0.1:8088/OZServlet40/server">
    <param name="connection.reportname" value="Sample_ServletAPI.ozr">
    <param name="odi.odinames" value="JASMIN_SAMPLE2">
    <param name="odi.JASMIN_SAMPLE2.frameworkurl" value="http://127.0.0.1:8088/sample.DataModuleSampleServlet">
    <param name="odi.JASMIN_SAMPLE2.fetchtype" value="BATCH">
    <param name="viewer.isframe" value="false">
    <param name="viewer.namespace" value="Sample_ServletAPI₩ozviewer">
  </OBJECT>
</BODY> </HTML>
```

※ 참고사항 : Servlet API를 이용하여 데이터를 바인딩할 경우에는 뷰어 패러미터 중 odi.odi명.framworkurl 또는 odi.framworkurl 또는 connection.framworkurl 패러미터 값에 연동할 클래스 파일에 대한 URL 경로를 반드시 설정하여야 합니다.

➢ 작성된 보고서를 오즈 서블릿 서버의 리파지토리 경로에 복사한 후 웹 서버와 오즈 서블릿 서버를 구동시킵니다.



➢ 웹 브라우저를 실행하여 보고서 파일을 호출하는 웹 페이지를 실행하여 보고서를 미리보기합니다.



예제 4 : XML 형태의 SDM 생성

다음은 Servlet API를 이용하여 XML 형태의 SDM을 생성하는 예제를 살펴보겠습니다.
Servlet API를 이용하여 XML 형태의 SDM을 생성하기 위해서는 먼저 ozsdmapi.jar, crimson.jar 파일과 XML 형태의 SDM을 생성하는 클래스 파일을 WAS에 등록한 후 UDS 스토어를 이용한 ODI 파일을 만들어야 합니다.

Step 1 ozsdmapi.jar과 crimson.jar 파일을 WAS에 등록하기

Servlet API 연동 모듈인 ozsdmapi.jar, crimson.jar 파일을 WAS의 라이브러리에 등록합니다. 본 예제에서는 Tomcat 5.0의 webapps\ROOT\WEB-INF\lib 폴더에 ozsdmapi.jar, crimson.jar 파일을 복사하였습니다.

Step 2 XML 형태의 SDM을 생성하는 클래스 파일 만들기

XML 형태의 SDM을 생성하는 클래스 파일을 만듭니다. 본 예제에서는 아래와 같은 java 소스 파일을 이용하여 DebugDMSampleServlet.class 파일로 만들었습니다.

```
package sample;

import java.io.*;
import javax.servlet.*;

import javax.servlet.http.*;
import java.sql.*;
import java.util.*;

import oz.framework.api.DataModule;
import oz.sdm.DataModuleFactory;

public class DebugDMSampleServlet extends HttpServlet {
	private static final String _KEY_ODI_FETCH_TYPE = "_OZ_ODIFetchType_";
	private static final String _KEY_ODIITEM = "_OZ_ODIITEM_";
	private static final String _KEY_OZ_DATASET_ = "_OZ_DATASET_";
	private static final String _KEY_OZ_DEBUG_ = "_OZ_DEBUG_";

	private static final String ODI_NAME_1 = "JASMIN_SAMPLE2.odi";
	private static final String ODI_NAME_2 = "JASMIN_SAMPLE3.odi";
	private static final String ODI_NAME_3 = "JASMIN_SAMPLE4.odi";

	private Connection m_conn = null;

	public void init(ServletConfig config) throws ServletException {
		super.init(config);
		System.out.println("init...");
	}

	public void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
		process(request, response);
	}
```

```
	public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
		process(request, response);
	}

	public void destroy() {
		super.destroy();
		System.out.println("destroy...");
	}

	private void process(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
		try {
			String _URL = "jdbc:odbc:ozdemokr30";
			Properties prop = new Properties();
			prop.put("user", "");
			prop.put("password", "");
			Driver driver =
(Driver)Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
			m_conn = driver.connect(_URL, prop);
			System.out.println("connection ok...");
		} catch(Exception e) {
			System.out.println("connection error...");
			System.out.println(e.getMessage());
			DataModule.sendErrorMessage("connection error...",
response.getOutputStream());
			close(m_conn, null, null);
			throw new ServletException(e.getMessage());
		}

		try {
			System.out.println("start--------------------");
			Enumeration enum = request.getParameterNames();

			System.out.println("parameter list--------------------");
			while (enum.hasMoreElements()) {
				String temp = (String) enum.nextElement();
				System.out.println("name=" + temp + " value=" +
getEncode(request.getParameter(temp)));
			}
			System.out.println("----------------------------------------");

			String odiFetchType =
getEncode(request.getParameter(_KEY_ODI_FETCH_TYPE));
			String item = getEncode(request.getParameter(_KEY_ODIITEM));
			String dataset = getEncode(request.getParameter(_KEY_OZ_DATASET_));

			if (odiFetchType == null) {
				odiFetchType = "DM_CONCURRENT_FETCH";
			}

			if ((dataset == null) || (dataset.length() == 0)) {
				if (item.equals(ODI_NAME_2)) {
					Master_Detail_Style(request, response, odiFetchType);
				} else if (item.equals(ODI_NAME_1) || item.equals(ODI_NAME_3)) {
					default_Style(request, response, odiFetchType);
				} else {
					System.out.println("unsupported odi error...");
					DataModule.sendErrorMessage("unsupported odi error...",
response.getOutputStream());
					throw new ServletException("Unknown Item :" + item);
				}
			} else {
				if (odiFetchType.equalsIgnoreCase("DM_CONCURRENT_FETCH") ||
odiFetchType.equalsIgnoreCase("DM_PER_DATASET")) {
```

```
                                setDefaultStyle(request, response, odiFetchType);
                                //setResultSetQueryStyle(request, response, odiFetchType);
                        } else {
                                System.out.println("unsupported error...");
                                DataModule.sendErrorMessage("unsupported error...",
response.getOutputStream());
                                throw new ServletException("unsupported");
                        }
                }
        } catch(Exception ex){
                System.out.println(ex.getMessage());
                DataModule.sendErrorMessage(ex.getMessage(), response.getOutputStream());
                throw new ServletException(ex.getMessage());
        } finally {
                close(m_conn, null, null);
                System.out.println("end--------------------");
        }
    }

    private void Master_Detail_Style(HttpServletRequest request, HttpServletResponse response,
String fetchType) throws ServletException, IOException {
        Statement stmt1 = null;
        Statement stmt4 = null;
        ResultSet rs1 = null;
        ResultSet rs4 = null;
        DataModule module = null;

        String[] _FN1 = {"FirstName"};
        int[] _FT1 = {java.sql.Types.VARCHAR};

        String[] _FN4 = {"ContactId", "FirstName", "LastName", "Phone", "Fax", "Email",
"OrgUnitId", "UserName" };
        int[] _FT4 = {4, 12, 12, 12, 12, 12, 5, 12 };
        try {
                module = DataModuleFactory.getDataModule(fetchType);

                // debug 모드로 실행하기 위해서 반드시 아래의 값이 true 로 세팅되어야 한다.
                // init 전에 setDebug 메소드가 호출되어야 한다.
                module.setDebug(parseBoolean(request), response);
                module.init(response.getOutputStream());

                module.addSetInfo("SET_1", "", _FN1, _FT1);
                module.addSetInfo("SET_4", "SET_1", _FN4, _FT4);
        } catch(Exception ex) {
                throw new ServletException(ex.getMessage());
        }

        try {
                module.startBinding();
                String query1 = "select distinct FirstName from contact";
                module.startSet("SET_1");

                stmt1 = m_conn.createStatement();
                rs1 = stmt1.executeQuery(query1);
                while(rs1.next()) {
                        String f1 = rs1.getString(_FN1[0]);
                        HashMap map1 = new HashMap();
                        map1.put(_FN1[0], f1);
                        module.addRow("SET_1", map1);

                        String query4 = "select * from contact where FirstName='" + f1 + "'order by
contactid";
                        module.startSet("SET_4");
                        stmt4 = m_conn.createStatement();
                        rs4 = stmt4.executeQuery(query4);
```

```
                        while(rs4.next()) {
                            HashMap map4 = new HashMap();
                            map4.put(_FN4[0], new Integer(rs4.getInt(_FN4[0])));
                            map4.put(_FN4[1], rs4.getString(_FN4[1]));
                            map4.put(_FN4[2], rs4.getString(_FN4[2]));
                            map4.put(_FN4[3], rs4.getString(_FN4[3]));
                            map4.put(_FN4[4], rs4.getString(_FN4[4]));
                            map4.put(_FN4[5], rs4.getString(_FN4[5]));
                            map4.put(_FN4[6], new Integer(rs4.getInt(_FN4[6])));
                            map4.put(_FN4[7], rs4.getString(_FN4[7]));
                            module.addRow("SET_4", map4);
                        }
                        rs4.close();
                        stmt4.close();
                        module.endSet("SET_4");
                    }
                    module.endSet("SET_1");
                    module.endBinding();
                } catch(Exception e) {
                    System.out.println(e.getMessage());
                    if(module != null) {
                        module.sendBindErrorMessage(e.toString());
                    }
                } finally {
                    close(null, stmt1, rs1);
                    close(null, stmt4, rs4);
                }
            }

            private void default_Style(HttpServletRequest request, HttpServletResponse response, String
fetchType) throws ServletException, IOException {
                System.out.println("select time=="+System.currentTimeMillis());
                Statement stmt = null;
                ResultSet rs = null;
                DataModule module = null;

                String[] _FN = {"CarID", "Maker", "EMaker", "CarName", "ECarName", "CarImageFile"};
                int[] _FT = {java.sql.Types.VARCHAR, java.sql.Types.VARCHAR,
java.sql.Types.VARCHAR, java.sql.Types.VARCHAR, java.sql.Types.VARCHAR,
java.sql.Types.VARCHAR};

                try {
                    module = DataModuleFactory.getDataModule(fetchType);

                    // debug 모드로 실행하기 위해서 반드시 아래의 값이 true 로 세팅되어야 한다.
                    // init 전에 setDebug 메소드가 호출되어야 한다.
                    module.setDebug(parseBoolean(request), response);
                    module.init(response.getOutputStream());
                    module.addParameter("PARAM1", java.sql.Types.VARCHAR,
getEncode(request.getParameter("PARAM1")));
                    module.addParameter("PARAM2", java.sql.Types.VARCHAR,
getEncode(request.getParameter("PARAM2")));
                    module.addSetInfo("SET_1", "", _FN, _FT);

                }catch(Exception ex){
                    throw new ServletException(ex.getMessage());
                }

                try {
                    module.startBinding();
                    String query = "select * from car";
                    module.startSet("SET_1");

                    stmt = m_conn.createStatement();
                    rs = stmt.executeQuery(query);
```

```
                while(rs.next()) {
                        HashMap map = new HashMap();
                        String a = rs.getString(_FN[0]);
                        String b = rs.getString(_FN[1]);
                        String c = rs.getString(_FN[2]);
                        String d = rs.getString(_FN[3]);
                        String e = rs.getString(_FN[4]);
                        String f = rs.getString(_FN[5]);

                        map.put(_FN[0], a);
                        map.put(_FN[1], b);
                        map.put(_FN[2], c);
                        map.put(_FN[3], d);
                        map.put(_FN[4], e);
                        map.put(_FN[5], f);

                        module.addRow("SET_1", map);

                        System.out.print("select FN0="+_FN[0]+" name="+a);
                        System.out.print(" FN1="+_FN[1]+" name="+b);
                        System.out.print(" FN2="+_FN[2]+" name="+c);
                        System.out.print(" FN3="+_FN[3]+" name="+d);
                        System.out.print(" FN4="+_FN[4]+" name="+e);
                        System.out.print(" FN5="+_FN[5]+" name="+f);
                        System.out.println("");
                }

                module.endSet("SET_1");
                module.endBinding();
            } catch(Exception e) {
                System.out.println(e.getMessage());
                module.sendBindErrorMessage(e.toString());
            } finally {
                close(null, stmt, rs);
            }
        }

        private void setDefaultStyle(HttpServletRequest request, HttpServletResponse response, String
fetchType) throws ServletException, IOException {
            System.out.println("1setDefaultStyle fetchType =="+fetchType);
            Statement stmt = null;
            ResultSet rs = null;
            DataModule module = null;

            String[] _FN = {"CarID", "Maker", "EMaker", "CarName", "ECarName", "CarImageFile"};
            int[] _FT = {java.sql.Types.VARCHAR, java.sql.Types.VARCHAR,
java.sql.Types.VARCHAR, java.sql.Types.VARCHAR, java.sql.Types.VARCHAR,
java.sql.Types.VARCHAR};
            try {
                module = DataModuleFactory.getDataModule(fetchType);
                // debug 모드로 실행하기 위해서 반드시 아래의 값이 true 로 세팅되어야 한다.
                // init 전에 setDebug 메소드가 호출되어야 한다.
                module.setDebug(parseBoolean(request), response);
                module.init(response.getOutputStream());
                module.addSetInfo("SET_1", _FN, _FT);
            }catch(Exception ex){
                throw new ServletException(ex.getMessage());
            }

            try {
                module.startBinding();
                String query = "select * from car";

                module.startSet("SET_1");
                stmt = m_conn.createStatement();
```

```
                rs = stmt.executeQuery(query);
                while(rs.next()) {
                        HashMap map = new HashMap();
                        String a = rs.getString(_FN[0]);
                        String b = rs.getString(_FN[1]);
                        String c = rs.getString(_FN[2]);
                        String d = rs.getString(_FN[3]);
                        String e = rs.getString(_FN[4]);
                        String f = rs.getString(_FN[5]);

                        map.put(_FN[0], a);
                        map.put(_FN[1], b);
                        map.put(_FN[2], c);
                        map.put(_FN[3], d);
                        map.put(_FN[4], e);
                        map.put(_FN[5], f);

                        module.addRow("SET_1", map);

                        System.out.print("select FN0="+_FN[0]+" name="+a);
                        System.out.print(" FN1="+_FN[1]+" name="+b);
                        System.out.print(" FN2="+_FN[2]+" name="+c);
                        System.out.print(" FN3="+_FN[3]+" name="+d);
                        System.out.print(" FN4="+_FN[4]+" name="+e);
                        System.out.print(" FN5="+_FN[5]+" name="+f);
                        System.out.println("");
                }

                module.endSet("SET_1");
                module.endBinding();
        } catch(Exception e) {
                System.out.println(e.getMessage());
                module.sendBindErrorMessage(e.toString());
        } finally {
                close(null, stmt, rs);
        }
    }

    private void setResultSetQueryStyle(HttpServletRequest request, HttpServletResponse
response, String fetchType) throws ServletException, IOException {
        System.out.println("setResultSetQueryStyle...");
        Statement stmt = null;
        ResultSet rs = null;
        DataModule module = null;

        try {
                module = DataModuleFactory.getDataModule(fetchType);
                // debug 모드로 실행하기 위해서 반드시 아래의 값이 true 로 세팅되어야 한다.
                // init 전에 setDebug 메소드가 호출되어야 한다.
                module.setDebug(parseBoolean(request), response);
                String query = "select * from car";
                stmt = m_conn.createStatement();
                rs = stmt.executeQuery(query);

                module.makeSDM_SET("SET_1", rs, response.getOutputStream());
        } catch(Exception e) {
                System.out.println(e.getMessage());
                throw new ServletException(e.getMessage());
        } finally {
                close(null, stmt, rs);
        }
    }

    private String getEncode(String value) {
        try {
```

```
                return new String(value.getBytes("8859_1"), "KSC5601");
            }catch(Exception ex){
                return value;
            }
        }

        private void close(Connection con, Statement stmt, ResultSet rs) {
            try {
                if(rs != null) {
                    rs.close();
                }

                if(stmt != null) {
                    stmt.close();
                }

                if(con != null) {
                    con.close();
                }
            } catch(SQLException se) {}
        }

        private boolean parseBoolean(HttpServletRequest request) {
            String tmp = (String)request.getParameter(_KEY_OZ_DEBUG_);
            if("true".equalsIgnoreCase(tmp)) {
                return true;
            } else {
                return false;
            }
        }
}
```

Step 3 클래스 파일 등록하기

DebugDMSampleServlet.class 파일을 WAS에 등록합니다.

- DebugDMSampleServlet.class 파일을 WAS의 클래스 폴더에 복사합니다. 본 예제에서는 Tomcat 5.0의 webapps\ROOT\WEB-INF\classes\sample 폴더에 클래스 파일을 복사하였습니다.

- DebugDMSampleServlet.class 파일을 WAS에 매핑합니다. 본 예제에서는 Tomcat 5.0의 webapps\ROOT\WEB-INF의 web.xml 파일을 편집기로 열어 아래와 같이 편집하여 복사한 클래스 파일을 매핑하였습니다.

```
...

<!-- JSPC servlet mappings start -->
...
<servlet>
    <servlet-name>sample.DebugDMSampleServlet </servlet-name>
    <servlet-class>sample.DebugDMSampleServlet</servlet-class>
</servlet>
...
<servlet-mapping>
```

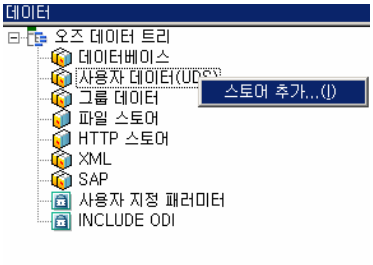
```
        <servlet-name>sample.DebugDMSampleServlet </servlet-name>
        <url-pattern>/sample.DebugDMSampleServlet </url-pattern>
</servlet-mapping>
...
<!-- JSPC servlet mappings end -->
```

Step 4 ODI 만들기

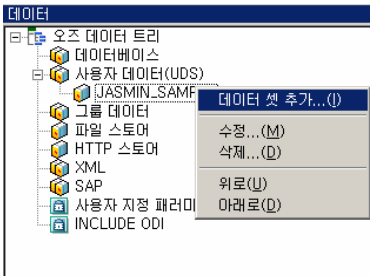
Servlet API를 이용할 경우에는 사용자 데이터(UDS) 스토어를 이용하고, 데이터셋과 데이터 필드를 추가하여야 합니다.
데이터셋과 데이터 필드를 추가할 때에는 수동으로 하나씩 추가하는 방법과 POST 패러미터 중 "_OZ_ODIITEM_"을 이용하여 자동으로 추가하는 방법이 있습니다.

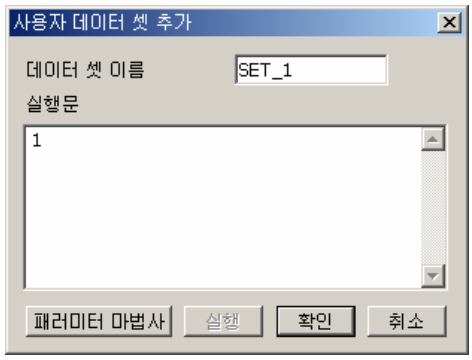
먼저 수동으로 데이터셋과 데이터 필드를 추가하는 방법은 다음과 같습니다.

- 오즈 쿼리 디자이너를 실행하여 아래와 같이 사용자 데이터(UDS) 스토어를 추가합니다.

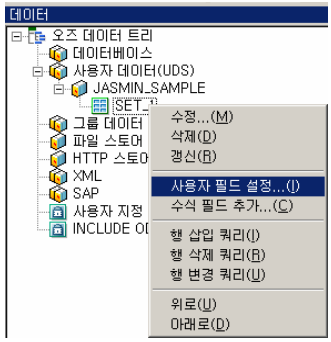


- 사용자 데이터 스토어에 데이터셋을 추가합니다.

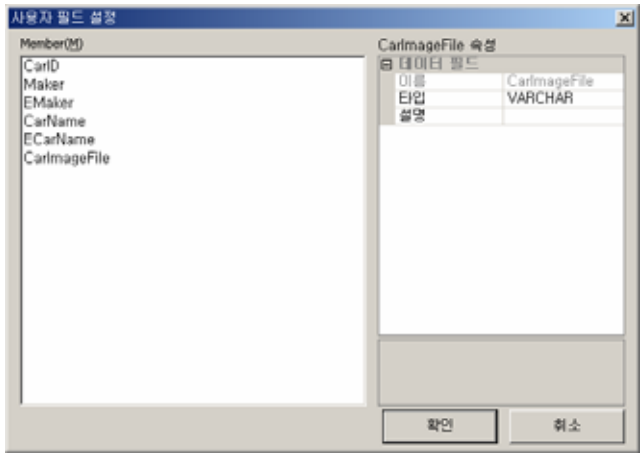




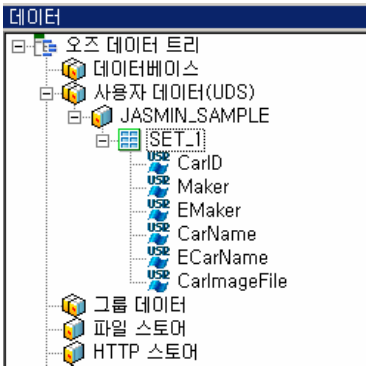
➢ 추가된 데이터셋에 사용자 필드를 추가합니다. 추가된 데이터셋을 마우스 오른쪽 버튼으로 클릭하여 나타나는 팝업 메뉴에서 [사용자 필드 설정] 메뉴를 클릭합니다.



➢ 사용자 필드 설정 다이얼로그에서 아래 그림처럼 사용자 필드를 추가한 후 [확인] 버튼을 누릅니다.

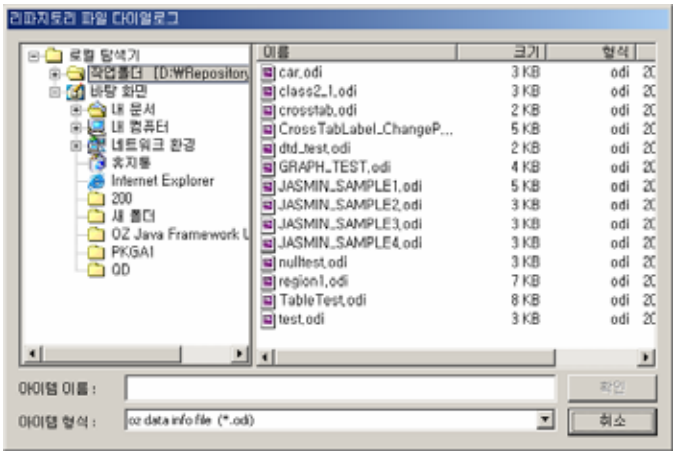


오즈 데이터 트리에 사용자 필드가 아래 그림처럼 추가되었는지 확인합니다.

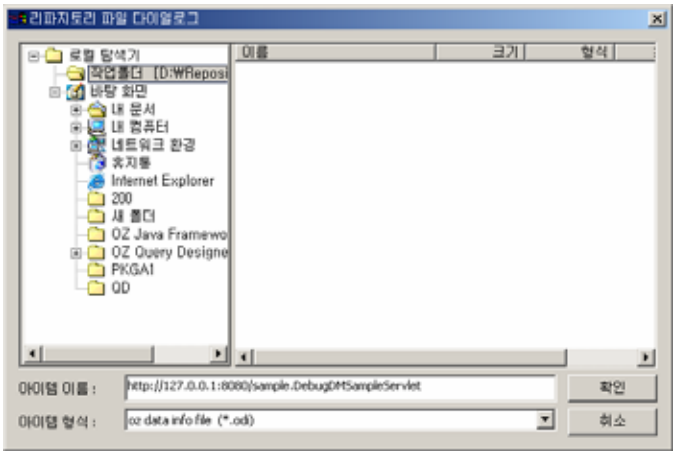


다음으로 자동으로 데이터셋과 데이터 필드를 추가하는 방법은 다음과 같습니다.

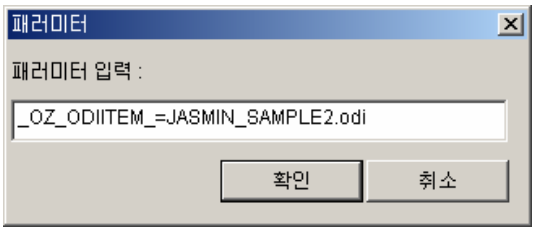
- 오즈 쿼리 디자이너를 실행한 후 툴바에서 열기( ) 아이콘을 클릭하여 리파지토리 열기 다이얼로그를 띄웁니다.



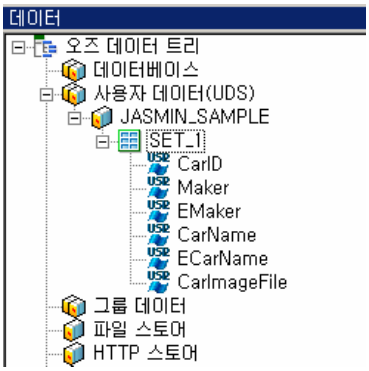
- 아이템 이름에 클래스 파일의 URL을 입력한 후 [확인] 버튼을 클릭합니다.



➢ 패러미터 입력 다이얼로그에 "_OZ_ODIITEM_=ODI이름"을 입력한 후 [확인] 버튼을 클릭합니다.



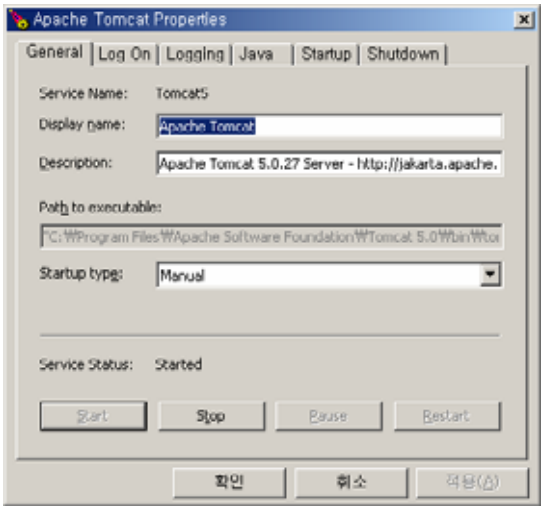
➢ 데이터 트리에 데이터셋과 데이터 필드가 추가되어 표시됩니다.



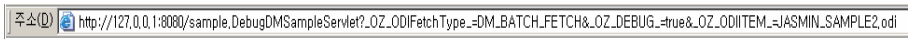
➢ 수동 또는 자동 방식으로 만들어진 ODI 파일을 "JASMIN_SAMPLE2.odi" 파일로 저장합니다.

Step 5 웹에서 XML 형태의 SDM 생성

➢ 웹 서버를 구동시킵니다.

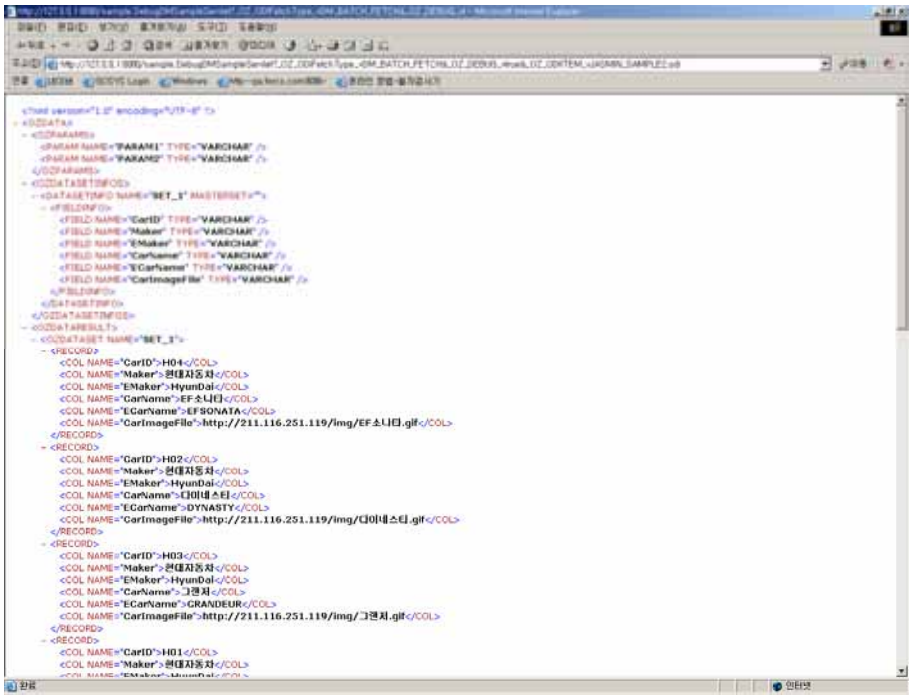


➢ 웹 브라우저에서 XML 형태의 SDM을 출력시키는 URL을 입력하여 웹 페이지를 실행시킵니다.



http://127.0.0.1:8080/sample.DebugDMSampleServlet?_OZ_ODIFetchType_=DM_BATCH_FETCH&_OZ_DEBUG_=true&_OZ_ODIITEM_=JASMIN_SAMPLE2.odi

➢ 웹 브라우저에서 XML 형태의 SDM을 출력합니다.



예제 5 : 오즈 자바 서버와 FXDataModule 연동

다음은 Servlet API를 이용하여 오즈 애플리케이션 FX Data Accesses의 "_OZData_" 데이터 모듈에서 마스터 디테일 셋을 추가하고 Table에 데이터를 표시하고 표시된 데이터를 삽입, 삭제, 변경하는 예를 들어 설명하겠습니다.

해당 예제의 데이터를 전달하는 방법은 압축 XML 형태이고 Servlet API를 이용하여 데이터를 검색하기 위해서는 먼저 ozsdmapi.jar, crimson.jar 파일이 WAS에 등록하여야 합니다.

Step 1 ozsdmapi.jar과 crimson.jar 파일을 WAS에 등록하기

Servlet API 연동 모듈인 ozsdmapi.jar, crimson.jar 파일을 WAS의 라이브러리에 등록합니다. 본 예제에서는 Tomcat 5.0의 webapps\ROOT\WEB-INF\lib 폴더에 ozsdmapi.jar, crimson.jar 파일을 복사하였습니다.

Step 2 FXDataModule 연동 클래스 파일 만들기

FXDataModule과 연동하는 클래스 파일을 만듭니다. 본 예제에서는 아래와 같은 java 소스 파일을 이용하여 Test.class 파일로 만들었습니다.

```
package sample;

import java.io.IOException;
import java.sql.Connection;
import java.sql.Driver;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Properties;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import oz.fxapi.core.ConvertException;
import oz.fxapi.core.DataActionParser;
import oz.fxapi.dm.FX_DataAction;
import oz.fxapi.dm.FX_DataFieldMeta;
import oz.fxapi.dm.FX_DataModule;
import oz.fxapi.dm.FX_DataSet;
import oz.fxapi.dm.FX_DataSetMeta;
import oz.fxapi.dm.FX_DataTypes;
import oz.fxapi.dm.FX_Parameter;
import oz.fxapi.dm.FX_Record;
import oz.fxapi.dm.FX_UnmatchingMetaException;
import oz.util.OZSQL;
import oz.util.OZString;

 /** <p>Title: Framework test module</p>
 * <p>Description: Tests framework functionality </p>
 * <p>Copyright: Copyright (c) 2006 by FORCS All rights reserved </p>
 * <p>Company: FORCS Co.,LTD.</p>
```

```
 * @author kgn
 */
public class Test extends HttpServlet
{
	private static final long serialVersionUID = 8673861145524516336L;

	public void doPost(HttpServletRequest request, HttpServletResponse response)
		throws ServletException, IOException
	{
		process(request, response);
	}

	public void doGet(HttpServletRequest request, HttpServletResponse response)
		throws ServletException, IOException
	{
		process(request, response);
	}

	private void process(HttpServletRequest request,
		HttpServletResponse response) throws ServletException, IOException
	{
		String type = request.getParameter("type");
		boolean isCompressed =
"true".equalsIgnoreCase(request.getParameter("compressed"));

		if (OZString.isNullOrEmpty(type))
			throw new ServletException("Operation type not specified.");

		try
		{
			if ("sdm".equalsIgnoreCase(type))
				processSDM(request, response, true, isCompressed);
			else if ("xml".equalsIgnoreCase(type))
				processSDM(request, response, false, isCompressed);
			else if ("dac".equalsIgnoreCase(type))
				processDAC(request, response);
			else
				throw new ServletException("Unknown operation code; " + type);
		} catch (Exception e)
		{
			e.printStackTrace();
			throw new ServletException(e.getMessage());
		}
	}

	/** Generate data module which has master-detail relation
	 * @param request
	 * @param response
	 * @param isSDM
	 * @throws Exception
	 */
	private void processSDM(HttpServletRequest request,
		HttpServletResponse response, boolean isSDM, boolean isCompressed) throws
Exception
	{
		FX_DataModule dm = new FX_DataModule();

		// Access does not support multiple ResultSet
		Connection con = null;
		Connection con2 = null;

		try
		{
			String url = "jdbc:odbc:ozcar";
			Properties prop = new Properties();
```

```
                Driver driver = (Driver) Class.forName(
                        "sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
                con = driver.connect(url, prop);
                con2 = driver.connect(url, prop);
            } catch (Exception e)
            {
                dm.sendErrorMessage(response.getOutputStream(), isSDM, isCompressed, e
                        .getMessage());
                OZSQL.close(con);
                OZSQL.close(con2);
                throw new ServletException(e.getMessage());
            }

            response.setContentType(isSDM || isCompressed ? "application/octet-stream" :
"text/xml");

            FX_DataSetMeta fxMeta = new FX_DataSetMeta("IDs");
            fxMeta.addDataFieldMeta(new FX_DataFieldMeta("CarID",
                    FX_DataTypes.FX_DT_VARCHAR));
            dm.addDataSetMeta(fxMeta);

            fxMeta = new FX_DataSetMeta("Informations");
            fxMeta.addDataFieldMeta(new FX_DataFieldMeta("Maker",
                    FX_DataTypes.FX_DT_VARCHAR));
            fxMeta.addDataFieldMeta(new FX_DataFieldMeta("CarName",
                    FX_DataTypes.FX_DT_VARCHAR));
            fxMeta.addDataFieldMeta(new FX_DataFieldMeta("ECarName",
                    FX_DataTypes.FX_DT_VARCHAR));
            fxMeta.setMasterSetName("IDs");
            dm.addDataSetMeta(fxMeta);

            Statement stmt = con.createStatement();
            Statement stmt2 = con2.createStatement();
            String id;

            ResultSet master = stmt.executeQuery("Select CarID from car");
            try
            {
                FX_DataSet masterSet = new FX_DataSet("IDs");
                while (master.next())
                {
                    FX_Record record = new FX_Record();
                    id = master.getString(1);
                    record.addColumn("CarID", id);

                    if (!OZString.isNullOrEmpty(id))
                    {
                        ResultSet detail = stmt2
                                .executeQuery("Select Maker, CarName, ECarName from
car where CarID = '"
                                        + id + "'");
                        try
                        {
                            FX_DataSet detailSet = new FX_DataSet("Informations");
                            while (detail.next())
                            {
                                FX_Record detailRecord = new FX_Record();
                                detailRecord
                                        .addColumn("Maker", detail.getString(1));
                                detailRecord.addColumn("CarName", detail
                                        .getString(2));
                                detailRecord.addColumn("ECarName", detail
                                        .getString(3));
                                detailSet.addRecord(detailRecord);
```

```
                                }
                                record.addDetailSet(detailSet);
                        } finally
                        {
                                detail.close();
                        }
                }

                masterSet.addRecord(record);
        }

        dm.addDataSet(masterSet);

    } catch (Exception e)
    {
        dm.sendErrorMessage(response.getOutputStream(), isSDM, isCompressed, e
                .getMessage());
        throw e;
    } finally
    {
        master.close();
        stmt.close();
        stmt2.close();
        OZSQL.close(con);
        OZSQL.close(con2);
    }

    try
    {
        String error = request.getParameter("error");
        if(!OZString.isNullOrEmpty(error))
        {
            throw new FX_UnmatchingMetaException(error);
        }
        dm.write(response.getOutputStream(), isSDM, isCompressed);
    }
    catch(FX_UnmatchingMetaException ue)
    {
        dm.sendErrorMessage(response.getOutputStream(), isSDM, isCompressed, ue
                .getMessage());
    }
    catch(ConvertException ce)
    {
        dm.sendErrorMessage(response.getOutputStream(), isSDM, isCompressed, ce
                .getMessage());
    }
    catch(IOException e)
    {
        // ignore
    }
}

private void processDAC(HttpServletRequest request,
        HttpServletResponse response) throws Exception
{
    Connection con = null;

    try
    {
        String url = "jdbc:odbc:OZCar";
        Properties prop = new Properties();
        prop.put("user", "");
        prop.put("password", "");

        Driver driver = (Driver) Class.forName(
```

```
                        "sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
                con = driver.connect(url, prop);
        } catch (Exception e)
        {
                OZSQL.close(con);
                throw new ServletException(e.getMessage());
        }

        FX_DataModule dm = new FX_DataModule();
        FX_DataAction[] dacs = dm.getDataAction(request);

        for (int i = 0; i < dacs.length; i++)
        {
                FX_DataAction dac = dacs[i];
                String type = dac.getActionType();
                if ("insert".equalsIgnoreCase(type))
                {
                        // do insert
                        insert(dac, con);
                } else if ("delete".equalsIgnoreCase(type))
                {
                        // do delete
                        delete(dac, con);
                } else if ("rowupdate".equalsIgnoreCase(type))
                {
                        // to update
                        update(dac, con);
                } else
                {
                        throw new Exception("Illegal data action type; " + type);
                }
        }
}

private String m_insert, m_update, m_delete;

private static final String TABLENAME = "car";

private PreparedStatement m_insertStmt, m_deleteStmt, m_updateStmt;

private void insert(FX_DataAction dac, Connection con) throws Exception
{
        FX_Parameter[] srcFields = dac.getSourceFields();

        if (OZString.isNullOrEmpty(m_insert))
        {
                // 맨 처음 호출한 경우 쿼리문을 만들어서 prepared화 시킨다.
                m_insert = "INSERT INTO " + TABLENAME + " (";
                for (int i = 0; i < srcFields.length; i++)
                {
                        if (i == srcFields.length - 1)
                        {
                                m_insert += srcFields[i].getName();
                        } else
                        {
                                m_insert += srcFields[i].getName() + ",";
                        }
                }
                m_insert += ") VALUES ( ";

                for (int i = 0; i < srcFields.length; i++)
                {
                        if (i == srcFields.length - 1)
                        {
                                m_insert += "?";
```

```
                    } else
                    {
                        m_insert += "? ,";
                    }
                }

                m_insert += ")";
                m_insertStmt = con.prepareStatement(m_insert);
            }

            for (int i = 0; i < srcFields.length; i++)
            {
                m_insertStmt.setString(i + 1, encode(srcFields[i].getValue()));
            }

            m_insertStmt.execute();

        }

        private void update(FX_DataAction dac, Connection con) throws Exception
        {
            FX_Parameter[] srcFields = dac.getSourceFields();
            FX_Parameter[] destFields = dac.getTargetFields();

            if (OZString.isNullOrEmpty(m_update))
            {
                // 맨 처음 호출한 경우 쿼리문을 만들어서 prepared화 시킨다.

                m_update = "UPDATE " + TABLENAME + " set ";

                for (int i = 0; i < srcFields.length; i++)
                {
                    if (i == srcFields.length - 1)
                    {
                        m_update += srcFields[i].getName() + " = ? ";
                    } else
                    {
                        m_update += srcFields[i].getName() + " = ?, ";
                    }
                }
            m_update += " WHERE ";

                for (int i = 0; i < destFields.length; i++)
                {
                    if (i == destFields.length - 1)
                    {
                        m_update += destFields[i].getName() + " = ? ";
                    } else
                    {
                        m_update += destFields[i].getName() + " = ? AND ";
                    }
                }

                m_updateStmt = con.prepareStatement(m_update);
            }

            int i = 0;
            for (i = 0; i < srcFields.length; i++)
            {
                m_updateStmt.setString(i + 1, encode(srcFields[i].getValue()));
            }

            for (int k = 0; k < destFields.length; k++)
            {
                m_updateStmt.setString(i + 1, encode(destFields[k].getValue()));
```

```
                i++;
            }
            m_updateStmt.execute();
        }

        private void delete(FX_DataAction dac, Connection con) throws Exception
        {
            FX_Parameter[] targetFields = dac.getTargetFields();

            if (OZString.isNullOrEmpty(m_delete))
            {
                m_delete = "DELETE FROM " + TABLENAME + " WHERE ";

                for (int i = 0; i < targetFields.length; i++)
                {
                    if (i == targetFields.length - 1)
                    {
                        m_delete += targetFields[i].getName() + " = ? ";
                    } else
                    {
                        m_delete += targetFields[i].getName() + " = ? AND ";
                    }
                }

                m_deleteStmt = con.prepareStatement(m_delete);
            }

            int i = 0;
            for (i = 0; i < targetFields.length; i++)
            {
m_deleteStmt.setString(i + 1, encode(targetFields[i].getValue()));
            }


            m_deleteStmt.execute();

        }

        private String encode(String value)
        {
            try
            {
                return new String(value.getBytes("8859_1"), "KSC5601");
            }catch(Exception ex)
            {
              return value;
            }
        }
}
```

Step 3 **클래스 파일 등록하기**

Test.class 파일을 WAS에 등록합니다.

- Test.class 파일을 웹 서버의 클래스 폴더에 복사합니다. 본 예제에서는 Tomcat 5.0의 webapps\ROOT\WEB-INF\classes 폴더에 sample 폴더를 새로 만든 후 클래스 파일을 복사하였습니다.

- Test.class 파일을 WAS에 매핑합니다. 본 예제에서는 Tomcat 5.0의

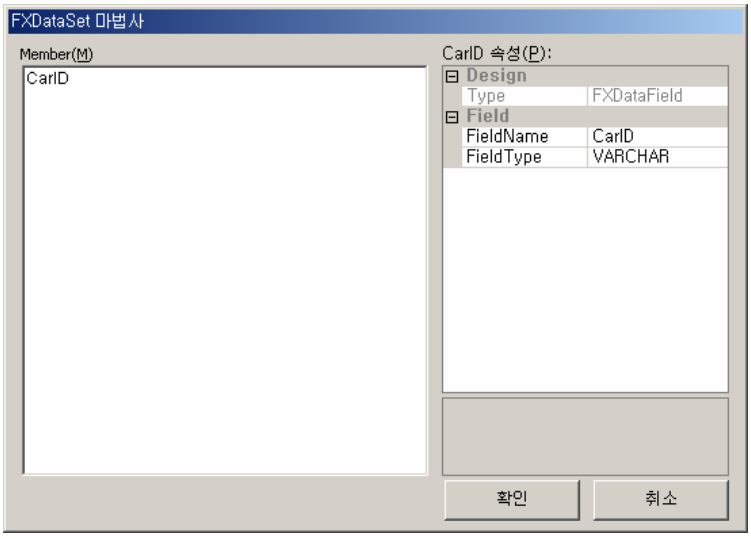
webapps\ROOT\WEB-INF의 web.xml 파일을 편집기로 열어 아래와 같이 편집하여 복사한 클래스 파일을 매핑하였습니다.

```
...

<!-- JSPC servlet mappings start -->
...
<servlet>
      <servlet-name>sample.Test</servlet-name>
      <servlet-class>sample.Test</servlet-class>
</servlet>
...
<servlet-mapping>
         <servlet-name>sample.Test </servlet-name>
         <url-pattern>/Test </url-pattern>
</servlet-mapping>
...
<!-- JSPC servlet mappings end -->
```

Step 4 데이터셋 추가하기

- FX Data Accesses의 "_OZData_"을 마우스 오른쪽 버튼으로 선택하여 나타나는 팝업 메뉴에서 [새 FX DataSet] 메뉴를 클릭하여 데이터셋 추가 다이얼로그를 띄운 후 마스터 데이터셋의 이름을 입력하고 [확인] 버튼을 클릭하여 마스터 데이터셋을 추가합니다.

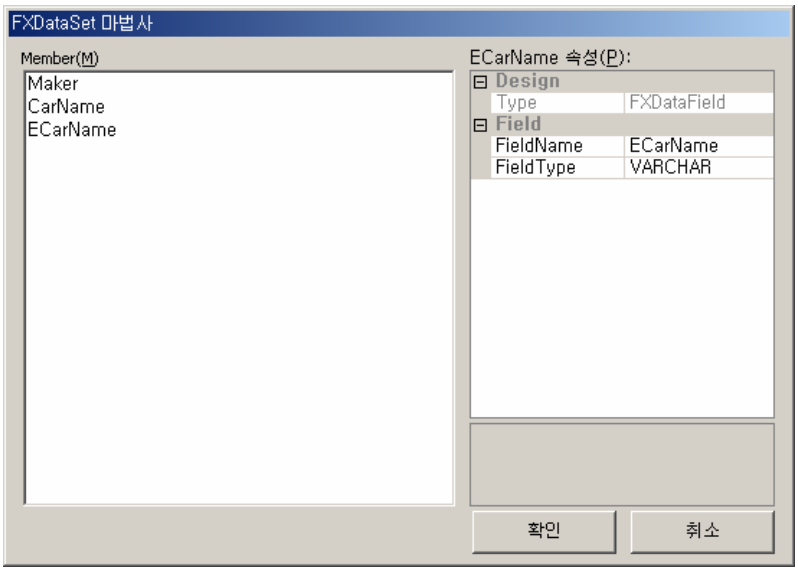
- 추가한 데이터셋을 마우스 오른쪽 버튼으로 선택하여 나타나는 팝업 메뉴에서 [새 필드 추가] 메뉴를 클릭하여 데이터셋의 필드를 추가합니다.



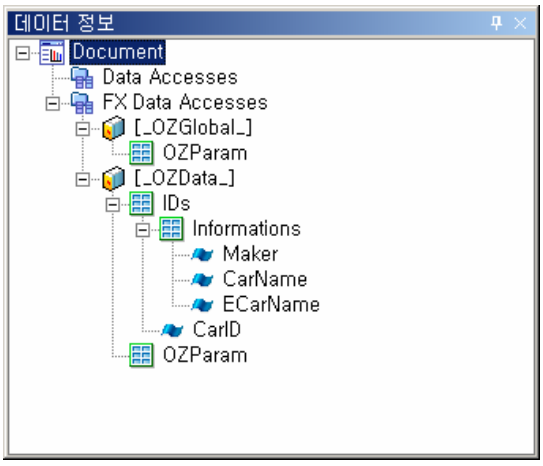
- 추가한 마스터 데이터셋을 마우스 오른쪽 버튼으로 선택하여 나타나는 팝업 메뉴에서 [새 Detail FX DataSet] 메뉴를 클릭하여 데이터셋 추가 다이얼로그를 띄운 후 디테일 데이터셋의 이름을 입력하고 [확인] 버튼을 클릭하여 디테일 데이터셋

을 추가합니다.

➢ 추가한 디테일 데이터셋을 마우스 오른쪽 버튼으로 선택하여 나타나는 팝업 메뉴에서 [새 필드 추가] 메뉴를 클릭하여 데이터셋의 필드를 추가합니다.



➢ 아래 그림과 같은 마스터-디테일 데이터셋이 추가되었음을 확인합니다.



Step 5 OZA 만들기

Board에 Panel, Label, TextBox, Button, Table을 추가하고 데이터 셋 연결, 레이아웃 등을 디자인합니다.

➢ Label을 추가한 후 각 Lable의 "Text" 속성값을 "CarID", "Maker", "CarName", "ECarName"으로 변경합니다.

- Button을 4개 추가한 후 각 Button의 "Text" 속성값을 "XML로 저장", "Insert", "Update", "Delete"로 변경합니다.

- 정보를 표시할 TextBox를 추가한 후 위치(Location), 크기(Size)등을 변경한 후 TextBox의 데이터 연동과 관련한 속성값을 변경합니다.
  TextBox1의 속성 중 "ODIKey"를 "_OZData_"로, "DataSet"을 "IDs"로, "Field"를 "CarID"로 변경하고 "TextBox2", "TextBox3", "TextBox4"의 "ODIKey"를 "_OZData_"로, "DataSet"을 "Informations"으로 변경한 후 "Field"를 각각 "Maker", "CarName", "ECarName"으로 변경합니다.

- 정보를 표시할 Table 2개를 추가한 후 Table1의 "ODIKey"를 "_OZData_"로, "DataSet"을 "IDs"로, "FireRowCursorChange"를 "True"로 변경하고 Table2의 "ODIKey"를 "_OZData_"로, "DataSet"을 "Informations"로 변경합니다.

- Table1의 마법사를 실행하여 "CarID" 필드를 추가하고, Table2의 마법사를 실행하여 "Maker", "CarName", "ECarName" 필드를 추가합니다.

Step 5 스크립트 입력

정보를 XML로 전달하고 DataAction을 위한 스크립트를 입력합니다.

- [XML로 전달] 버튼의 "OnClick" 이벤트를 선택하여 데이터가 XML 형태로 압축 전송되도록 스크립트를 입력합니다.

```
_GetFXDataModule().RemoveAllDataSet();
var xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");

// XML, 압축전송
xmlhttp.Open("POST", "http://127.0.0.1:8080/Test?type=xml", false);
xmlhttp.Send("");
var dm = _GetFXDataModule();
dm.ApplyData(xmlhttp.responseStream);
if(dm.FXErrorMessage != ""){
	_MessageBox(dm.FXErrorMessage);
}
```

- [Insert] 버튼의 "OnClick" 이벤트를 선택하여 "CarID: K04"에 대한 정보가 삽입되도록 스크립트를 입력합니다.

```
_GetFXDataModule().RemoveAllDataSet();
```

```
var xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");

// XML, 압축전송
xmlhttp.Open("POST",
http://127.0.0.1:8080/Test?type=dac&_OZ_DAC_CNT_=1&0.TYPE=insert&0.SRC_CNT=4&0.S
F_0=CarID&0.SFT_0=VARCHAR&0.SV_0=K04&0.SF_1=Maker&0.SFT_1=VARCHAR&0.SV_1=기
아자동차&0.SF_2=CarName&0.SFT_2=VARCHAR&0.SV_2=세피아&0.SF_3=ECarName&0.SFT_3
=VARCHAR&0.SV_3=SEPHIA, false);
xmlhttp.Send("");
var dm = _GetFXDataModule();
dm.ApplyData(xmlhttp.responseStream);
if(dm.FXErrorMessage != ""){
	_MessageBox(dm.FXErrorMessage);
}
```

- [Update] 버튼의 'OnClick' 이벤트를 선택하여 "CarID: K04"에 대한 정보가 수정되도록 스크립트를 입력합니다.

```
_GetFXDataModule().RemoveAllDataSet();
var xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");

// XML, 압축전송
xmlhttp.Open("POST",
"http://127.0.0.1:8080/Test?type=dac&_OZ_DAC_CNT_=1&0.TYPE=rowupdate&0.SRC_CNT=1
&0.SF_0=ECarName&0.SFT_0=VARCHAR&0.SV_0=SEPHIA2&0.TRG_CNT=1&0.DF_0=CarID&0.D
FT_0=VARCHAR&0.DV_0=K04", false);
xmlhttp.Send("");
var dm = _GetFXDataModule();
dm.ApplyData(xmlhttp.responseStream);
if(dm.FXErrorMessage != ""){
	_MessageBox(dm.FXErrorMessage);
}
```

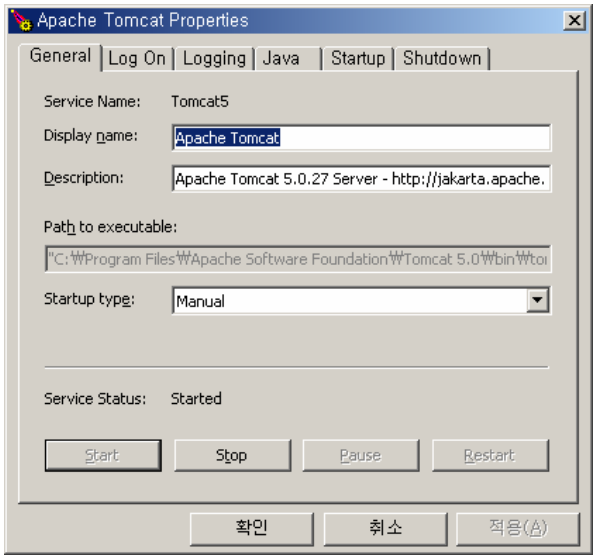
- [Delete] 버튼의 "OnClick" 이벤트를 선택하여 "CarID: K04"에 대한 정보가 삭제되도록 스크립트를 입력합니다.

```
_GetFXDataModule().RemoveAllDataSet();
var xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");

// XML, 압축전송
xmlhttp.Open("POST",
http://127.0.0.1:8080/Test?type=dac&_OZ_DAC_CNT_=1&0.TYPE=delete&0.TRG_CNT=1&0.D
F_0=CarID&0.DFT_0=VARCHAR&0.DV_0=K04, false);
xmlhttp.Send("");
var dm = _GetFXDataModule();
```

```
dm.ApplyData(xmlhttp.responseStream);
if(dm.FXErrorMessage != ""){
        _MessageBox(dm.FXErrorMessage);
}
```

Step 6 미리보기

➢ WAS를 구동시킵니다.



➢ [File] 메뉴의 [Preview]를 클릭하거나 툴바의 미리보기 아이콘( )을 클릭하여 미리보기합니다.