com.fr.decision.fun.TransferDataSetManagerProvider

10.0TransferDataSetManagerProvider

TransferDataSetManagerProvider.java

```
package com.fr.decision.fun;

import com.fr.base.TableData;
import com.fr.decision.webservice.data.transfer.dataset.TransferDataSetManager;
import com.fr.stable.fun.mark.Mutable;

/**
 * @author lidongy
 * @version 10.0
 * Created by lidongy on 2021/2/25
 */
public interface TransferDataSetManagerProvider<T extends TableData> extends Mutable, TransferDataSetManager<T>
{

    String XML_TAG = "TransferDataSetManagerProvider";

    int CURRENT_LEVEL = 1;
}
```

TransferDataSetManager.java

```
package com.fr.decision.webservice.data.transfer.dataset;

import com.fr.base.TableData;

/**
 * @author lidongy
 * @version 10.0
 * Created by lidongy on 2021/2/25
 */
public interface TransferDataSetManager<T extends TableData> {

    /**
     *
     *
     * @return
     */
    Class<? extends TableData> getDataSetClass();

    /**
     * dataset
     *
     * @param t
     * @return
     */
    String serialize(T t) throws Exception;

    /**
     * dataset
     *
     * @param str
     * @return
     */
    T deserialize(String str) throws Exception;

    /**
     *
     *
     * @param t
     * @return
     */
    String[] getDependencyPaths(T t) throws Exception;
}
```

FR	10.0		

plugin.xml

```
<extra-decision>
        <TransferDataSetManagerProvider class="your class name"/>
</extra-decision>
```

TransferDataSetFactory.java

```
package com.fr.decision.webservice.bean.data.transfer.builder.dataset;

import com.fr.decision.ExtraDecisionClassManager;
import com.fr.decision.fun.TransferDataSetManagerProvider;
import com.fr.decision.webservice.data.transfer.dataset.TransferDataSetManager;
import com.fr.module.tool.ActivatorToolBox;
import com.fr.stable.StringUtils;
import com.fr.stable.collections.CollectionUtils;

import java.util.HashMap;
import java.util.Map;
import java.util.Set;

/**
 * @author lidongy
 * @version 10.0
 * Created by lidongy on 2021/2/25
 */
public class TransferDataSetFactory {
    private static final Map<String, TransferDataSetManager> dataSetManagerMap = ActivatorToolBox.sandbox(new
HashMap<>());

    public static void register(TransferDataSetManager dataSetManager) {
        if (dataSetManagerMap == null) {
            return;
        }
        dataSetManagerMap.put(dataSetManager.getDataSetClass().getName(), dataSetManager);
    }

    public static void reset() {
        dataSetManagerMap.clear();
    }

    public static TransferDataSetManager getDataSetManager(String dataSetClassName) throws Exception {
        TransferDataSetManager dataSetManager = dataSetManagerMap.get(dataSetClassName);
        if (dataSetManager != null) {
            return dataSetManager;
        }
        Set<TransferDataSetManagerProvider> managerProviderSet = ExtraDecisionClassManager.getInstance().
getArray(TransferDataSetManagerProvider.XML_TAG);
        if (!CollectionUtils.isEmpty(managerProviderSet)) {
            for (TransferDataSetManagerProvider provider : managerProviderSet) {
                if (StringUtils.equals(provider.getDataSetClass().getName(), dataSetClassName)) {
                    return provider;
                }
            }
        }

        throw new RuntimeException("Transfer is not supported for this data set.");
    }
}
```

TransferDataSetDataBuilder.java

```
package com.fr.decision.webservice.bean.data.transfer.builder.impl;

...

/**
 * @author lidongy
 * @version 10.0
 * Created by lidongy on 2021/1/14
 */
public class TransferDataSetDataBuilder extends AbstractTransferDataBuilder {
```

```
    private static final String EXPORT_PATH = "dataSet";

    @Override
    public int getEntityType() {
        return DataSetAuthorityControllerImpl.AUTH_TYPE;
    }

    @Override
    public TransferExportEntityDataBean exportEntityData(String entityId) throws Exception {
        TransferExportEntityDataBean result = new TransferExportEntityDataBean();
        result.setId(entityId);
        return result;
    }

    @Override
    public TransferExportEntityDataExtraPropsBean exportDataExtraProps(String entityId) throws Exception {
        TransferDataSetExtraPropsBean result = new TransferDataSetExtraPropsBean();

        TableData tableData = TableDataConfig.getInstance().getTableData(entityId);
        String className = tableData.getClass().getName();
        TransferDataSetManager dataSetManager = TransferDataSetFactory.getDataSetManager(className);
        result.setDataSetClassName(className);
        result.setDataSetInfo(dataSetManager.serialize(tableData));
        return result;
    }

    @Override
    public List<TransferEntityDependencyBean> exportEntityDependencies(String entityId) throws Exception {
        TableData tableData = TableDataConfig.getInstance().getTableData(entityId);
        String className = tableData.getClass().getName();
        TransferDataSetManager dataSetManager = TransferDataSetFactory.getDataSetManager(className);

        List<TransferEntityDependencyBean> dependencyBeanList = new ArrayList<>();
        for (String path : dataSetManager.getDependencyPaths(tableData)) {
            TransferEntityDependencyBean dependencyBean = new TransferEntityDependencyBean();
            dependencyBean.setOriginPath(path);
            dependencyBean.setExportPath(StableUtils.pathJoin(TransferDataFactory.EXPORT_TEMP_DIR, EXPORT_PATH,
new File(path).getName()));
            dependencyBeanList.add(dependencyBean);
        }
        return dependencyBeanList;
    }


    @Override
    public TransferImportDisplayDataBean buildImportData(TransferExportDataBean transferExportDataBean) throws
Exception {
        TransferExportEntityDataBean entityDataBean = transferExportDataBean.getData();
        TransferImportDisplayDataBean displayDataBean = new TransferImportDisplayDataBean
(DataSetAuthorityControllerImpl.AUTH_TYPE);

        //
        String dataSetName = entityDataBean.getId();
        TransferImportDataValidBean entityFullPath = new TransferImportDataValidBean();
        entityFullPath.setImportDataId(dataSetName);
        entityFullPath.setImportDataAttr(dataSetName);
        entityFullPath.setDataAttrValid(TableDataConfig.getInstance().getTableData(dataSetName) == null);
        displayDataBean.setEntityFullPath(entityFullPath);

        //
        List<TransferEntityDependencyBean> dependencyBeanList = transferExportDataBean.getDependencies();
        List<TransferImportDataValidBean> originPathList = new ArrayList<>();
        for (TransferEntityDependencyBean dependencyBean : dependencyBeanList) {
            String originPath = dependencyBean.getOriginPath();
            if (StringUtils.isNotEmpty(originPath)) {
                originPathList.add(new TransferImportDataValidBean(originPath, StringUtils.EMPTY, !
ResourceIOUtils.exist(originPath)));
            }
        }
```

```
        //originPathListnull
        if (originPathList.size() != 0) {
            displayDataBean.setDependencyOriginPath(originPathList);
        }
        displayDataBean.setExportData(transferExportDataBean);
        return displayDataBean;
    }

    @Override
    public List<TransferImportResultBean> importData(List<TransferImportEntityBean> importEntities) {
        List<TransferImportResultBean> importResultBeanList = new ArrayList<>();
        for (TransferImportEntityBean importEntityBean : importEntities) {
            TransferDataSetExtraPropsBean dataSetExtraPropsBean = (TransferDataSetExtraPropsBean)
importEntityBean.getEntityDataExtraPropsBean();
            String dataSetName = importEntityBean.getEntityId();

            TransferImportResultBean resultBean = new TransferImportResultBean(importEntityBean.getImportId(),
DataSetAuthorityControllerImpl.AUTH_TYPE);
            resultBean.setFullPath(dataSetName);
            //
            List<TransferImportDataValidBean> originPath = importEntityBean.getDependencyOriginPath();
            if (!CollectionUtils.isEmpty(originPath)) {
                List<String> dependencyPath = new ArrayList<>();
                dependencyPath.add(originPath.get(0).getImportDataAttr());
                resultBean.setDependencyPath(dependencyPath);
            }

            try {
                String dataSetInfo = dataSetExtraPropsBean.getDataSetInfo();
                String className = dataSetExtraPropsBean.getDataSetClassName();

                TransferDataSetManager dataSetManager = TransferDataSetFactory.getDataSetManager(className);
                TableData tableData = dataSetManager.deserialize(dataSetInfo);

                if (importEntityBean.getEntityFullPath().isDataAttrValid()) {
                    Configurations.update(new WorkerAdaptor(TableDataConfig.class) {
                        @Override
                        public void run() {
                            TableDataConfig.getInstance().addTableData(dataSetName, tableData);
                        }
                    });
                } else {
                    Configurations.update(new WorkerAdaptor(TableDataConfig.class) {
                        @Override
                        public void run() {
                            TableDataConfig.getInstance().removeTableData(dataSetName);
                            TableDataConfig.getInstance().addTableData(dataSetName, tableData);
                        }
                    });
                }

                //
                List<TransferEntityDependencyBean> dependencyBeanList = importEntityBean.
getEntityDependencyBeans();
                for (TransferEntityDependencyBean dependencyBean : dependencyBeanList) {
                    File exportFile = new File(StableUtils.pathJoin(WorkContext.getCurrent().getPath(),
TransferDataFactory.IMPORT_TEMP_DIR, dependencyBean.getExportPath()));
                    TransferDataFactory.importDependencyFile(dependencyBean.getOriginPath(), exportFile);
                }

                resultBean.setImportResult(true);
            } catch (Exception e) {
                FineLoggerFactory.getLogger().error(e, e.getMessage());
                resultBean.setImportMsg(WebServiceUtils.getStackTraceInfo(e));
                resultBean.setImportResult(false);
            }
            importResultBeanList.add(resultBean);
        }
        return importResultBeanList;
    }
```

```
}
```

readXMLwriteXML

GeneralXMLTools.writeXMLableAsString(ds);
GeneralXMLTools.readStringAsXMLable(config,ds);

getDependencyPaths

demodemo-transfer-data-set-manager-provider