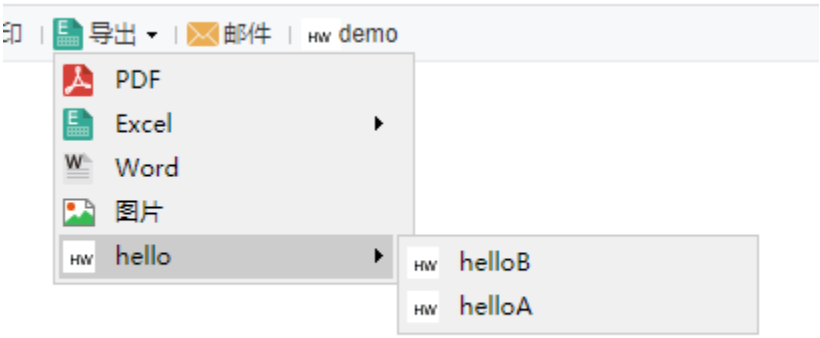
com.fr.report.fun.ExtensionButtonProvider

ToolbarItemProviderExtensionButtonProvider

ExtensionButtonProviderExportToolBarProvider



ExtensionButtonProvider.java

```
package com.fr.report.fun;

import com.fr.form.ui.Widget;
import com.fr.stable.fun.mark.Mutable;
import com.fr.stable.xml.XMLable;

/**
 *
 * @author focus
 * @date Jul 1, 2015
 * @since 8.0
 * ,
 *
 */
public interface ExtensionButtonProvider extends XMLable, Mutable{

        int CURRENT_LEVEL = 2;
        String XML_TAG = "ExtensionButtonProvider";

    /**
     * com.fr.form.ui.ToolBarMenuButton  com.fr.form.ui.ToolBarButton;
     * @return
     */
    Class<? extends Widget> classForDirectoryButton();

    /**
     *
     * @return
     */
         String getParentDirectory();

        /**
         *
         *
         * @return
         */
         String getType();

        /**
         * checkboxweb
         *
         * @return
         */
         String getRelatedCheckBoxTitle();

        /**
         * web
         *
         * @return
         */
         boolean isSelected();

        /**
         * web
         *
         * @param isSelected
         */
         void setSelected(boolean isSelected);

}
```

ExportToolBarProvider.java

```
package com.fr.design.fun;

import com.fr.plugin.injectable.SpecialLevel;
import com.fr.stable.fun.mark.Mutable;

import javax.swing.*;

/**
 * web
 */
public interface ExportToolBarProvider extends Mutable{

        String XML_TAG = SpecialLevel.ExportToolBarProvider.getTagName();

        int CURRENT_LEVEL = 1;

        /**
         *
         *  webcheckbox
         *
         * @param pane
         * @return
         */
        JPanel updateCenterPane(JPanel pane);

        /**
         *
         */
        void populate();

        /**
         *
         */
        void update();
}
```

FR	8.0		
FR	9.0		
FR	10.0		
BI	3.6		BI
BI	4.0		BI
BI	5.1		BI
BI	5.1.2		BI
BI	5.1.3		BI

plugin.xml

```
<extra-report>
        <ExtensionButtonProvider class="your class name"/>
</extra-report>
<extra-designer>
        <ExportToolBarProvider class="your class name"/>
</extra-designer>
```

Export.java

```
package com.fr.report.web.button;

import com.fr.base.IconManager;
import com.fr.base.TemplateUtils;
import com.fr.form.ui.Button;
import com.fr.form.ui.ToolBarMenuButton;
import com.fr.general.ComparatorUtils;
import com.fr.log.FineLoggerFactory;
import com.fr.report.ExtraReportClassManager;
import com.fr.report.fun.ExtensionButtonProvider;
import com.fr.stable.xml.XMLPrintWriter;
import com.fr.stable.xml.XMLableReader;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Set;

/**
 * ”“PDFExcelWordHTML
 * ExcelsheetExcel
 */
public final class Export extends ToolBarMenuButton {
    private boolean pdfAvailable = true;
    private boolean excelPAvailable = true;
    private boolean excelOAvailable = true;
    private boolean excelSAvailable = true;
    private boolean wordAvailable = true;
    private boolean imageAvailable = true;
    private boolean htmlAvailabel = true;

    //
    private List<ExtensionButtonProvider> parentDerectorys;
    // map
    private Map<ExtensionButtonProvider, List<ExtensionButtonProvider>> options;
    //
    private Map<ExtensionButtonProvider, List<ExtensionButtonProvider>> selectedOptions;

    public Export() {
        super(TemplateUtils.i18nTpl("Export"), IconManager.EXPORT.getName());
        initCompoents();
    }

    //
    private void initCompoents() {
        Set<ExtensionButtonProvider> exportBtnAdapters = ExtraReportClassManager.getInstance().getArray
(ExtensionButtonProvider.XML_TAG);
        initExtraParentDerectory(exportBtnAdapters);
        initExtraOptions(exportBtnAdapters);
        initExtraSelectedOptions();
    }

    //
    private void initExtraParentDerectory(Set<ExtensionButtonProvider> exportBtnAdapters) {
        if (exportBtnAdapters != null) {
            for (ExtensionButtonProvider provider : exportBtnAdapters) {
                if (provider.getParentDirectory() == null) {
                    if (parentDerectorys == null) {
                        parentDerectorys = new ArrayList<ExtensionButtonProvider>();
                    }
                    parentDerectorys.add(provider);
```

```
            }
        }
    }
}

//
private void initExtraOptions(Set<ExtensionButtonProvider> exportBtnAdapters) {
    if (parentDerectorys != null) {
        for (int j = 0; j < parentDerectorys.size(); j++) {
            ExtensionButtonProvider parent = parentDerectorys.get(j);
            List<ExtensionButtonProvider> childItems = new ArrayList();
            for (ExtensionButtonProvider exportBtnAdapter : exportBtnAdapters) {
                if (exportBtnAdapter.getParentDirectory() != null) {
                    if (ComparatorUtils.equals(parent.getType(), exportBtnAdapter.getParentDirectory())) {
                        childItems.add(exportBtnAdapter);
                    }
                }
                if (options == null) {
                    options = new HashMap<ExtensionButtonProvider, List<ExtensionButtonProvider>>();
                }
                options.put(parent, childItems);
            }

        }
    }
}

//
private void initExtraSelectedOptions() {
    if (options != null) {
        boolean flag = false;
        Iterator<ExtensionButtonProvider> it = options.keySet().iterator();
        while (it.hasNext()) {
            ExtensionButtonProvider key = it.next();
            List<ExtensionButtonProvider> childItems = options.get(key);
            for (int i = 0; i < childItems.size(); i++) {
                if (childItems.get(i).isSelected()) {
                    flag = true;
                    break;
                }
            }
            if (flag) {
                if (selectedOptions == null) {
                    selectedOptions = new HashMap<ExtensionButtonProvider, List<ExtensionButtonProvider>>();
                }
                selectedOptions.put(key, childItems);
            }
        }
    }
}

/**
 * PDF
 *
 * @return PDFtrue
 */
public boolean isPdfAvailable() {
    return pdfAvailable;
}

/**
 * PDF
 *
 * @param pdfAvailable truePDF
 */
public void setPdfAvailable(boolean pdfAvailable) {
    this.pdfAvailable = pdfAvailable;
}

/**
 * Word
```

```
     *
     * @return Wordtrue
     */
    public boolean isWordAvailable() {
        return wordAvailable;
    }

    /**
     * Word
     *
     * @param wordAvailable trueWord
     */
    public void setWordAvailable(boolean wordAvailable) {
        this.wordAvailable = wordAvailable;
    }

    /**
     * Excel
     *
     * @return Exceltrue
     */
    public boolean isExcelPAvailable() {
        return excelPAvailable;
    }

    /**
     * Excel
     *
     * @param excelPAvailable trueExcel
     */
    public void setExcelPAvailable(boolean excelPAvailable) {
        this.excelPAvailable = excelPAvailable;
    }

    /**
     * Excel
     *
     * @return Exceltrue
     */
    public boolean isExcelOAvailable() {
        return excelOAvailable;
    }

    /**
     * Excel
     *
     * @param excelOAvailable trueExcel
     */
    public void setExcelOAvailable(boolean excelOAvailable) {
        this.excelOAvailable = excelOAvailable;
    }

    /**
     * Excelsheet
     *
     * @return Excelsheettrue
     */
    public boolean isExcelSAvailable() {
        return excelSAvailable;
    }

    /**
     * Excelsheet
     *
     * @param excelSAvailable trueExcelsheet
     */
    public void setExcelSAvailable(boolean excelSAvailable) {
        this.excelSAvailable = excelSAvailable;
    }

    /**
```

```
     * Excelsheet
     *
     * @return Exceltrue
     */
    public boolean isExcelAvailable() {
        return excelPAvailable || excelOAvailable || excelSAvailable;
    }


    /**
     *
     *
     * @param imageAvailable truefalse
     */
    public void setImageAvailable(boolean imageAvailable) {
        this.imageAvailable = imageAvailable;
    }

    /**
     *
     *
     * @return true
     */
    public boolean isImageAvailable() {
        return imageAvailable;
    }

    /**
     * HTML
     *
     * @return HTMLtrue
     */
    public boolean isHtmlAvailabel() {
        return htmlAvailabel;
    }

    /**
     * HTML
     *
     * @param htmlAvailabel true
     */
    public void setHtmlAvailabel(boolean htmlAvailabel) {
        this.htmlAvailabel = htmlAvailabel;
    }

    /**
     *
     *
     * @return false
     */
    public boolean supportMobile() {
        return false;
    }


    @Override
    public String getXType() {
        return ButtonTypeConstants.EXCEL_MENU;
    }

    /**
     *
     *
     * @return
     */
    public Button[] createMenuItems() {
        java.util.List list = new ArrayList();
        if (pdfAvailable) {
            list.add(new PDF());
        }
        if (isExcelAvailable()) {
```

```
            list.add(new Excel(excelPAvailable, excelOAvailable, excelSAvailable));
        }
        if (wordAvailable) {
            list.add(new Word());
        }
        if (imageAvailable) {
            list.add(new Image());
        }

        if (selectedOptions != null) {
            Iterator<ExtensionButtonProvider> it = selectedOptions.keySet().iterator();
            while (it.hasNext()) {
                ExtensionButtonProvider adapter = it.next();
                Class export = adapter.classForDirectoryButton();
                try {
                    if (export != null) {
                        Button button = (Button) export.newInstance();
                        list.add(button);
                    }
                } catch (InstantiationException e) {
                    FineLoggerFactory.getLogger().error(e.getMessage(), e);
                } catch (IllegalAccessException e) {
                    FineLoggerFactory.getLogger().error(e.getMessage(), e);
                }
            }
        }


        //23452 HTML,
//                if (htmlAvailabel) {
//                        list.add(new HTML());
//                }

        return (Button[]) list.toArray(new Button[list.size()]);
    }

    private void readExtraXML(XMLableReader reader) {
        if (selectedOptions != null && !selectedOptions.isEmpty()) {
            Iterator<ExtensionButtonProvider> it = selectedOptions.keySet().iterator();
            while (it.hasNext()) {
                List<ExtensionButtonProvider> adapters = (List<ExtensionButtonProvider>) selectedOptions.get(it.
next());
                for (int i = 0; i < adapters.size(); i++) {
                    adapters.get(i).readXML(reader);
                }
            }
        }
    }


    /**
     * XML
     *
     * @param reader XML
     */
    public void readXML(XMLableReader reader) {
        super.readXML(reader);
        readExtraXML(reader);
        if (reader.isChildNode()) {
            if (reader.getTagName().equals("Buttons")) {
                Export.this.setPdfAvailable(reader.getAttrAsBoolean("pdf", true));
                Export.this.setWordAvailable(reader.getAttrAsBoolean("word", true));
                Export.this.setExcelPAvailable(reader.getAttrAsBoolean("excelP", true));
                Export.this.setExcelOAvailable(reader.getAttrAsBoolean("excelO", true));
                Export.this.setExcelSAvailable(reader.getAttrAsBoolean("excelS", true));
                Export.this.setImageAvailable(reader.getAttrAsBoolean("image", true));
                Export.this.setHtmlAvailabel(reader.getAttrAsBoolean("html", true));
            }
        }
    }
```

```
    /**
     * XML
     *
     * @param writer xml
     */
    public void writeExtraXML(XMLPrintWriter writer) {
        if (selectedOptions != null && !selectedOptions.isEmpty()) {
            Iterator it = selectedOptions.keySet().iterator();
            while (it.hasNext()) {
                List<ExtensionButtonProvider> adapters = (List<ExtensionButtonProvider>) selectedOptions.get(it.
next());
                for (int i = 0; i < adapters.size(); i++) {
                    adapters.get(i).writeXML(writer);
                }
            }
        }
    }

    /**
     * XML
     *
     * @param writer xml
     */
    public void writeXML(XMLPrintWriter writer) {
        super.writeXML(writer);
        writeExtraXML(writer);
        writer.startTAG("Buttons")
                .attr("pdf", pdfAvailable)
                .attr("excelP", excelPAvailable)
                .attr("excelO", excelOAvailable)
                .attr("excelS", excelSAvailable)
                .attr("word", wordAvailable)
                .attr("image", imageAvailable)
                .attr("html", htmlAvailabel)
                .end();
    }

    /**
     *
     *
     * @param obj
     * @return truefalse
     */
    public boolean equals(Object obj) {
        return obj instanceof Export && super.equals(obj)
                && ((Export) obj).excelPAvailable == excelPAvailable
                && ((Export) obj).excelOAvailable == excelOAvailable
                && ((Export) obj).excelSAvailable == excelSAvailable
                && ((Export) obj).imageAvailable == imageAvailable
                && ((Export) obj).pdfAvailable == pdfAvailable
                && ((Export) obj).wordAvailable == wordAvailable
                && ((Export) obj).htmlAvailabel == htmlAvailabel;
    }
}
```

webExportExportToolBarProviderselectedOptions

ExtensionButtonProviderExportToolBarProvider

ExtensionButtonProvider

1.AbstractExtensionButton

2.AbstractExtensionMenuButtoncreateMenuItems

booltrue

SundryKit.loadToolbarIcon(,);

getParentDirectorygetType

xml

demo

demodemo-extension-button-provider

(1)web

(1)JSCSS

(1)

demo-export-xml