DB 가이드

Last update: 2024/01/22

www.softbase.co.kr

목차

DB 가이드

이 화면은 DB 컴포넌트에 대한 예시화면이다.

DB 컴포넌트는 전용브라우저 환경에서만 동작하며, 데이터베이스와의 인터페이스 기능을 제공한다.

전용브라우저 버전(32/64비트) 비트수에 맞춘, 접속 대상 DBMS에 해당하는 ODBC 드라이버 S/W를 설치해야 한다.

관련 속성으로 module_path가 있다.

기본 API로는 init, commit, rollback, connect, disconnect, executesql, getresultrecordcount, getresultfieldcount, getresultrecordfetch,

환경 설정 관련 API로 getautocommit, setautocommit, isconnected가 있다.

예시

템플릿 위치: /HTML5/COMPONENT/DB/db_basic

템플릿 파일

- db_basic.xml
- db_basic.js
- 새창으로 실행

화면 스크립트

```
// 화면 로드 이벤트
function screen_on_load()
{
    // DB 컴포넌트 초기화
    this.ctrlDB.init();
}

// DB 연결/해제 버튼 이벤트
function btnConnect_on_mouseup(objInst)
{
  // 연결이 되지 않은 경우, 연결 시도
    if (this.ctrlDB.isconnected() == false) {
        this.fnOpenDatabae();
    }
  // 이미 DB가 연결되어 있는 경우, 연결 해제 처리
    else {
        // DB 연결 해제
```

```
        this.ctrlDB.disconnect();
    }

    this.ShowOpenDatabaeStatus();
}

// DB 연결 결과 정보 설정
function ShowAutoCommitStatus()
{
  // 연결 상태 확인
    if (this.ctrlDB.isconnected() == false) { return; }

    if (this.ctrlDB.getautocommit() == true) {
        this.chkAutoCommit.setcheck(true);
        this.btnCommit.setenable(false);
        this.btnRollback.setenable(false);
    }
    else {
        this.chkAutoCommit.setcheck(false);
        this.btnCommit.setenable(true);
        this.btnRollback.setenable(true);
    }
}

// DB 연결 결과 정보 설정
function ShowOpenDatabaeStatus()
{
    if (this.ctrlDB.isconnected()) {
        // DB 연결 버튼 텍스트 설정 및 DB 연결 정보 패널 활성화
        this.btnConnect.settext("Disconnect");
        this.pnlConnect.setenable(true);
    }
    else {
        // DB 연결 버튼 텍스트 설정 및 DB 연결 정보 패널 비활성화
        this.btnConnect.settext("Connect");
        this.pnlConnect.setenable(true);
    }

    this.ShowAutoCommitStatus();
}

/**
 * DB 연결 함수
 */
function fnOpenDatabae()
{
    var nDBKind = this.cbDBKind.getselectedcode();
    var strDBName = this.fieldDBname.gettext();
    var strServiceName = this.fieldServiceName.gettext();
    var strIPAddress = this.fieldIPAddress.gettext();
    var nPort = Number(this.fieldPort.gettext());
```

```
    var strUID = this.fieldUID.gettext();
    var strPWD = this.fieldPWD.gettext();

  // 연결 상태 확안
    if (this.ctrlDB.isconnected() == true) { return true; }

    // DB 연결 수행
    var bConnect = this.ctrlDB.connect(nDBKind, strDBName, strServiceName,
strIPAddress, nPort, strUID, strPWD);

    return bConnect;
}

// 쿼리 실행 버튼 이벤트
function btnExecQuery_on_mouseup(objInst)
{
    var ret, strQuery, nFieldCount, nFieldIndex, nRecordCount, strValue,
nRowIndex, nColumnIndex;

  // 그리드 내용 및 컬럼 삭제
    this.gridResult.deleteall();
    this.gridResult.deleteallcolumn();

    if (this.ctrlDB.isconnected() == false && this.fnOpenDatabae() == false)
{
        screen.alert("DB가 연결되어 있지 않습니다.");
        return;
    }

    strQuery = this.mmQuery.gettext();
    if (strQuery.length <= 0) {
        screen.alert("쿼리를 입력하세요.");
        return;
    }

  // 쿼리 실행
    if (this.ctrlDB.executesql(strQuery) == false) {
        screen.alert("쿼리 수행에 실패하였습니다.");
        return;
    }

  // 쿼리 결과 레코드 갯수를 구함
    nRecordCount = this.ctrlDB.getresultrecordcount();
    factory.consoleprint("nRecordCount : " + nRecordCount);
    if (nRecordCount <= 0) {
        screen.alert("쿼리 실행 결과 레코드 갯수 = " + nRecordCount);
        return;
    }

  // 쿼리 결과 필드 갯수를 구함 (Select 쿼리가 아닌 경우에는, 필드 갯수가 0임)
    nFieldCount = this.ctrlDB.getresultfieldcount();
    if (nFieldCount <= 0) {
```

```
        screen.alert("쿼리 실행 결과 레코드 갯수 = " + nRecordCount);
        return;
    }

    factory.consoleprint("nFieldCount : " + nFieldCount);

  // 쿼리 결과 필드 갯수만큼 그리드 컬럼 생성
    for (nFieldIndex = 0; nFieldIndex < nFieldCount; nFieldIndex++) {
        nColumnIndex = this.gridResult.addcolumn(false);
        if (nColumnIndex < 0) {
            screen.alert("그리드 컬럼 추가 실패하였습니다.");
            continue;
        }

    // 필드 이름을 구하여 컬럼 헤더 텍스트 설정
        this.gridResult.setheadertext(0, nColumnIndex,
this.ctrlDB.getresultfieldname(nFieldIndex), false);

    // 컬럼 데이터 형식을 한글로 설정
        this.gridResult.setcolumndatatype(nColumnIndex, 2, false);
    }

  // 쿼리 결과 한 레코드를 패치
    nResult = this.ctrlDB.getresultrecordfetch();
    factory.consoleprint("Result record fetch : " + nResult);

  // 쿼리 결과 한 레코드를 패치가 성공인 경우일 동안 Loop
    while (nResult == 1) {
    // 그리드에 행 추가
        nRowIndex = this.gridResult.additem(false, false);

    // 레코드에서 컬럼 인덱스에 대한 값을 구하여 그리드 아이템에 설정
        for(nColumnIndex = 0; nColumnIndex < nFieldCount; nColumnIndex++) {
            strValue = this.ctrlDB.getfieldvalue(nColumnIndex);
            this.gridResult.setitemtextex(nRowIndex, nColumnIndex, strValue,
false);
        }

    // 쿼리 결과 한 레코드를 패치
        nResult = this.ctrlDB.getresultrecordfetch();
    }

  // 그리드 내용 Refresh
    this.gridResult.refreshcolumn();
}

// "Commit" 버튼 이벤트
function btnCommit_on_mouseup(objInst)
{
  // 연결 상태 확인
    if (this.ctrlDB.isconnected() == false) { return; }
```

```
    // commit 수행
    if (this.ctrlDB.commit() == false) { screen.alert("Commit error"); }
}

// "Rollback" 버튼 이벤트
function btnRollback_on_mouseup(objInst)
{
  // 연결 상태 확인
    if (this.ctrlDB.isconnected() == false) { return; }

    // rollback 수행
    if (this.ctrlDB.rollback() == false) { screen.alert("Rollback error"); }
}

// "Auto Commit" 체크박스 이벤트
function chkAutoCommit_on_click(objInst)
{
    if (this.chkAutoCommit.getcheck() == true) {
        if (this.ctrlDB.getautocommit() == false) {
            this.ctrlDB.setautocommit(true);
        }
    }
    else {
        if (this.ctrlDB.getautocommit() == true) {
            this.ctrlDB.setautocommit(false);
        }
    }

    this.ShowAutoCommitStatus();
}
```