1. **Descrição das classes**
    a. **ThermalPrinter**
    b. **GenerateTicket**
    c. **IDeviceContext**
2. **Descrição dos metêdos**
    a. **ThermalPrinter::ThermalPrinter**
    b. **ThermalPrinter::BuildQuickResponseCode**
    c. **ThermalPrinter::SearchingPrinter**
    d. **ThermalPrinter::ProcessXMLFile**
    e. **ThermalPrinter::ProcessTicketData**
    f. **ThermalPrinter::IsReady**
    g. **ThermalPrinter::Write**
    h. **ThermalPrinter::Build**
    i. **ExtensibleMarkupLanguage::Load**
    j. **ExtensibleMarkupLanguage::Parse**
    k. **GenerateTicketData::GetHeader**
    l. **GenerateTicketData::GetPDV**
    m. **GenerateTicketData::GetExtract**
    n. **GenerateTicketData::GetListOfProductHeader**
    o. **GenerateTicketData::GetProductItems**
    p. **GenerateTicketData::GetTotal**
    q. **GenerateTicketData::GetTypePayment**
    r. **GenerateTicketData::GetCustomer**
    s. **GenerateTicketData::GetCustomerDescription**
    t. **GenerateTicketData::GetValueTributtes**
    u. **GenerateTicketData::GetSatDateTime**
    v. **GenerateTicketData::GetQuickResponseCode**
    w. **GenerateTicketData::GetComments**
    x. **IDeviceContext::Create**
    y. **IDeviceContext::PreparePage**
    z. **IDeviceContext::Print**
    aa. **IDeviceContext::Close**
    bb. **IDeviceContext::SetDriverName**
    cc. **IDeviceContext::SetPrinterMode**
3. **Descrição dos typedef em safemode**
4. **Dependencias**
5. **Configuração necessária**
6. **Recomendações**
7. **Exemplos**

Descrição das classes

Classe **ThermalPrinter** e principal para se iniciar o processamento do arquivo XML, geração do QRCode Bitmap, geração do ticket e impressão do mesmo. Mas e possivél reconfigurar a ordem de geração deste ticket assim como será descrito a seguir.

Exemplo de ordem completa:

```
_void main()
{
	try
	{
		LPThermalPrinter p = LPThermalPrinter(new ThermalPrinter);
		//
		p->SearchingPrinter();
		p->ProcessXMLFile("/sources/xmlfile.xml");
		p->ProcessTicketData();
		p->Write();
	}
	catch(std::exception const & e)
	{
	}
}
```

Exemplo de ordem simples:

```
_void main()
{
	try
	{
		LPThermalPrinter p = LPThermalPrinter(new ThermalPrinter);
		//
		p->Build("/sources/xmlfile.xml");
	}
	catch(std::exception const & e)
	{
	}
}
```

Todas as classes devem iniciadas como no modelo acima ou utilizar o shared_ptr safe thread mode.
Classe **GenerateTicketData** e responsavél por gerar a ordem do ticket de impressão ja processado e dados inseridos na estrutura de dados definida, com formatação identifica ao do xml.
Classe **ExtensibleMarkupLanguage** e responsavél por processar o arquivo xml e gerar os dados em ordem da estrutura de dados de mesma formatação do xml.
Classe **IDeviceContext** e responsavél por processar dados gerados pela classe **GenerateTicketData** em forma de HashTable por ordem de inserção e geração impressão do tickert. Esta ordem quando modificada por geração informações fora da ordem correta.

ThermalPrinter::ThermalPrinter()
Classe responsavél por iniciar o processo de conversão do xml em cupom impresso.

virtual std::wstring const BuillQuickResponseCode(std::wstring const & szData,
int nVersion = 1, unsigned uLevel = 3, unsigned uEncode = 2, uPrescaler = 8)
metodo responsavél por gerar o qrcode imagem.
std::wstring szData - dados para geração do qrcode qualquer dado no formato de wide_string.
unsigned nVersion - padrão 1, mas pode ser 1 entre 4.
unsigned uLevel - padrão 3 0 nenhum, 1 medio, 2 alto, 3 super alto
unsigned uEncode - padrão 2 para 8-bit data
unsigned uPrescaler - padrão 8 deve sempre ser utilizado divisores pares.
Retona std::wstring const & com o nome e caminho do arquivo de mapa de bits do qrcode ou ***std::exception***

virtual bool WINAPI SearchingPrinter ()
pesquisa pela impressora se esta disponivél no sistema para impressão e gerar informações necessárias para conectar a impressora..

virtual bool WINAPI ProcessXMLFile(std::wstring const & szXMLFileName)
processo o arquivo xml e preenche a estrutura de dados CFe declarada com o mesmo formato do xml.
szXMLFileName - *nome do arquivo*
Retorna nome do arquivo do qrcode ou ***std::exception.***

virtual bool WINAPI IsReady()
verifica se a impressora esta conectada,mas somente deve ser chamada após o metodo ***SearchingPrinter****, nunca antes.*
Retona true ou false (true impressora detectada e pronta para impressão, false impressora não esta pronta) ou ***std::exception.***

virtual bool WINAPI Write()
identifica a impressora se suportada pelo layer e imprime o cupom.
Retorna true processo efetuado com sucesso ou false processo não efetuado, ou ***std::exception***

Virtual bool WINAPI Build(std::wstring const & szXMLFileName, DWORD dwDriver)
executa todos os processos em sequencia.
szXMLFileName - nome do arquivo a ser processado
dwDriver - identificação do metedo a ser utilizado.
Valores:
pthermal::printer::mode::VirtualUniversalSerialBus - não suportado qrcode no momento
pthermal::printer::mode::WindowsDriver - gera impressão via driver de spool da impressora
pthermal::printer::mode::DynamicLibraryPrinter - não suportado erros de carga da DLL.
pthermal::printer::mode::WindowsSpool - gera impressão via default printer no sistema.
pthermal::printer::mode::UseXPSPrinter - gera impressão via arquivo XPS

Classe **ExtensibleMarkupLanguage** responsavél à processar o arquivo xml e gerar estrutura de dados no mesmo formato do xml, na estrutura CFe declarada no arquivo pthermal.hpp.
Dependencias - *MSXML60.DLL*

Explicit ExtensibleMarkupLanguage (std::wstring const & szFileName)
Gera endereço de entrada para processamento do arquivo xml.

virtual void Load()
Este metodo não tem retorno, caso ocorra falha no processo retorna ***std::exception.***

virtual void Parse()
Este metodo não tem retorno, caso ocorra falha no processo retorna ***std::exception.***

Class **GenerateTicketData,** responsavél à gerar o modelo do cupom separado pelos seguintes metodos os quais são executados internamente.
Exemplo do metodo Build:

```
LPITicketDataFieldQueue GenerateTicketData::Build()
{
LPITicketDataFieldQueue                    p=LPITicketDataFieldQueue(new
std::vector<LPITicketDataField> ( ) );
  //
  p->push_back ( this->GetHeader ( ) );
  p->push_back ( this->GetPDV ( ) );
  p->push_back ( this->GetExtract ( ) );
```

```
  p->push_back ( this->GetListOfProductHeader ( ) );
  p->push_back ( this->GetProductItems ( ) );
  p->push_back ( this->GetTotal ( ) );
  p->push_back ( this->GetTypePayment ( ) );
  p->push_back ( this->GetCustomer ( ) );
  p->push_back ( this->GetCustomerDescription ( ) );
  p->push_back ( this->GetValueTributes ( ) );
  p->push_back ( this->GetSatDateTime ( ) );
  p->push_back ( this->GetQuickResponseCode ( ) );
  p->push_back ( this->GetComments ( ) );
  //
  return p;
}
```

O cupom esta divido por partes conforme descrito no exemplo acima.

virtual LPITicketDataField GetHeader()
Gera informações de cabeçalho do cupom.

virtual LPITicketDataField GetPDV()
Gera informações data e hora retirado do campo dEmi e hEmi.

virtual LPITicketDataField GetExtract ( );
Gera informações do campo ide.cNF

virtual LPITicketDataField GetListOfProductHeader ( );
Gera cabeçalho de produto.

virtual LPITicketDataField GetProductItems ( );
Gera lista de produtos.

virtual LPITicketDataField GetTotal ( );
Gera campo de totais.

virtual LPITicketDataField GetTypePayment ( );
Gera tipo de pagamento.

virtual LPITicketDataField GetCustomer ( );
Gera informações do cliente.

virtual LPITicketDataField GetCustomerDescription ( );
Gera ifnromações de contribuinte.

virtual LPITicketDataField GetValueTributes ( );
Gera informações de tributos.

virtual LPITicketDataField GetSatDateTime ( );
Gera número do SAT e data e hora.

virtual LPITicketDataField GetQuickResponseCode ( );
Gera informações para montagem do qrcode.

virtual LPITicketDataField GetComments ( );
Adiciona comentarios ao cupom.

virtual LPITicketDataFieldQueue Build ( );
Gera HashTable dos campos a processados pela class ***IDeviceContext*** *para geração do cupom.*

virtual void SetPaperWidth(unsigned uPaperWidth);
Define largura da pagina de impressão para alinhamento dos dados.

Class **IDeviceContext** responsavél por processar os dados gerados pela classe **GenerateTicketData,** e imprimo-los conforme largura e altura da pagina de impressora gerando um disposito em memoria e depois enviados para controlador de impressora do windows.

explicit IDeviceContext(pthermal::LPIUniversalSerialBusData const & pBus)
pBus - *ponteiro gerado no modo safe thread que contêm todas informações necessarias para processamento dos dados do cupom e impressora.*

virtual bool Create(pthermal::LPXMLCFeData const & pcfe)
Cria informações do cupom e prepara os dados para ser inseridos no contexto do dispositivo a imprimir, não deve ser sobrescrita.
Retorna true, false ou **std::exception**

virtual bool PreparePage()
Este metodo prepara a pagina de impressão onde pode ser sobre escrita para gera novas informações e novos formatos de cupom.

virtual bool Print()
Este metodo formata o cupom conforme os comandos inseridos na HashTable gerado pela classe ***GenerateTicketData,*** *e coloca no device context.*

virtual void Close()
Finaliza o processo e fecha todos os manipuladores abertos no processo. ***este metodo não deve ser chamado em nenhum momento a propria classe faz isto automaticamente.***

virtual void SetDriverName(std::wstring const & szDriver)
Define qual driver de impressão será utilizado, ***não e necessário chamar esta função.***

virtual void SetPrinterMode(std::wstring const & szMode)
Define qual modelo de impressão será utilizado, ***não e necessario chamar esta função.***

Descrição dos typedef em safemode
Todos os tipos com prefixo ***LP*** *são tipo safe.*

LPITicketData - estrutura de dados do cupom
typedef std::shared_ptr<pthermal::ITicketData>

LPITicketDataField - estrutura de dados do campo do cupom
typedef std::shared_ptr<pthermal::ITiketDataField>

LPITicketDataFieldQueue - vetor de dados do cupom
typedef std::shared_ptr<vector<LPITicketDataField>>

Dependencias
MSXML60.DLL
LIBUSB.LIB
GRGENERATOR.LIB
Drivers SPOOL de cada impressora instaladas.

Configuração necessária
Todos os drivers de impressoras no modo SPOOL instaladas.

Recomendações
Este Layer foi construido sobre VS2013 com STL e shared memory safe em todos os ponteiros.
Na inserção de campos da classe **GenerateTicketData,** não deve ser colocado nos campos de dados o comando **NextLine.**
Use sempre comandos com namespace escpos.
GenerateTicketData::replicate(<wide_string>, <comando>) -> std::wstring

Exemplos:
Gerando dados em cupom.

```
LPITicketDataField p = LPITIcketDataField(new ITicketFieldData);
//
p->Data->ulFlags = DataType::Text;

p->push_back(FieldData(L"", Print::NextLine)); //pula uma linha
p->push_back(FieldData("Descrição", escpos::Position::Center)); //coloca o texto centralizado.
p->psuh_back(FieldData(*replicate('-', 48), escpos::Position::Left)); //coloca uma linha a
esquerda
```

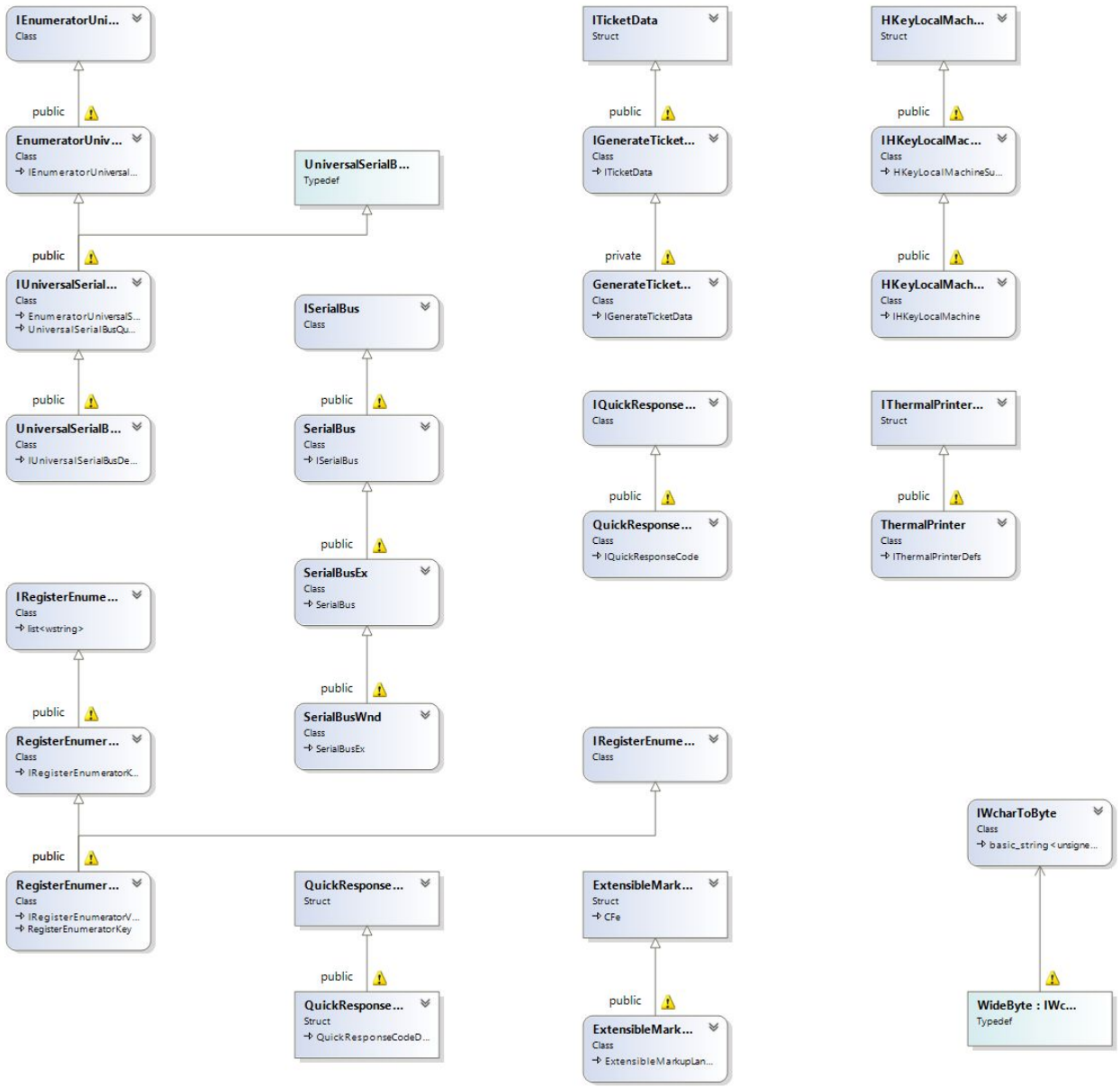
Sobrescrevendo o metodo PreparePage da classe IDeviceContext para gerar um novo cupom:

```
#pragma once
#pragma warning(disable:4275)
#pragma warning(disable:4251)

#incldue <IDeviceContext.h>
namespace device
{
class PTHERMAL_API MyIDeviceContext: public IDeviceContext
{
public:
        MyIDeviceContext(pthermal::LPIUniversalSerialBus const & pBus):
        IDeviceContext(pBus)
        {
        }
        virtual bool PreparePage();
};
}
namespace pthermal
{
class MyThermalPritner: ThermalPrinter
{
public:
        MyThermalPrinter():ThermalPrinter();
        virtual bool Write();
}
```

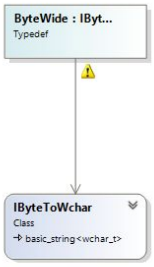
```
}


MyIDeviceContext.cpp
#include <GenerateTicketData.hpp>
#include <escpos.hpp>
#include <escbema.hpp>
#include <IWcharToChar.h>
using namespace escpos;
using namespace pthermal;

bool MyIDeviceContext::PreparePage()
{
IDeviceContext::PreparePage();
this->m_queue->clear();
//create new header
LPITicketDataField pNewHeader = LPITicketDataField(new ITicketDataField);
//
pNewHeader->Data->ulFlags = DataType::Text;
pNewHeader->push_back(FieldData(L"", Print::NextLine));
pNewHeader->push_back(FieldData(*replicate('-', 48), Position::Left));
pNewHeader->push_back(FieldData(L"My new cupom", Position::Center));
//insert in m_queue
m_queue->push_back(pNewHeader);
//etc etc
}

MyIThermapPrinter.cpp

#include <ThermalPrinter.h>
#include <ExtensibleMarkupLanguage.hpp>
#include <GenerateTicketData.h>
#include <MyIDeviceContext.h>
using namespace pthermal;

MyThermalPrinter::MyThermalPrinter()
{
        this->SearchingPrinter();
        this->ProcessXMLData("c:/sources/270220160929.xml");
        this->ProcessTicketData();
}

bool MyThermapPrinter::Write()
```

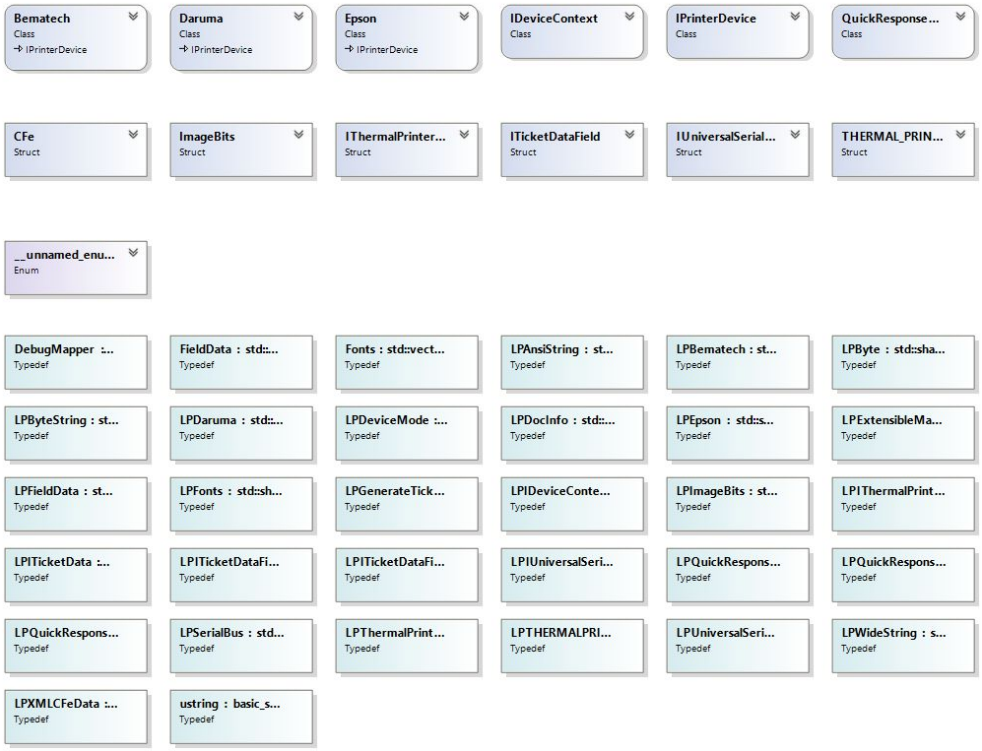
```
{
        std::shared_ptr<MyIDeviceContext> p = std::shared_ptr<MyIDeviceContext>(new
MyIDeviceContext(m_pHandle));
        p->Create(m_pXMLData);
        p->PreparePage();
        p->Print();
        return true;
}
```

Designer class




IWcharToChar
Class
basic_string<char>
WideChar : IWc...
Typedef


ByteWide : IByt...
Typedef
IByteToWchar
Class
basic_string<wchar_t>


Bematech
Class
IPrinterDevice
Daruma
Class
IPrinterDevice
Epson
Class
IPrinterDevice
IDeviceContext
Class
IPrinterDevice
Class
QuickResponse...
Class
CFe
Struct
ImageBits
Struct
IThermalPrinter...
Struct
ITicketDataField
Struct
IUniversalSerial...
Struct
THERMAL_PRIN...
Struct
__unnamed_enu...
Enum
DebugMapper :...
Typedef
FieldData : std::...
Typedef
Fonts : std::vect...
Typedef
LPAnsiString : st...
Typedef
LPBematech : st...
Typedef
LPByte : std::sha...
Typedef
LPByteString : st...
Typedef
LPDaruma : std::...
Typedef
LPDeviceMode :...
Typedef
LPDocInfo : std::...
Typedef
LPEpson : std::s...
Typedef
LPExtensibleMa...
Typedef
LPFieldData : st...
Typedef
LPFonts : std::sh...
Typedef
LPGenerateTick...
Typedef
LPIDeviceConte...
Typedef
LPImageBits : st...
Typedef
LPIThermalPrint...
Typedef
LPITicketData :...
Typedef
LPITicketDataFi...
Typedef
LPITicketDataFi...
Typedef
LPIUniversalSeri...
Typedef
LPQuickRespons...
Typedef
LPQuickRespons...
Typedef
LPQuickRespons...
Typedef
LPSerialBus : std...
Typedef
LPThermalPrint...
Typedef
LPTHERMALPRI...
Typedef
LPUniversalSeri...
Typedef
LPWideString : s...
Typedef
LPXMLCFeData :...
Typedef
ustring : basic_s...
Typedef