Using Layout Painter Extensions

You can place external tools into a dedicated area of the Layout Painter and get access to the XML layout that is currently edited. Thus, you can build editor extensions which typically generate a certain part of an XML layout which can then be added to the XML layout that is currently edited in the Layout Painter.

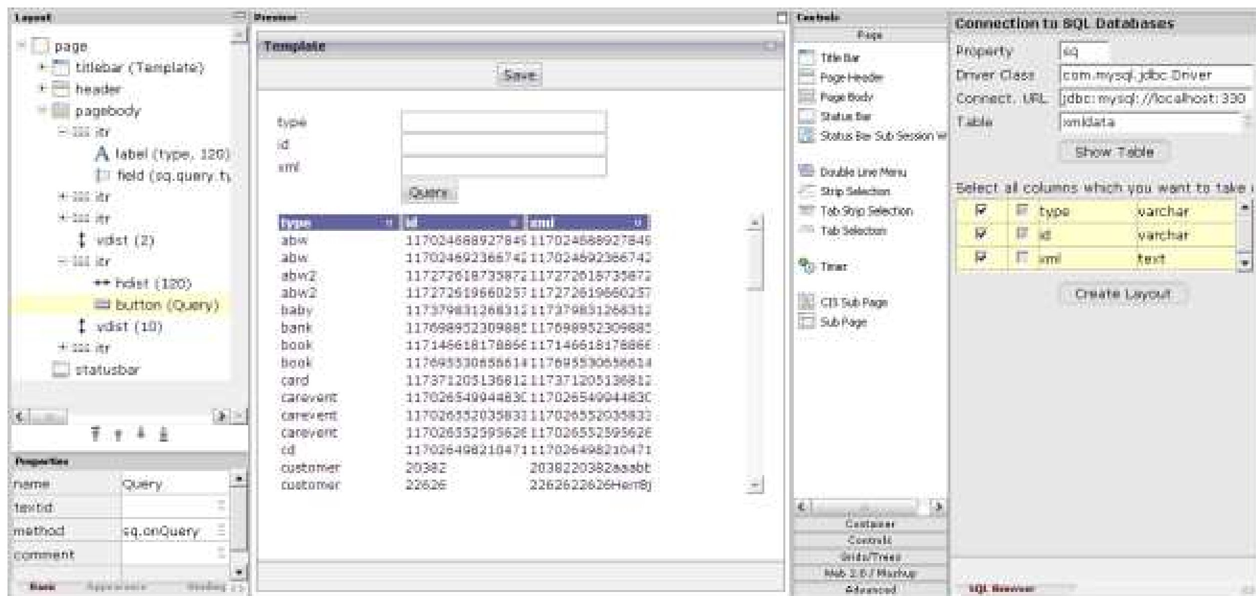
This chapter covers the following topics:

- Example
- Details on the Extension
- Extension Meets Pattern

For information on how to use the Layout Painter, see *Layout Painter* in the *Development Workplace* documentation.

Example

Using extensions, you can easily generate layout elements into an existing layout. This example shows how to build SQL query screens. The result will look as follows:



The right side of the Layout Painter provides an additional extension area in which you can enter database parameters and in which the columns of a selected table are shown. This extension is used in the following way:

- You specify a property name. This property name will later occur in the names of the properties for the generated layout elements.

  Example: if you specify "sq" as the property name, the query field **type** may point, for example, to the property `sq.query.type`.

- You specify the class name and the connection URL of the database to be accessed.

  Note:
  There are also other ways of accessing a database, however, this example concentrates on Layout Painter extension concepts, not on database management.

- The column definitions of the table are shown. You select the columns that are to appear in the layout and choose the **Create Layout** button. As a result, a layout is generated into the page body that reflects a query screen.

- You can manipulate the query screen using the normal edit functionality of the Layout Painter.

Details on the Extension

The definition of an extension is simple:

- You define the extension screen and adapter just as you define a normal Application Designer page.

- You add the interface `IEditorExtension` to the adapter implementation.

- You register the extension in the *cisconfig.xml* file.

In the above example, the layout of the extension is defined in the following way:

```
<page model="com.softwareag.cis.editor.sql.SQLExtensionAdapter">
    <titlebar name="Connection to SQL Databases" withclose="false">
    </titlebar>
    <pagebody horizdist="false" pagebodystyle="background-color: #deebf7" paddingleft="5"
              paddingright="5" paddingtop="5" paddingbottom="5">
        <itr>
            <label name="Property" width="100">
            </label>
            <field valueprop="prefix" width="50">
            </field>
        </itr>
        <itr takefullwidth="true" fixlayout="true">
            <label name="Driver Class" width="100">
            </label>
            <field valueprop="driverClass" width="100%">
            </field>
        </itr>
        <itr takefullwidth="true" fixlayout="true">
            <label name="Connect. URL" width="100">
            </label>
            <field valueprop="connectionURL" width="100%">
            </field>
        </itr>
        <itr takefullwidth="true">
            <label name="Table" width="100">
            </label>
            <field valueprop="table" width="100%" popupmethod="openIdValueComboOrPopup">
            </field>
        </itr>
        <vdist height="5">
        </vdist>
        <itr>
            <hdist width="100">
            </hdist>
            <button name="Show Table" method="onShowTable">
            </button>
        </itr>
        <vdist height="15">
        </vdist>
        <rowdynavis valueprop="tableVisible">
            <itr takefullwidth="true">
                <label name="Select all columns which you want to take over into your layout. Then press &amp;quot;Create Layout&amp;quot;." asplaintext="true">
                </label>
            </itr>
            <rowtablearea2 griddataprop="columns" rowcount="10" vscroll="auto"
                           firstrowcolwidths="true">
                <repeat>
                    <str valueprop="selected" showifempty="false">
                        <checkbox valueprop="selected" flush="server" width="50" align="center">
                        </checkbox>
                        <checkbox valueprop="key" width="30" displayonly="true">
                        </checkbox>
                        <textout valueprop="column" width="50%">
                        </textout>
                        <textout valueprop="type" width="50%">
                        </textout>
                    </str>
                </repeat>
            </rowtablearea2>
            <vdist height="10">
            </vdist>
            <itr>
                <hdist width="100">
```

```
            </hdist>
            <button name="Create Layout" method="onCreateLayout">
            </button>
          </itr>
        </rowdynavis>
    </pagebody>
    <statusbar withdistance="false">
    </statusbar>
</page>
```

The adapter implementation looks as follows:

```
package com.softwareag.cis.editor.sql;

// not shown: ...pacakge import statements...

public class SQLExtensionAdapter
    extends Adapter
    implements IEditorExtension
{
    // ------------------------------------------------------------------------
    // inner classes
    // ------------------------------------------------------------------------

    public class ColumnsItem
    {
        String m_column;
        public String getColumn() { return m_column; }
        public void setColumn(String value) { m_column = value; }

        boolean m_key;
        public boolean getKey() { return m_key; }
        public void setKey(boolean value) { m_key = value; }

        boolean m_selected;
        public boolean getSelected() { return m_selected; }
        public void setSelected(boolean value) { m_selected = value; }

        String m_type;
        public String getType() { return m_type; }
        public void setType(String value) { m_type = value; }
    }

    GRIDCollection m_columns = new GRIDCollection();
    public GRIDCollection getColumns() { return m_columns; }

    String m_connectionURL;
    public String getConnectionURL() { return m_connectionURL; }
    public void setConnectionURL(String value) { m_connectionURL = value; }

    String m_driverClass;
    public String getDriverClass() { return m_driverClass; }
    public void setDriverClass(String value) { m_driverClass = value; }

    String m_prefix;
    public String getPrefix() { return m_prefix; }
    public void setPrefix(String value) { m_prefix = value; }

    String m_table;
    public String getTable() { return m_table; }
    public void setTable(String value) { m_table = value; }

    boolean m_tableVisible = false;
    public boolean getTableVisible() { return m_tableVisible; }

    IExtensionPoint m_extensionPoint;

    // ------------------------------------------------------------------------
    // IEditorExtension
    // ------------------------------------------------------------------------

    public String buildPageURL()
    {
        return "/HTMLBasedGUI/cis.editor.SQLExtension";
    }

    public void init(IExtensionPoint extensionPoint)
    {
        m_extensionPoint = extensionPoint;
    }

    public void reactOnLoadDocument() {}
```

```
public void reactOnUnloadDocument() {}

// ------------------------------------------------------------------------
// public usage
// ------------------------------------------------------------------------

public void init()
{
    m_driverClass = "com.mysql.jdbc.Driver";
    m_connectionURL = "jdbc:mysql://localhost:3306/cispersist?user=root&password=admin";
}

/** */
public ValidValueLine[] findValidValuesForTable()
{
    return SQLUtil.findValidValuesForTable(m_driverClass,m_connectionURL);
}

/**
 * Read table meta data and transfer into columns list.
 */
public void onShowTable()
{
    if (m_prefix == null || m_prefix.trim().length() == 0)
    {
        outputMessage(MT_ERROR,"Please specify property first");
        return;
    }
    if (m_table == null || m_table.trim().length() == 0)
    {
        outputMessage(MT_ERROR,"Please specify table name first");
        return;
    }
    // fetch meta data from SQL and load into columns table
    try
    {
        SQLUtil.ColumnMetaData[] cols = SQLUtil.readMetaDataForTable(m_driverClass,m_connectionURL,m_table);
        m_columns.clear();
        for (int i=0; i<cols.length; i++)
        {
            ColumnsItem ci = new ColumnsItem();
            ci.setColumn(cols[i].m_column);
            ci.setType(cols[i].m_type);
            ci.setKey(cols[i].m_isKey);
            m_columns.add(ci);
        }
        m_tableVisible = true;
    }
    catch (Throwable t)
    {
        outputMessage(MT_ERROR,t.toString());
    }
}

/**
 * Create the layout out of the column definition and transfer the layout
 * into the page's XML.
 */
public void onCreateLayout()
{
    List selColumns = new ArrayList();
    Iterator iter = m_columns.iterator();
    while (iter.hasNext())
    {
        ColumnsItem ci = (ColumnsItem)iter.next();
        if (ci.getSelected() == true)
            selColumns.add(ci);
    }
    if (selColumns.size() == 0)
    {
        outputMessage(MT_ERROR,"Please select all columns for which layout elements should be generated");
        return;
    }
    // generate XML layout definition
    StringBuffer sb = new StringBuffer();
    iter = selColumns.iterator();
    while (iter.hasNext())
    {
        ColumnsItem ci = (ColumnsItem)iter.next();
        sb.append("<itr>");
```

```
            sb.append("<label name='"+ci.getColumn()+"' width='120'/>");
            sb.append("<field valueprop='"+m_prefix+".query."+ci.getColumn()+"' width='200'/>");
            sb.append("</itr>");
        }
        sb.append("<vdist height='2'/>");
        sb.append("<itr>");
        sb.append(  "<hdist width='120'/>");
        sb.append(  "<button name='Query' method='"+m_prefix+".onQuery'/>");
        sb.append("</itr>");
        sb.append("<vdist height='10'/>");
        sb.append("<itr takefullwidth='true'>");
        sb.append(  "<textgridsss2 width='100%' rowcount='15' griddataprop='"+m_prefix+".lines'>");
        iter = selColumns.iterator();
        while (iter.hasNext())
        {
            ColumnsItem ci = (ColumnsItem)iter.next();
            sb.append("<column name='"+ci.getColumn()+"' property='"+ci.getColumn()+"' width='120'/>");
        }
        sb.append(  "</textgridsss2>");
        sb.append("</itr>");
        // pass created XML into the page
        if (m_extensionPoint != null)
        {
            String layoutXML = m_extensionPoint.findXMLLayout();
            layoutXML = StringMgmt.replaceInString("</pagebody>",sb.toString()+"</pagebody>",layoutXML);
            m_extensionPoint.updateXMLLayout(layoutXML);
        }
    }

}
```

The building blocks of the code are:

- The adapter implements the interface `IEditorExtension` and therefore needs to implement the following methods:
    - **buildPageURL**
      This method tells the extension framework the name of the extension's HTML page. The extension is registered at the server side. Therefore, the Layout Painter needs to know which page it has to embed in the extension area.
    - **init**
      This method is called by the Layout Painter. The most important parameter is the `IExtensionPoint` which is an abstraction of the Layout Painter environment. It allows access to the inner parts of the Layout Painter such as the XML layout definition.
    - **reactOnLoadDocument and reactOnUnloadDocument**
      These methods are called when the user opens or closes a layout definition inside the editor.
- The adapter provides a method `onShowTable()` which accesses the database via an `SQLUtil` class and fills the grid containing information on each column.
- The adapter provides a method `onCreateLayout()`. This method creates a layout which is then passed into the page's layout. For the generation of the layout, the following rules apply:
    - For each column that is selected, a query line is generated. The query line is an ITR line holding a LABEL and a FIELD definition.
    - One TEXTGRIDSSS2 definition is created. For each selected column, one COLUMN definition inside the grid is created.

The `SQLUtil` class provides generic functions for accessing the database:

```
package com.softwareag.cis.editor.sql;

import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;

import com.softwareag.cis.server.IDynamicAccess;
import com.softwareag.cis.server.ServerLog;
import com.softwareag.cis.server.util.ValidValueLine;
import com.softwareag.cis.editor.sql.SQLExtensionAdapter.ColumnsItem;

/**
 * Collection of static functions which manage the interface
 * to the database.
 */
public class SQLUtil
{
    // ------------------------------------------------------------------------
    // inner classes
    // ------------------------------------------------------------------------

    public static class ColumnMetaData
    {
        public boolean m_isKey;
        public String m_column;
        public String m_type;
    }

    // ------------------------------------------------------------------------
    // public usage
    // ------------------------------------------------------------------------

    /**
     */
    public static Connection createConnection(String driverClassName,
                                              String connectionURL)
        throws Exception
    {
        Class.forName(driverClassName).newInstance();
        Connection conn = DriverManager.getConnection(connectionURL);
        return conn;
    }

    /**
     */
    public static ColumnMetaData[] readMetaDataForTable(String driverClassName,
                                                        String connectionURL,
                                                        String table)
    {
        try
        {
            List columns = new ArrayList();
```

```
            Connection conn = createConnection(driverClassName,connectionURL);
            DatabaseMetaData dbmd = conn.getMetaData();
            // get primary key and column info
            Set primaryKeys = new HashSet();
            ResultSet rs = dbmd.getPrimaryKeys(null,null,table);
            while (rs.next())
                primaryKeys.add(rs.getString("COLUMN_NAME"));
            rs = dbmd.getColumns(null,null,table,null);
            while (rs.next())
            {
                ColumnMetaData cmd = new ColumnMetaData();
                cmd.m_column = rs.getString("COLUMN_NAME");
                cmd.m_type = rs.getString("TYPE_NAME");
                if (primaryKeys.contains(cmd.m_column))
                    cmd.m_isKey = true;
                columns.add(cmd);
            }
            conn.close();
            ColumnMetaData[] result = new ColumnMetaData[columns.size()];
            columns.toArray(result);
            return result;
        }
        catch (Throwable t)
        {
            throw new Error(t);
        }
    }

    /**
     */
    public static ValidValueLine[] findValidValuesForTable(String driverClassName,
                                                          String connectionURL)
    {
        try
        {
            Connection conn = createConnection(driverClassName,connectionURL);
            DatabaseMetaData dbmd = conn.getMetaData();
            // get primary key and column info
            List vvs = new ArrayList();
            ResultSet rs = dbmd.getTables(null,null,null,null);
            while (rs.next())
            {
                ValidValueLine vv = new ValidValueLine(rs.getString("TABLE_NAME"));
                vvs.add(vv);
            }
            conn.close();
            ValidValueLine[] result = new ValidValueLine[vvs.size()];
            vvs.toArray(result);
            return result;
        }
        catch (Throwable t)
        {
            ServerLog.appendException(t);
            return new ValidValueLine[0];
        }
    }

    /**
     */
    public static List executeQuery(String driverClassName,
                                    String connectionURL,
                                    String sql,
```

```
                                    String[] columns)
    {
        try
        {
            List resultList = new ArrayList();
            Connection conn = createConnection(driverClassName,connectionURL);
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(sql);
            while (rs.next())
            {
                Map columnMap = new HashMap();
                for (int i=0; i<columns.length; i++)
                {
                    String column = columns[i];
                    String value = rs.getString(column);
                    columnMap.put(column,value);
                }
                resultList.add(columnMap);
            }
            conn.close();
            return resultList;
        }
        catch (Throwable t)
        {
            return new ArrayList();
        }
    }

}
```

Note:
The functions for table processing are a bit limited as they deal only with string-type columns, however, this is a demo use case only.

Finally, let us have a look at the *cisconfig.xml* file in which all extensions are registered. For this example, we have inserted a new editor extension named "SQL Browser".

```
<cisconfig startmonitoringthread="true"
            requestclienthost="false"
            debugmode="false"
            loglevel="EWI"
            logtoscreen="true"
            sessiontimeout="3600"
            xmldatamanager="com.softwareag.cis.xmldata.filebased.XMLDataManager"
            useownclassloader="true"
            browserpopuponerror="false"
            framebuffersize="3"
            onlinehelpmanager="com.softwareag.cis.onlinehelp.projectbased.FrameHelpOHManager"
            textencoding="UTF-8"
            enableadapterpreload="true"
            animatecontrols="true">

    <requestrecording recordrequests="false"
                      recorddirectory="c:/temp/traces/">
    </requestrecording>

    <editorextensions>
           <editorextension name="WSDL Browser"
                  classname="com.softwareag.cis.editor.extension.wsdlpage.WSDLPageAdapter">
           </editorextension>
           <editorextension name="SQL Browser"
                  classname="com.softwareag.cis.editor.sql.SQLExtensionAdapter">
           </editorextension>
```

```
        <editorextension name="Map Converter" classname="PluginMapCreatorAdapter">
        </editorextension>
    </editorextensions>

    <generationaddons>
        <generationaddon classname="com.softwareag.cis.gui.generate.XSDGenerationAddon">
        </generationaddon>
    </generationaddons>

</cisconfig>
```

In the section `editorextensions`, the class names of all editor extensions are listed. The class needs to provide a constructor without parameters and needs to implement the interface `IEditorExtension`.

Extension Meets Pattern

A Layout Painter extension generates a certain XML layout which is taken over into the page. It makes sense to generate the layout in such a way that it meets a processing pattern within the adapter of the generated page.

In the above SQL example, the controls that are generated by the extension are binding to properties in the following way:

- There is a general property definition that is input into the extension. Let us assume that the user specifies "sq" as the general property name.
- All query fields are binding to a property definition `valueprop="sq.query.`*`columnName`*`"`.
- The TEXTGRIDSSS2 binds to `griddataprop="sq.lines"` and each COLUMN definition inside the girds binds to `property="`*`columnName`*`"`.

Let us have a look at the adapter code of the page that is generated inside the Layout Painter:

```
// This class is a generated one.

import java.util.*;
import com.softwareag.cis.server.*;
import com.softwareag.cis.server.util.*;
import com.softwareag.cis.util.*;
import com.softwareag.cis.editor.sql.SQLQueryMgr;

public class SSQQLL1Adapter
    extends Adapter
{
    SQLQueryMgr m_sq = new SQLQueryMgr(
            "com.mysql.jdbc.Driver",
            "jdbc:mysql://localhost:3306/cispersist?user=root&password=admin",
            "xmldata");
    public SQLQueryMgr getSq() { return m_sq; }
}
```

This is not much code. The main coding is done in the `SQLQueryMgr` class. This pattern class can be used throughout various adapters to provide the ".query" and ".lines" data and processing. Let us have a look at this class:

```
package com.softwareag.cis.editor.sql;

import java.sql.Connection;
import java.sql.DriverManager;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

import com.softwareag.cis.file.CSVManager;
import com.softwareag.cis.server.IDynamicAccess;
import com.softwareag.cis.server.util.TEXTGRIDCollection;

public class SQLQueryMgr
{
    // ------------------------------------------------------------------------
    // inner classes
    // ------------------------------------------------------------------------

    /**
     * This is the object that holds all the query parameters. Each column
     * is represented as one property which is reachable by IDynamicAccess.
     */
    public class TableRow
        implements IDynamicAccess
    {
        Map m_data = new HashMap();
        public String[] findDynamicAccessProperties()
        {
            return m_columnNames;
        }
        public Class getClassForProperty(String property)
        {
            return null; // default: String...
        }
        public Object getPropertyValue(String propertyName)
        {
            return this.m_data.get(propertyName);
        }
        public void invokeMethod(String methodName)
        {
        }
        public void setPropertyValue(String propertyName, Object value)
        {
            this.m_data.put(propertyName,value);
        }
    }

    public class SelectableTableRow
        extends TableRow
    {
        boolean m_selected;
        public void setSelected(boolean value) { this.m_selected = value; }
        public boolean getSelected() { return this.m_selected; }
    }

    // ------------------------------------------------------------------------
    // members
    // ------------------------------------------------------------------------

    String m_driverClassName;
    String m_connectionURL;
    String m_table;
    SQLUtil.ColumnMetaData[] m_columns; // loaded in init()
    String[] m_columnNames; // loaded in init()
```

```
TableRow m_query = new TableRow();
TEXTGRIDCollection m_lines = new TEXTGRIDCollection();

// ------------------------------------------------------------------------
// constructors
// ------------------------------------------------------------------------

public SQLQueryMgr(String driverClassName,
                   String connectionURL,
                   String table)
{
    m_driverClassName = driverClassName;
    m_connectionURL = connectionURL;
    m_table = table;
    m_columns = SQLUtil.readMetaDataForTable(m_driverClassName,m_connectionURL,m_table);
    m_columnNames = new String[m_columns.length];
    for (int i=0; i<m_columnNames.length; i++)
        m_columnNames[i] = m_columns[i].m_column;
}

// ------------------------------------------------------------------------
// public usage
// ------------------------------------------------------------------------

public TableRow getQuery() { return m_query; }
public TEXTGRIDCollection getLines() { return m_lines; }

public void onQuery()
{
    try
    {
        // build SQL string
        StringBuffer sql = new StringBuffer();
        sql.append("SELECT * FROM " + m_table + " ");
        int counter = 0;
        for (int i=0; i<m_columnNames.length; i++)
        {
            String colName = m_columnNames[i];
            Object colValue = m_query.getPropertyValue(colName);
            if (colValue == null) continue;
            if (counter == 0)
                sql.append("WHERE ");
            else
                sql.append("AND ");
            sql.append(colName + " LIKE " + "'%"+colValue+"%'");
            counter++;
        }
        sql.append(";");
        // execute query
        List queryList = SQLUtil.executeQuery(m_driverClassName,
                                              m_connectionURL,
                                              sql.toString(),
                                              m_columnNames);
        // transfer into lines
        m_lines.clear();
        Iterator iter = queryList.iterator();
        while (iter.hasNext())
        {
            Map columnMap = (Map)iter.next();
            SelectableTableRow str = new SelectableTableRow();
            for (int i=0; i<m_columnNames.length; i++)
                str.setPropertyValue(m_columnNames[i],columnMap.get(m_columnNames[i]));
            m_lines.add(str);
        }

    }
```

```
        catch (Throwable t)
        {
            t.printStackTrace();
        }
    }
}
```

The building blocks are:

- There is a class `TableRow` which is used both for the query properties (`sq.query.type`, `sq.query.id`, etc.) and for the line objects of the grid. This class supports a dynamic set of properties using the `IDynamicAccess` interface. The definition which properties to support comes from the table definition. Again, the `SQLUtil` class is used for accessing the table definition.
- There is an `onQuery` method which builds the SQL selection string out of the query parameters, passes it to the SQL database processing and transfers the result back into the grid collection (`m_lines`).

As you have seen in the above example, only a small amount of coding is required for building extensions that greatly simplify the creation of typical screens. The XML layout that is added using extensions typically matches a pattern on the server side that provides the properties and the processing "without coding" (as seen from the usage point of view).