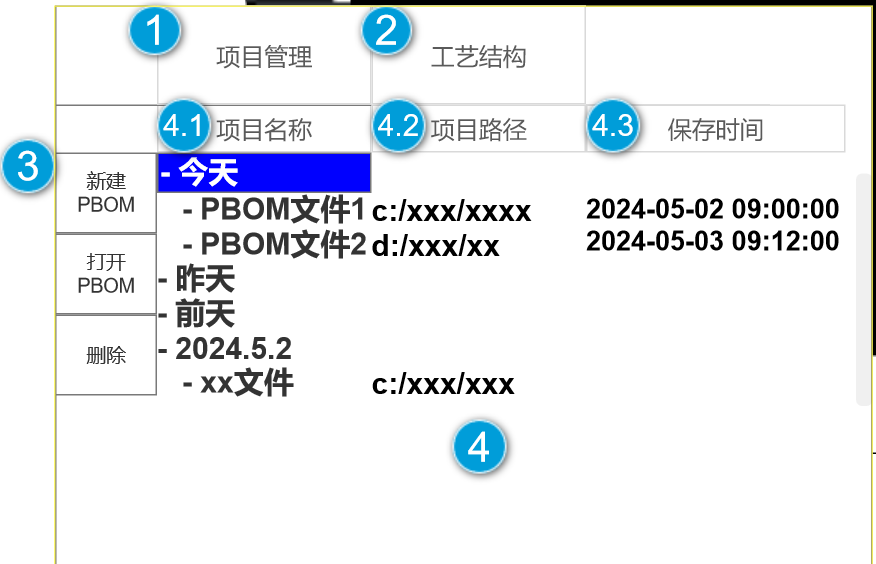
CAPP工艺管理设计方案概要

此文件暂且列出核心的类以及类的核心成员，更加细节的实现在编写程序时补足

# 工艺设计面板



CAPPPanel用于在 BrisCAD平台中显示Qt控件，CAPPPanel作为整个面板的容器， 利用BrisCAD提供的接口将CAPPPanel对象嵌入至侧边栏。运行面板显示命令PC\_ProcessTree时展开面板。

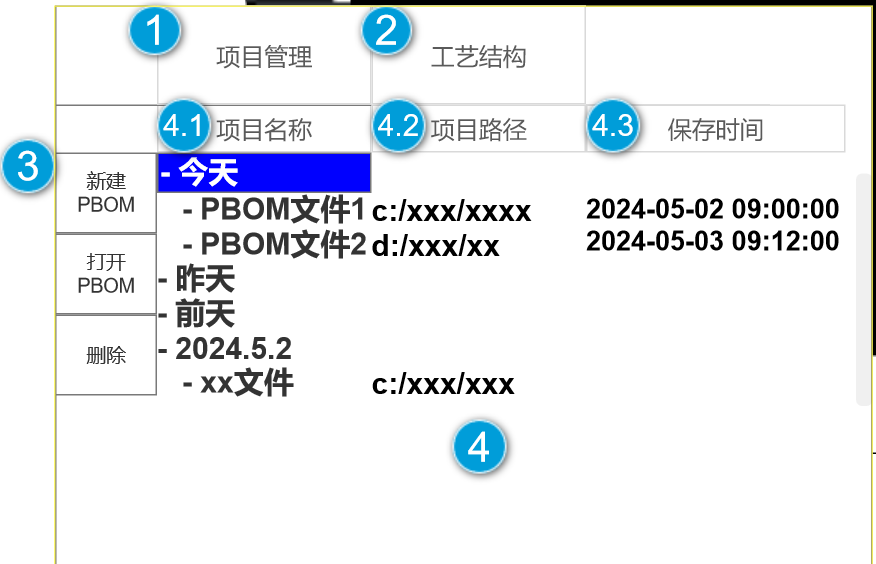
1. Class CAPPPanel: public BcUiPanelMFC
2. {
3. public:
4. CAPPPanel();
5. ~CAPPPanel();
6. void setClose(bool);// 获取面板状态
7. bool getClose();
8. //响应事件
9. virtual void SizeChange();
10. virtual void OnClosing();
11. virtual void OnShow(bool);
12. void init();
13. private:
14. bool m\_isClosed;
15. CAPPPanelView\* m\_panelView;*//Qt最外层容器，用于存放所有Qt控件*
16. }

//Qt最外层容器，用于存放所有Qt控件

1. Class CAPPPanelView: public QWidget
2. {
3. public:
4. CAPPPanelView();//
5. ~CAPPPanelView();
6. private:
7. //备注：tab页对象可以在CAPPPanelView构造中生成，也可以在ui设计[QtDesigner]中添加
8. CAPPPanelViewClass\* ui;// 整个面板的ui设计，包含一个TabWidget用于管理项目管理tab和工艺结构树tab,
9. //CAPPProcessTreeTab \* m\_pProcessTreeTab; *//工艺结构树对应的Tab视图*
10. //CAPPProjectManagerTab \* m\_pProjectManagerTab; *// 项目管理对应的Tab视图*

}

## 项目管理视图



显示项目管理面板时，由一个QGroupBox和一个QTabWidget 水平布局占满CAPPProjectManageTab容器。其中③部分为QGroupBox以及其中的按钮。①和②为QTabWidget 中两个tab, ④部分为项目管理面板对应的QWeight,其中放置一个QTreeWidget用于显示项目管理的历史记录，数据记录在本地CAD pbom插件下的文件中。

//////////////////////////CAPP Project TAB///////////////////////////

// the CAPP Project TAB

Class CAPPProjectManagerTab:QWidget

{

Public:

QOBJECT

CAPPProjectManager\* getProjectManager();

CAPPProjectManagerTab(QWidget\*); //构造函数中绑定按钮和事件

Ui:: CAPPProjectManagerTabClass; // 按钮组和树在ui文件中设计

CAPPProjectManager\* m\_p CAPPProjectManager;//项目管理对象

}

//CAPP项目配置类

//该类表示CAPP项目的配置

Class CAPPProjectConfig

{

Bool setNodeConfigFilePath(string);

String getNodeConfigFilePath();

//一些项目全局的配置

// 节点配置xml路径

String strNodeConfigFilePath;

}

//CAPP项目类

//该类表示CAPP项目

Class CAPPProject

{

//构造

CAPPProject(string);

//获取项目路径

Bool setProjectPath(string);

string getProjectPath();

//

Bool save();

Bool saveAs(string);

Bool close();

QString m\_strName;//项目名称

QString m\_strCreateTime;//创建时间

QString m\_strLastModified;//修改时间

QString m\_strPath;//路径

ProcessTreeModel\* pProcessTreeModel;//结构树数据

CAPPProjectConfig\* m\_pConfig;//项目配置

}

//项目历史类

Class CAPPProjectHistory

{

//序列化相关

bool SerializeOut(QDataStream&);

bool SerializeIn(const QDataStream&);

QString m\_strName;//项目名称

QString m\_strCreateTime;//创建时间

QString m\_strLastModified;//修改时间

QString m\_strPath;//路径

}

//项目历史管理类

Class CAPPProjectHistoryManager

{

//成员相关

Void clear();

Void addHistory(const CAPPProjectHistory&);

QList<CAPPProjectHistory>& getAllHisrory();

//文件相关

bool writeToFile();//写入文件

bool readFromFile ();//从文件读入

string getFilePath();

QList<CAPPProjectHistory> m\_pHistory;//项目打开历史列表

}

//项目管理类

//项目管理类用于CAPP项目的管理

Class CAPPProjectManager

{

bool openProject(string path);

void saveProject();

bool closeProject();

bool newProject(string path);

bool delProject();

bool setProcessTreeManager(ProcessTreeManager\*);

CAPPProject\* m\_pCurProject;//当前项目

QTreeView\* m\_pTreeView;//管理的项目打开历史的显示

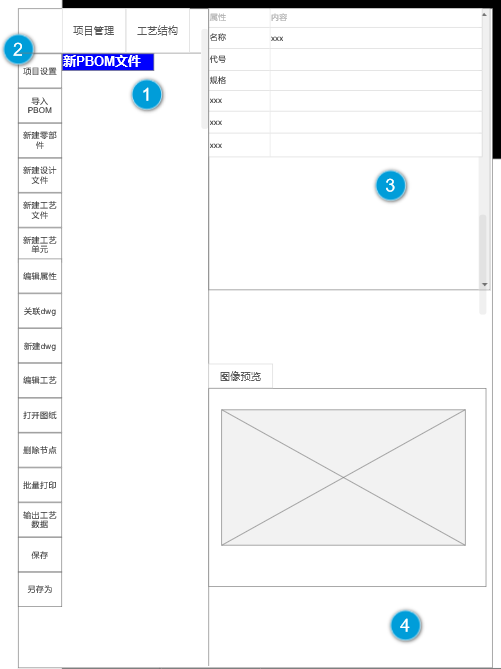
QStandardModel\* m\_pModel;//管理的项目历史显示对应的模型

CAPPProjectHistoryManager\* m\_pHistoryManager;//打开历史管理类对象

ProcessTreeManager\* m\_p ProcessTreeManager;//工艺结构树的管理对象

}

## 工艺结构视图



1. QTreeWidget 用于显示工艺结构树, QGroupBox放置多个侧边按钮

由CAPPProcessTreeTab管理工艺结构Tab页中的结构树的显示。

//////////////////the CAPP Process Tree TAB //////////////////

Class CAPPProcessTreeTab:QWidget

{

Public:

QOBJECT

CAPPProcessTreeManager\* getProcessTreeManager();

CAPPProcessTreeTab (QWidget\*); //构造函数中绑定按钮和事件

Ui:: CAPPProcessTreeTabClass; // 按钮组和树在ui文件中设计

CAPPProcessTreeManager\* m\_pProcessTreeManager;//工艺结构树管理对象

}

//工艺结构树Tab管理类，用于管理整个工艺结构树

Class ProcessTreeManager

{

Public:

//节点相关

bool importPBOM();//导入PBOM至当前节点

void onNewPart();//新建零部件节点

void onNewDesignNode();//新建设计文件节点

void onDelNode();//删除节点

void onEditNodeAttributes();//编辑节点属性

void onRelatedDwg();//关联图纸

void onNodeConfig ();/节点配置

void onRemoveNode();//删除节点

void onRenameNode();//重命名节点

void onNodeAttribute();//显示节点属性

void onNodeMoveUpLevel(); //节点上移一级

void onNodeMoveDownLevel(); //节点上移一级

void onNodeMoveUp(); //节点上移

void onNodeMoveDown(); //节点上移

//图纸相关

void onCreateDwg();

void onOpenDwg();

void onBatchPrint(); //批量打印

void onOutputProcess();//输出工艺数据

//项目相关

Bool setCurProject(CAPPProject\*);

CAPPProject\* getCurProject();

Bool setProjectManager(CAPPProjectManager\*);

//数据相关

ProcessTreeModel\* getProcessTreeModel();

Private:

QTreeView\* m\_pTreeView;// 树的显示部分

ProcessTreeModel \* m\_pTreeModel;//结构树的数据管理部分

CAPPProjectManager\* pProjectManager;//项目管理类对象

}

//保存工艺结构树的数据, QStandardModel 本身是一个树结构

// 模板类需要需要实现序列化函数

Template <T>

class ProcessTreeModel: public QStandardModel

{

Public:

//序列化相关

bool SerializeOut(QDataStream&);

bool SerializeIn(const QDataStream&);

//节点相关

T\* getCurData();//获取选中节点数据

QStandardItem\* getCurItem();//获取当前选中item

QStandardItem\*GetRootItem();//获取根

}

// 属性显示视图

1. class NodeAttributeView:QWidget
2. {
3. public:
4. NodeAttributeView();
5. ~NodeAttributeView();
6. void setAttributes(const QMap<QString,QString>&);
7. QMap<QString,QString>& getAttributes();
8. private:
9. NodeAttributeViewClass\* ui;
10. QMap<QString,QString> m\_mapAttributes;
11. }

//图像预览区域

1. class DwgPreviewView: QWidget
2. {
3. public:
4. DwgPreviewView();
5. ~DwgPreviewView();
6. void setFile(const QString&);
7. QString getFile();
8. void refresh();//更新显示
9. private:
10. DwgPreviewViewClass\* ui;
11. QString m\_strFile;
12. }

# 节点定义

节点类型定义

节点类型定义类完全定义了一个节点类型的属性和与其他节点类型之间的连接规则

**//**节点定义类，用于定义一个节点类型

class NodeTypeDefinition

{

public:

QString getNodeType();

bool setNodeType();

const QStringList& getAttrDefs();

bool setAttrDefs(const QStringList&);

const QStringList& getAllowedParents();

bool setAllowedParents(const QStringList&);

const QStringList& getAllowedChildren();

bool setAllowedChildren(const QStringList&);

bool getAllowedAttachFile();

void setAllowedAttachFile(bool);

bool isAllowedParent(string);

bool isAllowAllParents();

bool isAllowedChild(string);

bool isAllowAllChildren();

private:

QString m\_strNodeTypeName;//节点类型的名字

QString m\_strIconPath;//节点类型对应图标的文件名(不含路径)

QStringList m\_slAttrDefs;//节点类型的属性的列表

QStringList m\_slRefAttrDefs;//节点类型的关系属性的列表

QStringList m\_slAllowedParents;//允许作为父节点的类型的列表

QStringList m\_slAllowedChildren; //允许作为子节点的类型的列表

bool m\_bAllowedAttachFile;//是否允许绑定文件

bool m\_bAllowedAllParents;//是否允许任何类型作为父节点

bool m\_bAllowedAllChildren;//是否允许任何类型作为子节点

};

//节点类型管理类

//节点类型管理类用于管理所有的节点类型和与节点类型相关的数据，并包含节点类型配置的读档和归档

class NodeTypeManager

{

bool readConfigFromFile();//从配置文件中读入配置

bool writeConfigToFile();//将配置写入文件

QStringList getAllNodeTypes();//获取所有节点类型列表

bool newNodeType(const NodeTypeDefinition&);//新增节点类型

bool delNodeType(QString strNodeType);//删除节点类型

NodeTypeDefinition\* findNodeType(QString strNodeType);//查找指定节点类型并可用于修改

const QStringList& getCommonAttributes();//获取所有节点类型公用的属性列表，用于更加方便地配置各个节点类型的属性，暂放此处

void setCommonAttributes(const QStringList&);//设置所有节点类型公用的属性列表

//get the xml file path of the node type configure

std::wstring getConfigFilePath();

private:

QStringList m\_slCommonAttrs; //所有节点类型公用的属性列表

QList<NodeTypeDefinition> m\_lsAllNodeTypes; //所有节点类型

};

该类会在新的CAD进程第一次运行CAPP插件程序后初始化单例，从配置文件中导入配置。之后所有关于节点类型的配置都通过此类进行，如有必要可以将此类设计为单例类防止构造。

// 树节点

项目树使用QTreeView展示，使用QStandardModel保存树的数据结构， QStandardModel中的每个节点为一个QStandardItem，其可以设置一个自定义类作为数据类，此处使用TreeNodeData作为它的数据类，以下是TreeNodeData的定义。

Class TreeNodeData

{

QString m\_NodeName;//节点名称

QString m\_TypeName;//节点类型名称

QMap<QString, QString> m\_mapAttributeContent;//节点属性内容

QString m\_strIcon;//节点图标

QString m\_strFilePath;//绑定图纸文件路径

CString m\_strFileDescription; //图纸文件的描述

//…

}

//节点工厂类，所有的节点对象通过此类生成

class NodeFactory

{

public:

NodeFactory(NodeTypeManager\*);//构造函数中传入对应的节点类型管理对象

// 工厂方法，用于创建节点

TreeNodeData\* createNode(std::string strTypeName, std::string strNodeName);

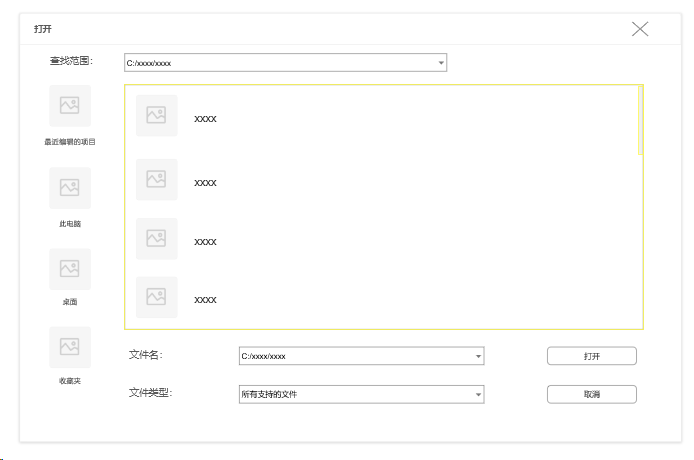
Private:

NodeTypeManager\* m\_pNodeTypeManager;//

}

# 其他功能逻辑

## 打开PBOM



相关业务逻辑在以下函数中处理:  
bool CAPPProjectManager::onOpenProject();

{

//select xxx.pdm file

String filePath;

CAPPProject\* pProject = new CAPPProject(filePath);

If(!pProject->openStatus) return false;

//关闭当前项目

m\_curProject->close();

m\_curProject = pProject;

//同步更新结构树tab

m\_pProjectTreeManager->setCurProject(pProject);

//更新历史记录

m\_pHistoryManager->addHistory(new CAPPProjectHistory(pProject));

}

选择文件后，将xxx.pbm文件序列化为CAPPProject对象，赋值给CAPPProjectManager对象的CAPPProcessProject\* m\_curProject 成员，同时更新工艺结构树。

## 新建PBOM

相关业务逻辑在以下函数中处理:  
bool CAPPProjectManager::onNewProject()

{

//新建路径

String projectPath;

//新建CAPPProject对象

m\_curProject = new CAPPProject(projectPath);

If(!pProject->openStatus) return false;

//关闭当前项目

m\_curProject->close();

m\_curProject = pProject;

//同步更新工艺结构树

m\_pProjectTreeManager->setCurProject(m\_curProject);

}

## 导入PBOM

和打开PBOM逻辑相同，打开之后将此CAPPProject的节点插入到需要导入的节点下，同时需要将需要导入的pbom对应的文件夹中的文件拷入主项目对应的文件夹下。

## 保存

相关业务逻辑在以下函数中处理:

bool CAPPProjectManager::onSaveProject()

{

If(m\_curProject)

m\_curProject->save();

}

将当前CAPPProject对象序列化至CAPPProject对象成员m\_strFilePathName对应的路径中。

## **项目另存为**

bool CAPPProjectManager::onSaveProject()

{

//select filepath

String filePath;

If(m\_curProject)

m\_curProject->saveAs(filePath);

}

将当前CAPPProject对象序列化至用户选择的文件中。

# 对话框

## 节点配置

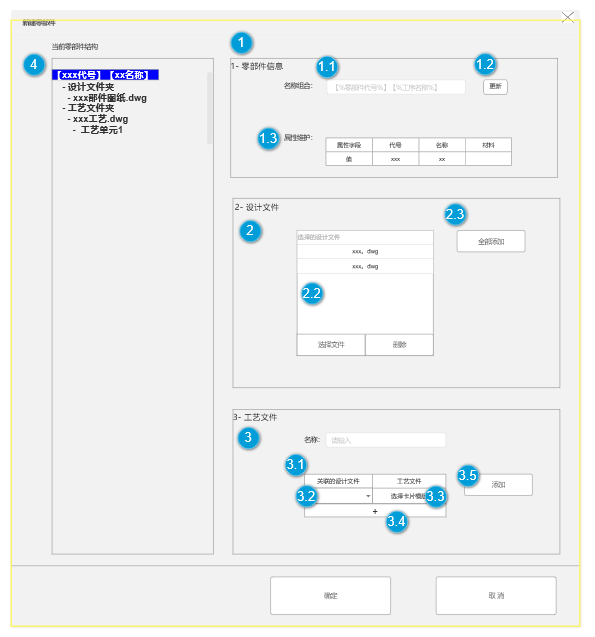


1. class CAPPNodeConfigDlg: QWidget
2. {
3. public:
4. CAPPNodeConfigDlg();
5. ~CAPPNodeConfigDlg();
6. void init();*// 控件和数据的初始化*
7. void initNodeTypeComboBox();*// 初始化类型名称下拉控件*
8. void initNodeKeyFieldComboBox();*// 初始化关键字段控件*
9. // 响应事件
10. void onAddNodeType(QString strNodeType);
11. void onRenameNodeType();
12. void  onDelNodeType();
13. void onSelNodeTypeIcon();*//选择图标*
14. void onNewKeyField();*// 新建关键字段*
15. void onAddAttribute();*//添加属性至当前类型*
16. void onWithdrawAttribute();*// 撤回*
17. void onNewAttribute();*//新建属性*
18. void onDelAttribute();*// 删除*
19. void onRenameAttribute();*// 删除*
20. void onOK();
21. vodi onCancel();
22. private:
23. CAPPNodeConfigDlgClass\* ui;*// QT ui*

}

对于节点配置，通过全局单例对象NodeTypeManager提供的函数来实现。

## 新建零部件



1. class CAPPNewPartNodeDlg: QWidget
2. {
3. public:
4. CAPPNewPartNodeDlg();
5. ~CAPPNewPartNodeDlg();
6. void init();*// 控件和数据的初始化*
7. void initAttrbuteTable();*//初始化属性维护控件*
8. void initProcessFileTable();*//初始化工艺文件控件*
9. // 响应事件
10. void onUpdateName();*// 更新名称*
11. void onSelDesignFile();*//选择设计文件*
12. void onDelDesignFile();*//删除设计文件*
13. void onAddDesignFile(); *// 将设计文件添加至树*
14. void onAddProcessRow();*//工艺文件表添加行*
15. void onSelProcessCardTemplate();*//选择卡片模板*
16. void onAddProcess();*//添加工艺文件至子树*
17. void onOK();
18. vodi onCancel();
19. private:
20. CAPPNewPartNodeDlgClass\* ui;*// QT ui*
21. QStandardItemModel\* m\_pModel;*//子树的数据*
22. }

④处结构树使用工艺结构面板中的相同的结构，方便直接插入到主树中。

确认之后依据需求在pbom项目对应文件夹中生成相应的工艺文件。在新建一个工艺文件时，执行以下步骤：

1.新建一个工艺文件dwg并且打开。  
2.从设计文件的模型空间中取出图纸放入工艺文件dwg的模型空间。

3.根据工艺单元的新建对应图纸空间，并使用生成卡片命令在对应的图纸空间中生成卡片。

## 新建工艺文件夹节点



1. class CAPPNewProcessFileDlg: QWidget
2. {
3. public:
4. CAPPNewProcessFileDlg();
5. ~CAPPNewProcessFileDlg();
6. void init();*// 控件和数据的初始化*
7. // 响应事件
8. void onSelDesignFile();*//选择设计文件*
9. void onDelDesignFile();*//删除设计文件*
10. void onSelProcessCardTemplate();*//选择卡片模板*
11. void onOK();*//确认*
12. vodi onCancel();*//取消*
13. private:
14. CAPPNewProcessFileDlgClass\* ui;*// QT ui*
15. }

实现同新建零部件。

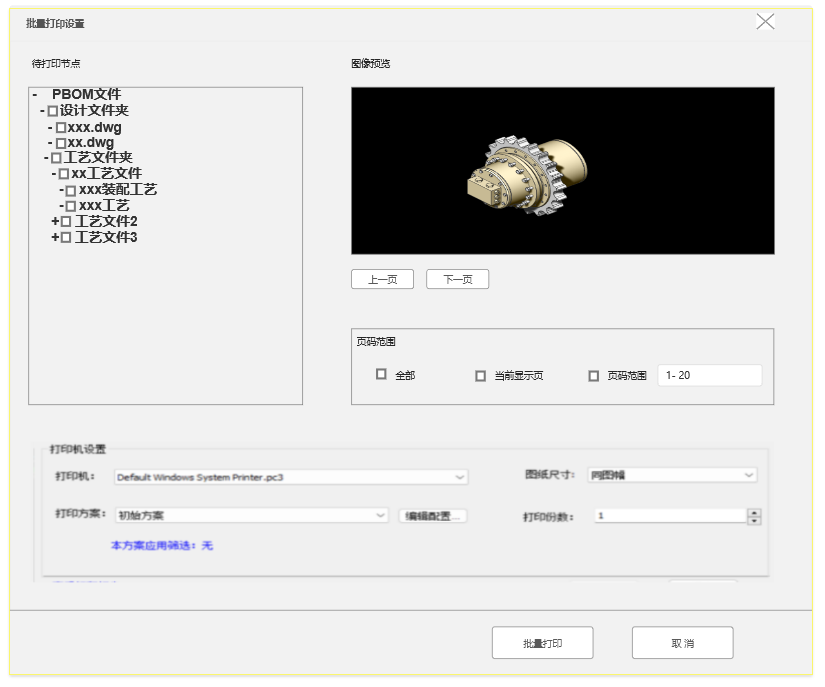
## 编辑属性



1. class CAPPEditNodeAttributeDlg: QWidget
2. {
3. public:
4. CAPPEditNodeAttributeDlg();
5. ~CAPPEditNodeAttributeDlg();
6. void init();*// 控件和数据的初始化*
7. // 响应事件
8. void onOK();*//确认*
9. vodi onCancel();*//取消*
10. private:
11. CAPPEditNodeAttributeDlgClass\* ui;*// QT ui*
12. }

直接获取节点对象的属性进行编辑即可。

## 批量打印[第一版先不实现]



1. class CAPPBatchPrintDlg: QWidget
2. {
3. public:
4. CAPPBatchPrintDlg();
5. ~CAPPBatchPrintDlg();
6. void init();*// 控件和数据的初始化*
7. void initTree();*// 初始化树控件*
8. void initImagePreview();*//初始化图像预览*
9. void initPrintDeviceComboBox();*//初始化打印机下拉控件*
10. void initPrintSlnCombo();*// 初始化打印方案控件*
11. void initDrawingSizeCombo();*// 初始化图纸尺寸控件*
12. void initPageRangeCheckGroup();*//初始化页码选择复选框组控件*
13. // 响应事件
14. void onCheckNode();*//响应勾选节点*
15. void onPreviewNextPage();*//下一页*
16. void onPreviewPrePage();*//上一页*
17. void onOK();*//确认打印*
18. vodi onCancel();*//取消*
19. private:
20. CAPPBatchPrintDlgClass\* ui;*// QT ui*
21. QStandardItemModel\* m\_pModel;*//子树的数据*
22. }

## 插入节点、新建子节点



1. class CAPPNewNodeDlg: QWidget
2. {
3. public:
4. CAPPNewNodeDlg();
5. ~CAPPNewNodeDlg();
6. void init();*// 控件和数据的初始化*
7. void initNodeTypeCombo();*//初始化类型名称复选框控件*
8. // 响应事件
9. void onNodeTypeChanged();
10. void onOK();*//确认打印*
11. vodi onCancel();*//取消*
12. private:
13. CAPPNewNodeDlgClass\* ui;*// QT ui*
14. }

通过全局单例对象NodeTypeManager提供的函数即可实现此功能。

## 数据库工具类

ADO::\_ConnectionPtr,用于访问数据库

数据库表  
模板库放在本地数据库文件中

ProcessCardTemplate // 本地卡片模板表  
ProcessCardTemplateID[int] ProcessCardTemplateName[str]

## XML 操作类[tinyxml]

用xml文件储存本地配置

xml结构

<?xml version="1.0" encoding="UTF-8"?>

<CappConfigure>

<NodeTypes>

<NodeType>

<NodeTypeName>类型2</NodeTypeName>

<IsDefault></IsDefault>

<IconPath>C:\</IconPath>

<NodeTypeAttrDefs>

<NodeTypeAttrDef>

<AttributeName>属性1</AttributeName>

<IsDefault>false</IsDefault>

</NodeTypeAttrDef>

</NodeTypeAttrDefs>

<NodeTypeRefAttrDefs>

</NodeTypeRefAttrDefs>

<AllowedParents>

</AllowedParents>

<AllowedChildren>

<AllowedChild>all</AllowedChild>

</AllowedChildren>

<AllowedAttachFile>true</AllowedAttachFile>

</NodeType>

</NodeTypes>

<CommonAttributes>

<CommonAttribute>属性1</CommonAttribute>

</CommonAttributes>

</CappConfigure>

# 注意事项

## 文件路径相关

- 新建的工艺文件放在xxx.pdm项目文件同名的文件夹下，并用名称组合【设计文件+卡片模板名称+唯一标识】

## 同步问题

在结构树中做的相关操作如删除工艺单元需同步到CAD中，即删除对应的图纸空间。~~同理在CAD中例如删除模型空间的操作通过反应器捕捉到信号将结果更新到结构树中。  
 通过结构树找到对应图纸的思路：  
pbom项目同级目录应有文件夹，其中存放着具有唯一id标识的图纸文件[工艺文件]。若在CAD中删除图纸空间，也可在通过当前打开的图纸文件的名称对应到结构树上的节点。~~

## 序列化版本管理

- 先序列化版本号，再依次序列化节点数据  
序列化方法：  
遍历树，对于节点数据，将其写入QDataStream， 再将QDataStream保存至文件

## 保存问题

因为删除文件相关的操作的不易撤回性质，项目文件相关的修改操作将被即时保存项目文件，避免在项目未保存情况下出现结构树与文件不对应的情况。由于操作设计文件管理，做一个撤回功能较为复杂，第一版不考虑。

## 代理类型

打开项目文件，可能存在本地配置中没有的类型，该类型的规则由一个默认的代理类型决定。