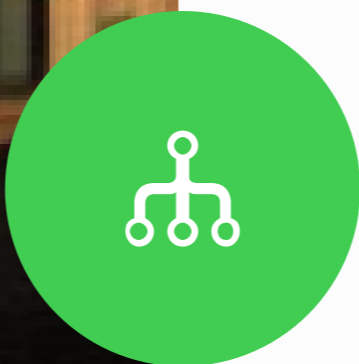


Qt



Qt Plugins



Filipe Saraiva
filipe@kde.org



Filipe Saraiva

Professor na UFPA
Desenvolvedor no KDE



filipe@kde.org

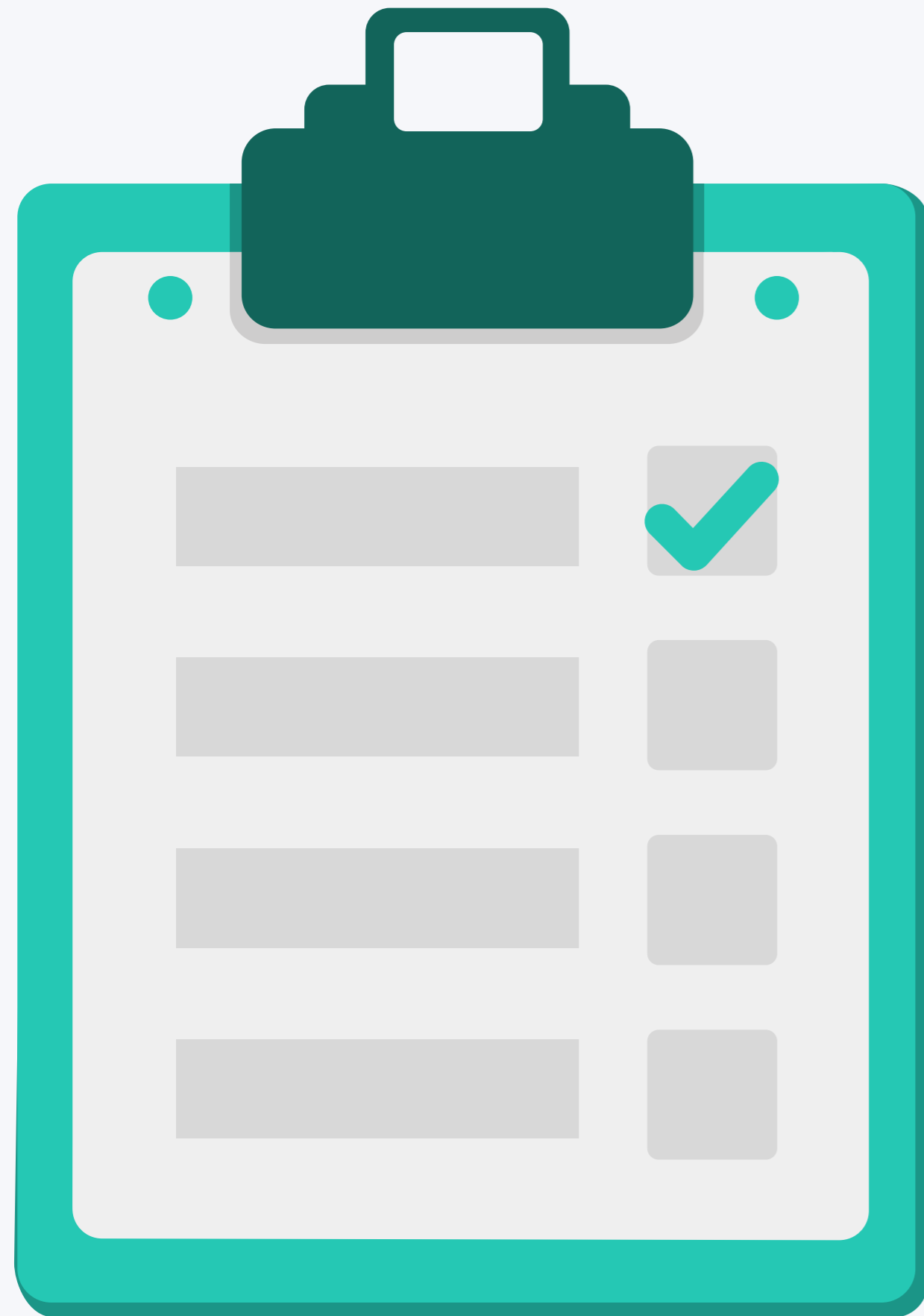


[@filipesaraiva](https://twitter.com/filipesaraiva)



filipesaraiva.info

Agenda



- 1 Para que plugins?
- 2 Plugins no Qt
- 3 Plugins no KDE Frameworks 5
- 4 Conclusões

1

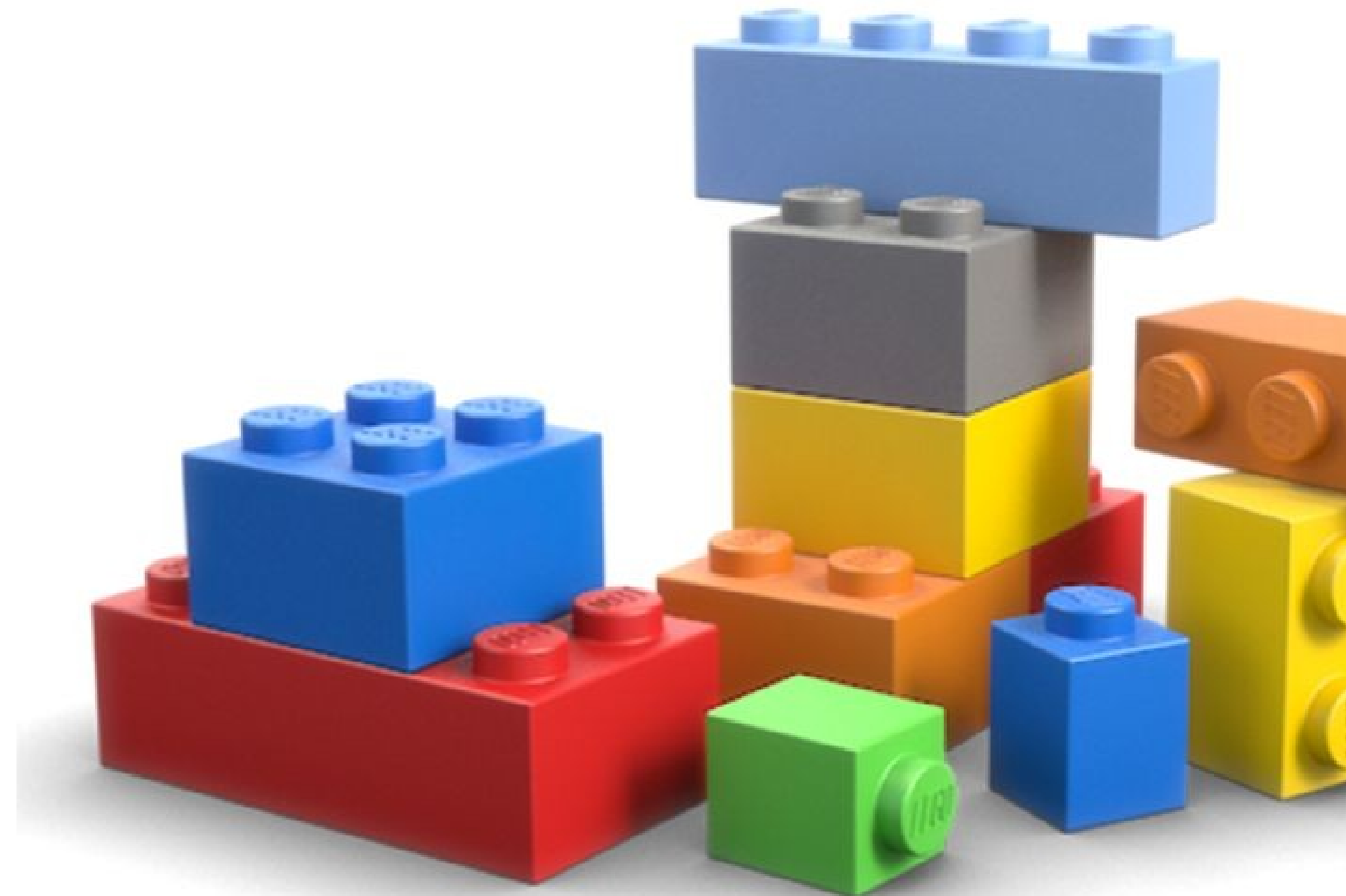
Para que
plugins?

Para que plugins?

...

No desenvolvimento de software, muitas vezes queremos que eles possam ser extensíveis através da implementação de novas funcionalidades.

No geral, a depender dos objetivos, uma arquitetura de plugins é interessante para um projeto.

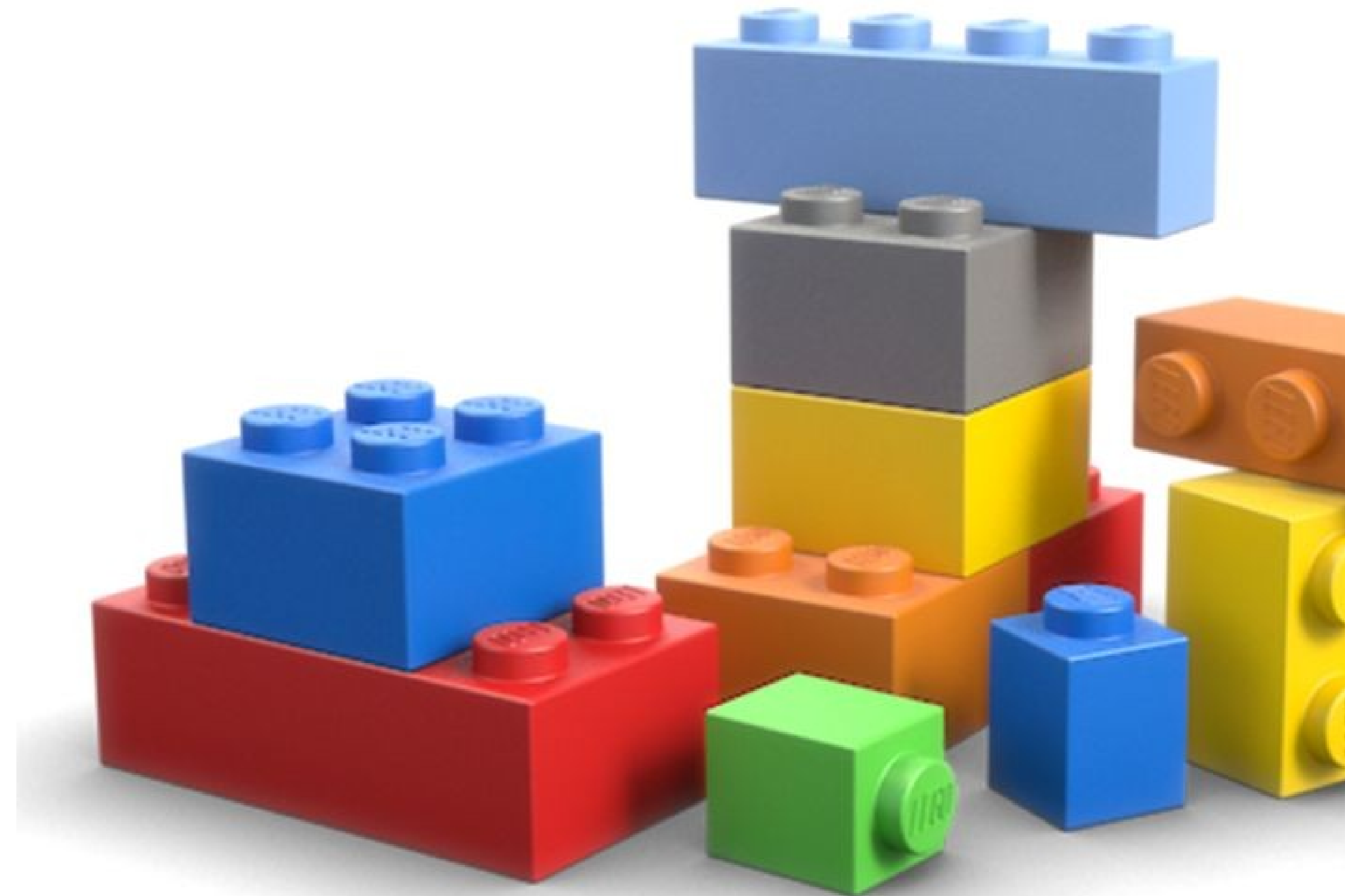


Para que plugins?

...

O que são plugins?

Plugins são binários carregados pela aplicação principal em tempo de execução.

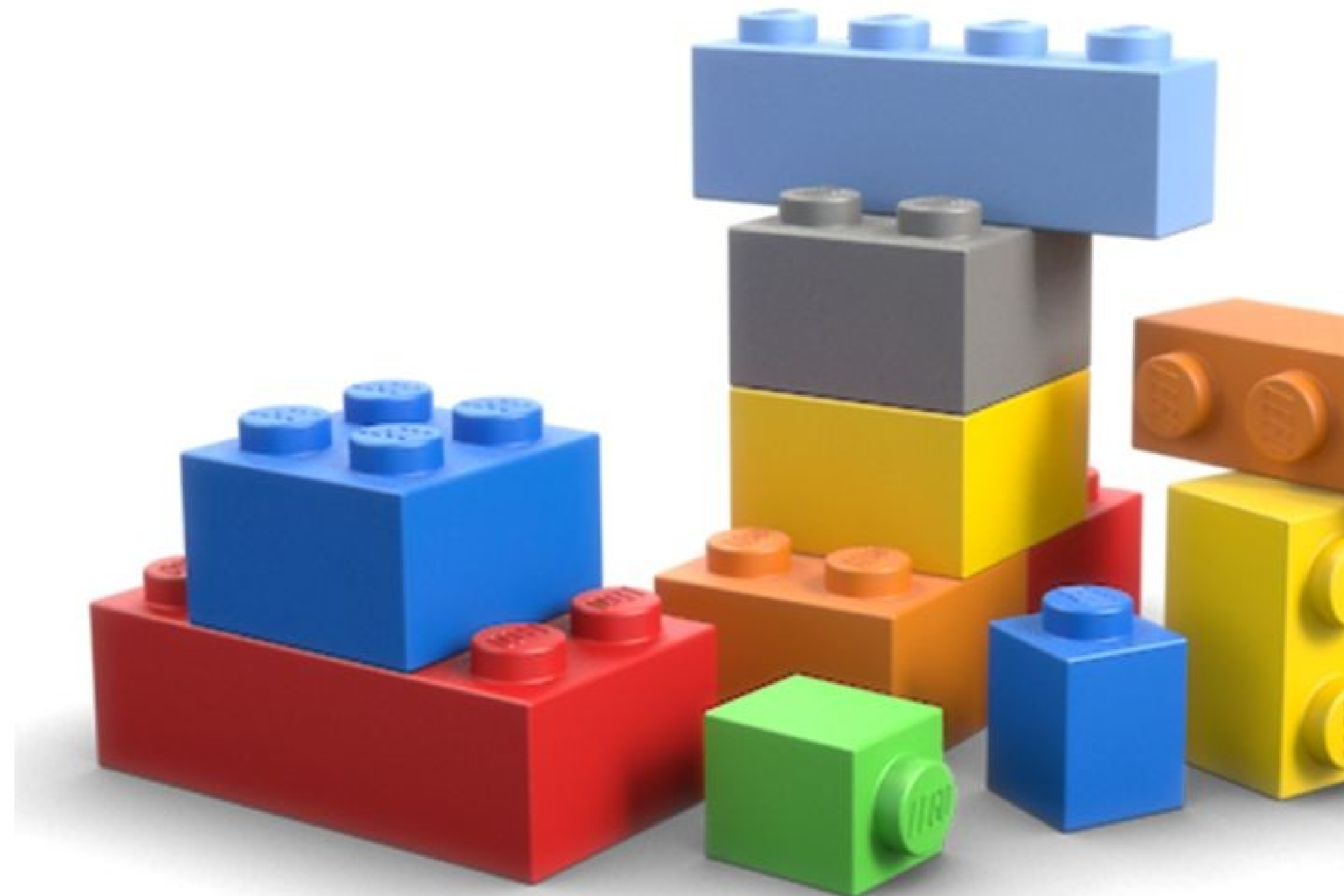


Para que plugins?

...

Alguns motivos para se utilizar plugins:

- Estender funcionalidades sem precisar conhecer o *core*;
- Prover suporte a novos *backends* para funcionalidades já existentes;
- Reduzir o tamanho de uma aplicação;
- *Bypass* de licenças de software;
- Permitir a criação de um ecossistema em torno de um projeto.

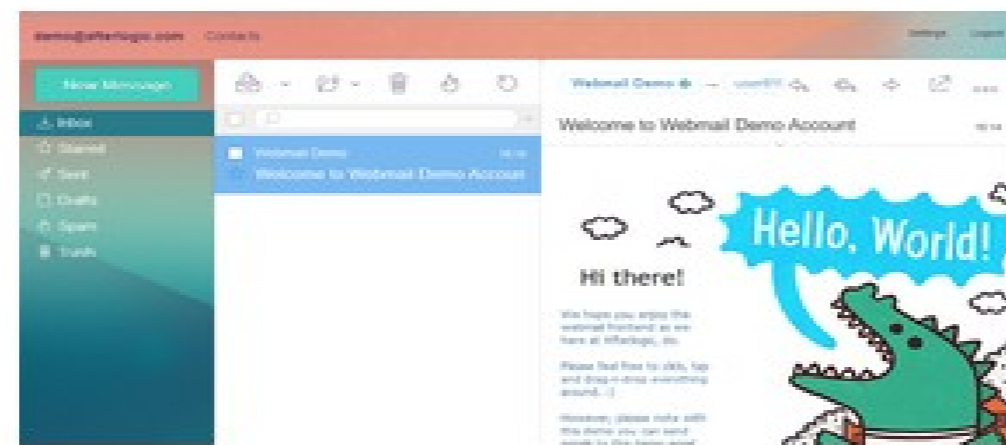


Para que plugins?



- Seus aplicativos
- Ativar aplicativos
- Aplicativos desabilitados
- Atualizações
- Pacotes de aplicativos
- Personalizar
- Ficheiros
- Jogos
- Integração**
- Monitorização
- Multimédia
- Escritório & texto
- Organização
- Pesquisar
- Segurança
- Social & comunicação

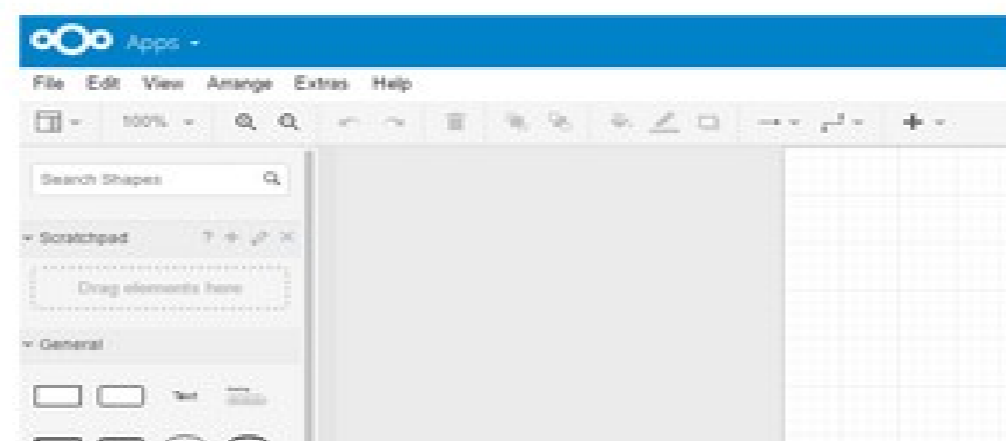
5



AfterLogic
Integration with AfterLogic WebMail client



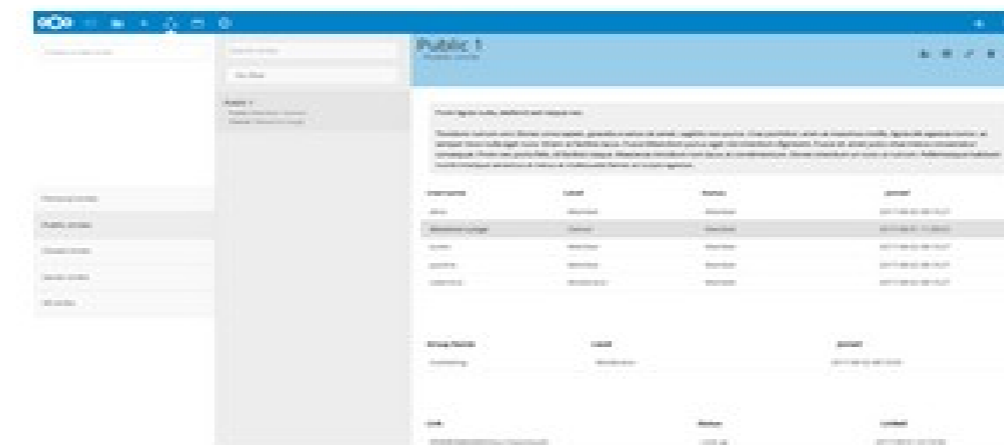
Baixar e ativar



Draw.io
Draw.io integration app



Baixar e ativar



Circles
Bring cloud-users closer together.



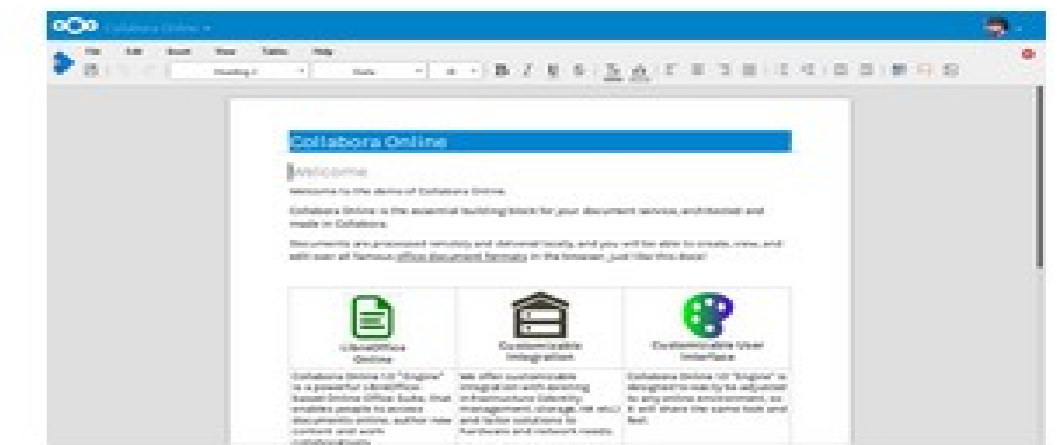
Baixar e ativar



External sites
Add external sites to your Nextcloud navigation



Baixar e ativar



Collabora Online
Edit office documents directly in your browser.



Baixar e ativar



ONLYOFFICE
ONLYOFFICE connector



Baixar e ativar

Para que plugins?

WordPress interface showing the "Instalar plugins" (Install plugins) page. The page displays a list of recommended plugins with their icons, names, descriptions, and installation options.







Top navigation: Filipe Saraiva's blog, 0 comments, + Novo, Olá, Filipe Saraiva, Ajuda

Section: Instalar plugins [Enviar plugin](#)

Filters: Destaques, Populares, Recomendado, Favoritos

Search: Palavra-chave

Plugins estendem e expandem a funcionalidade do WordPress. Você pode instalar automaticamente plugins do [Diretório de plugins do WordPress](#) ou enviar um plugin no formato .zip clicando no botão no topo desta página.

Plugin Name	Rating	Active Installations	Last Update	Compatibility	Action
 BuddyPress O BuddyPress ajuda construtores de site e desenvolvedores a adicionar funcionalidades de comunidade a seus... <i>Por The BuddyPress Community</i>	★★★★☆ (314)	200.000+ instalações ativas	Última atualização: 2 meses atrás	✓ Compatível com essa versão do WordPress	Instalar agora Mais detalhes
 Health Check & Troubleshooting Este plugin realiza algumas verificações na sua instalação do WordPress para detectar erros comuns de... <i>Por The WordPress.org community</i>	★★★★☆ (80)	60.000+ instalações ativas	Última atualização: 4 meses atrás	✓ Compatível com essa versão do WordPress	Instalar agora Mais detalhes
 bbPress bbPress é um fórum de software, feito à maneira WordPress. <i>Por The bbPress Community</i>	★★★★☆ (290)	300.000+ instalações ativas	Última atualização: 4 semanas atrás	✓ Compatível com essa versão do WordPress	Instalar agora Mais detalhes
 Editor Clássico Restores the Editor Clássico and the old-style Edit Post screen layout (TinyMCE, Meta boxes, etc.)... <i>Por WordPress Contributors</i>	★★★★★ (69)	500.000+ instalações ativas	Última atualização: 2 semanas atrás	✓ Compatível com essa versão do WordPress	Instalar agora Mais detalhes
 Jetpack por WordPress.com				Ativo	
 Gutenberg					Instalar agora

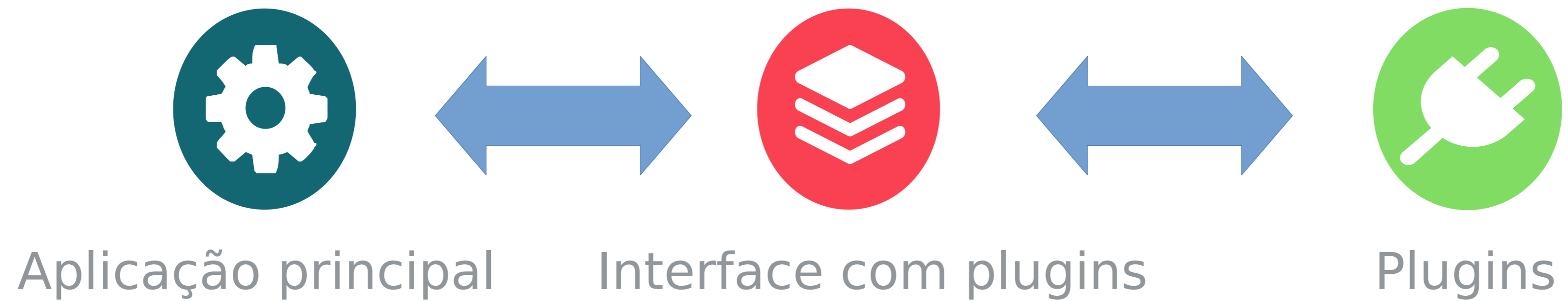
Para que plugins?

...



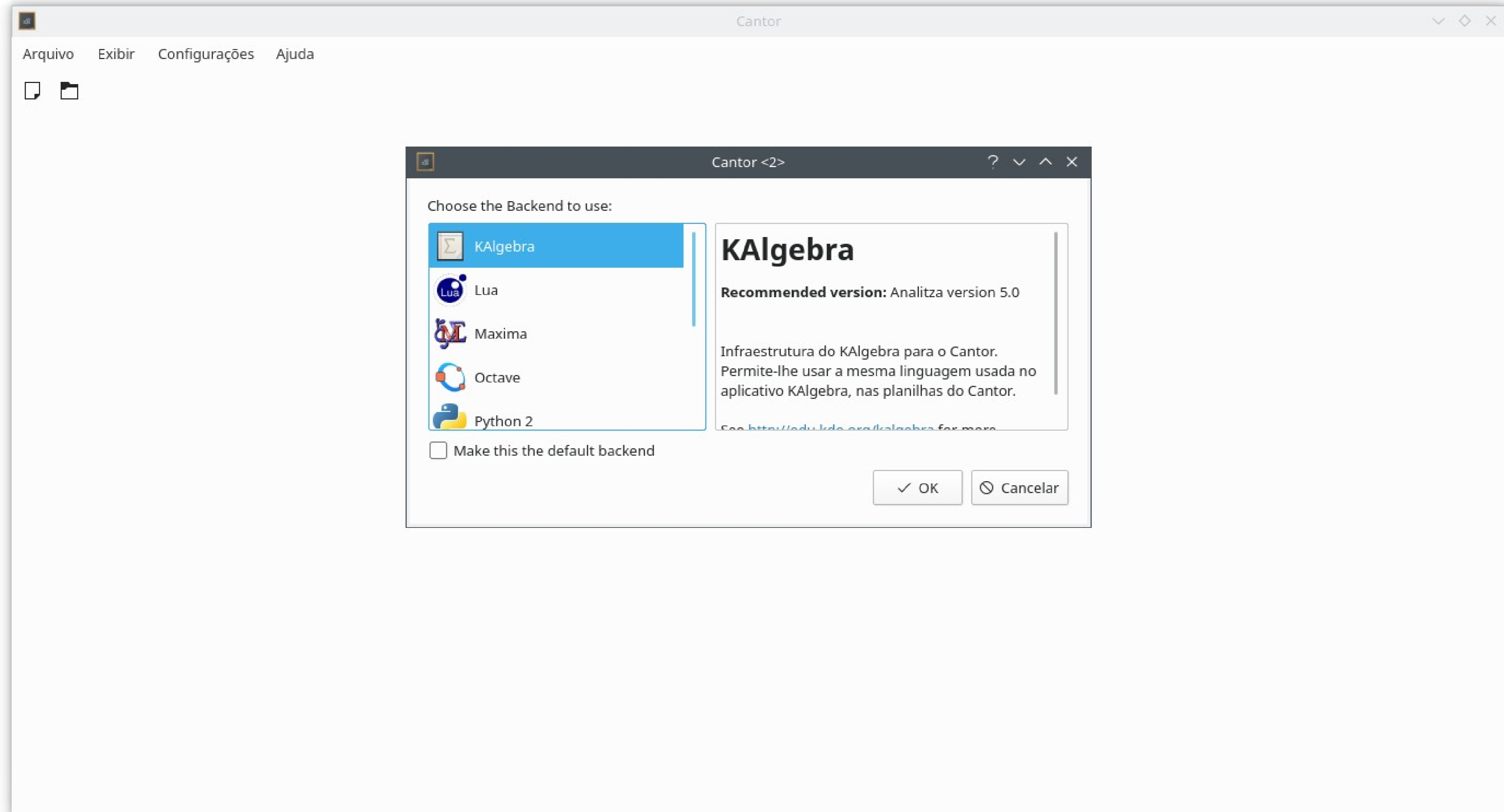
Para que plugins?

...



Para que plugins?

...



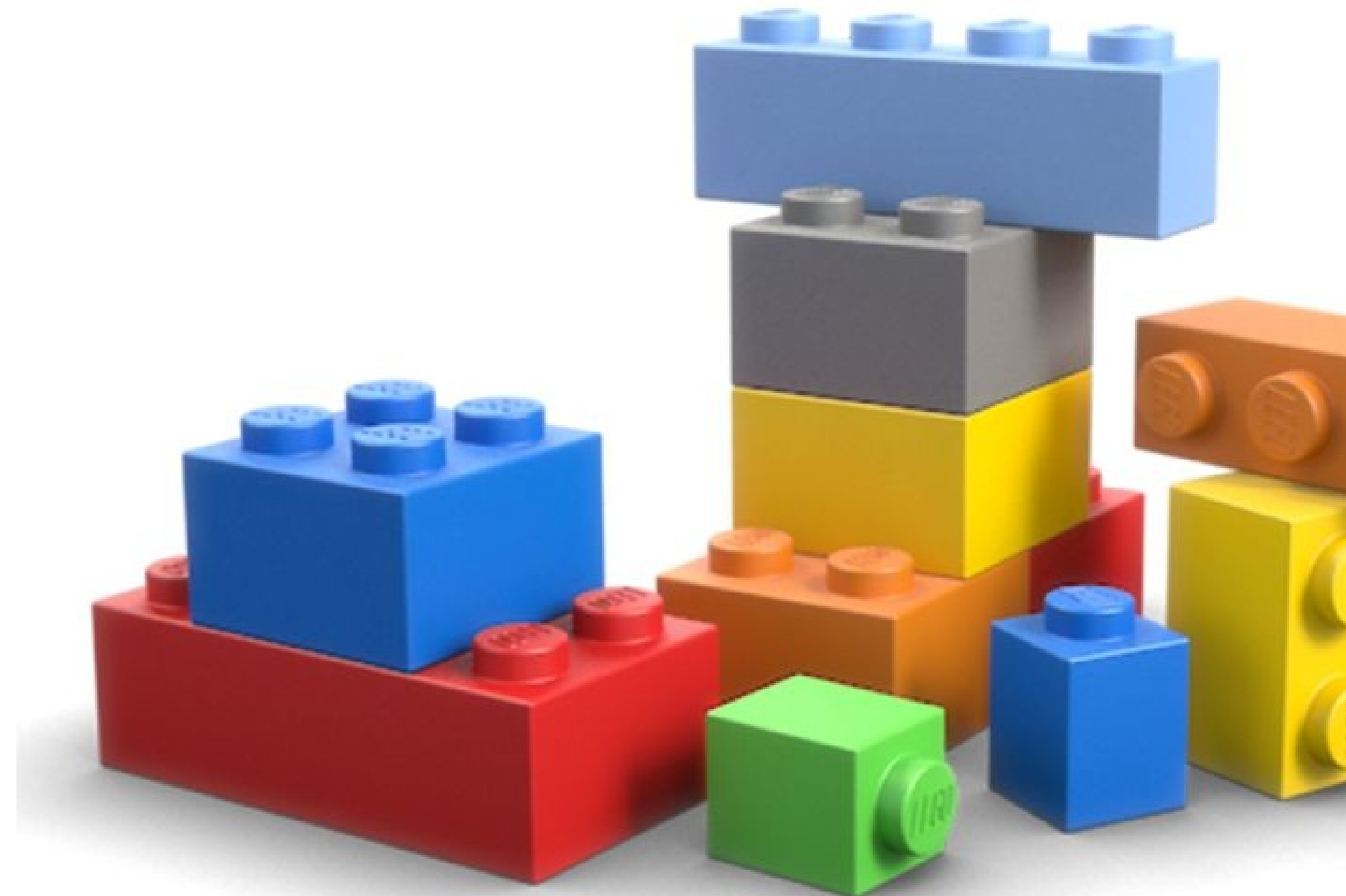
2

Plugins no Qt

Plugins no Qt

...

Qt tem um sistema de plugins que permite rápida implementação desse tipo de arquitetura.



Plugins no Qt

...

No lado da aplicação principal, é necessário:

- Definir as interfaces;
- Usar `Q_DECLARE_INTERFACE()`;
- Carregar plugins com `QPluginLoader`;
- Testar plugins com `qobject_cast()`.



Aplicação principal



Interface com plugins

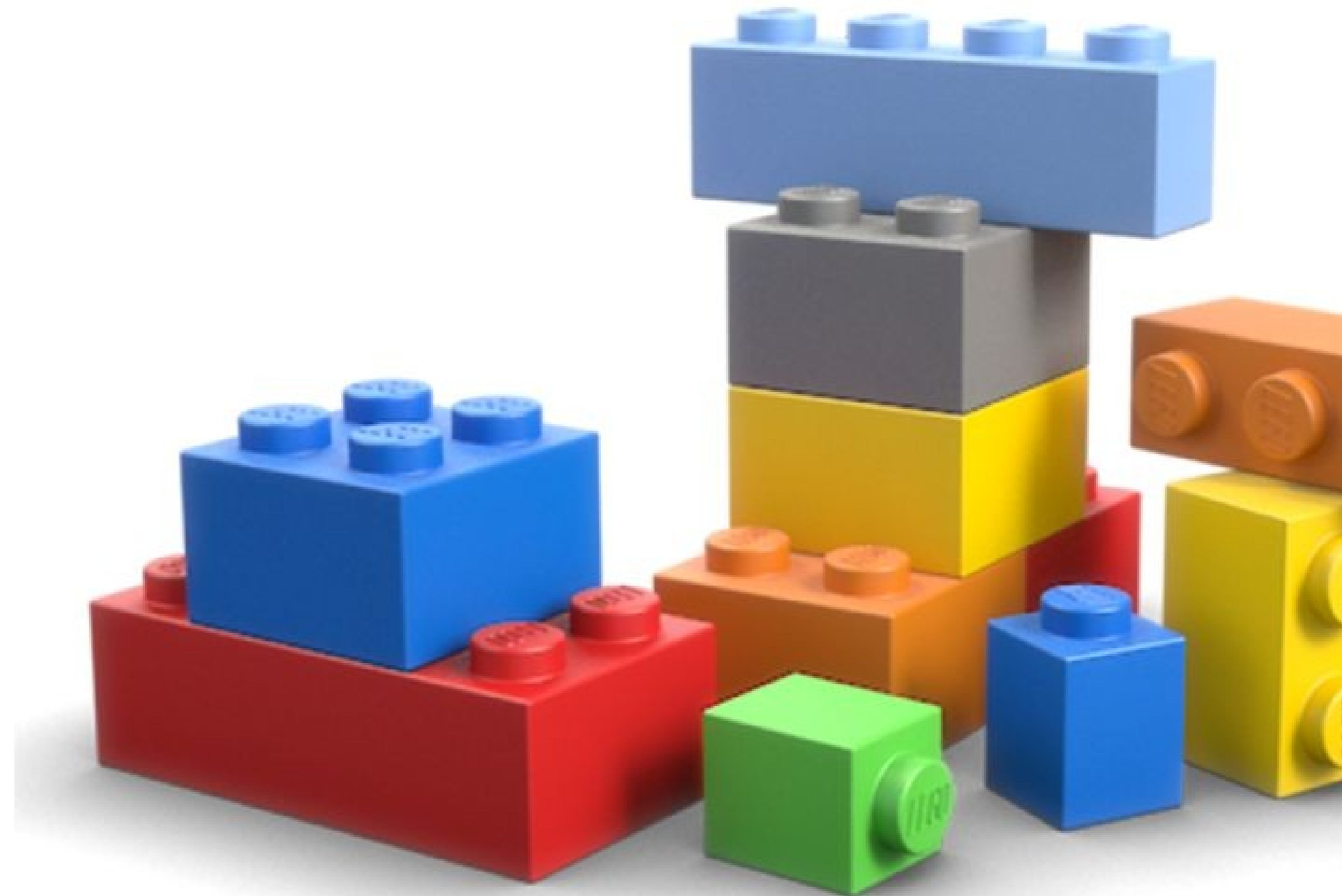
Plugins no Qt

...

Definir as interfaces

Começamos a implementar o suporte a plugins escrevendo no lado da aplicação principal um conjunto de interfaces que deverão ser estendidas e implementadas nos plugins.

As interfaces são classes abstratas com métodos virtuais em C++.



Plugins no Qt

...

...

```
/**
```

```
* @brief The IFirmware class
```

```
* Base Class for Firmware Plugins
```

```
*/
```

```
class ATCORE_EXPORT IFirmware : public QObject
```

```
{
```

```
    Q_OBJECT
```

```
    Q_PROPERTY(QString name READ name)
```

```
    Q_PROPERTY(bool sdSupport READ isSdSupported)
```

```
...
```

Exemplo em [atcore/src/core/ifirmware.h](#)

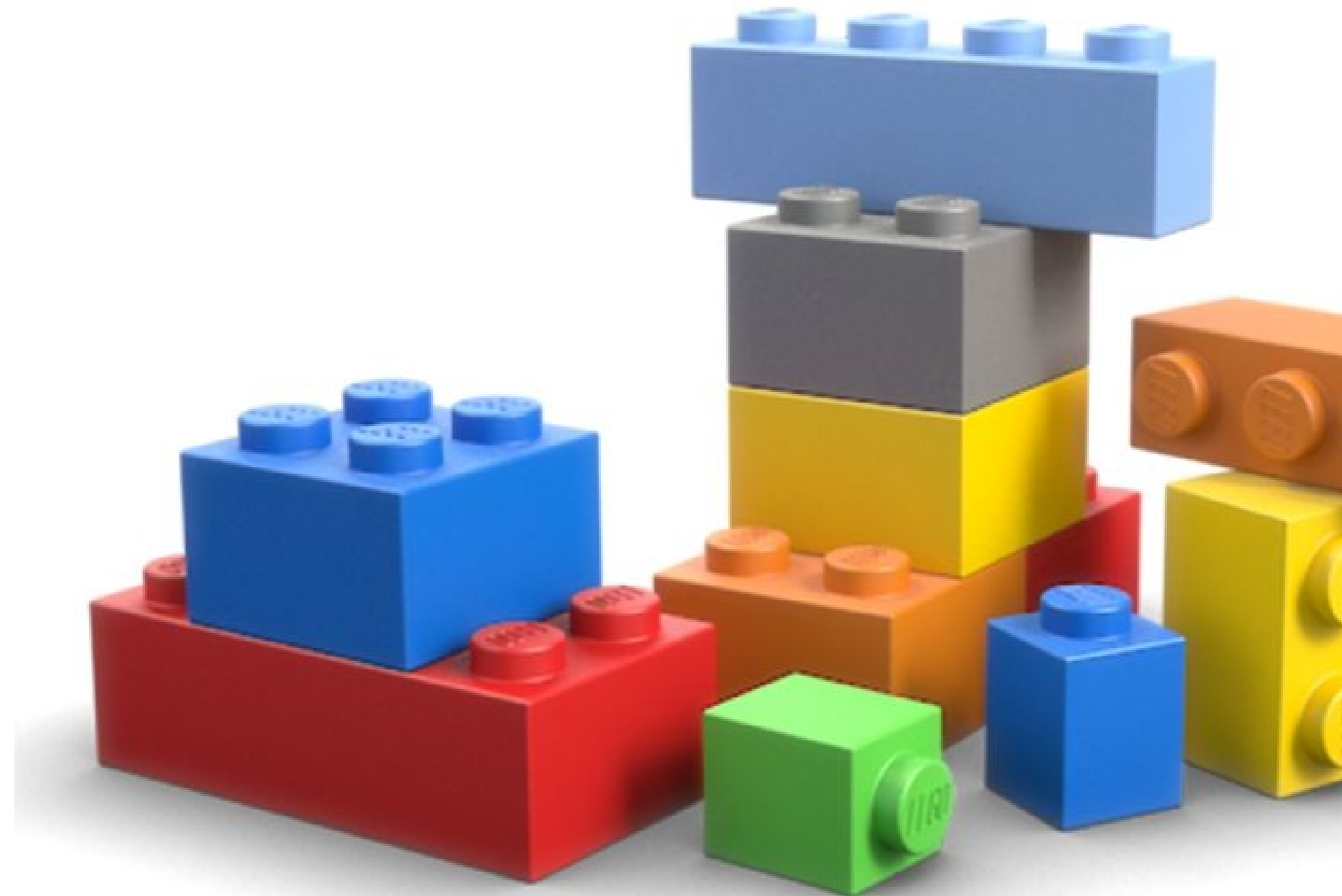
Plugins no Qt

...

Usar `Q_DECLARE_INTERFACE()`

Essa macro serve para falar ao moc que uma classe será declarada como interface e, portanto, poderá ser utilizada para implementações de plugins.

Os argumentos são a classe e um identificador único.



Plugins no Qt

...

...

signals:

```
/**
```

```
 * @brief emit when firmware is ready for a command
```

```
 */
```

```
void readyForCommand(void);
```

```
};
```

```
Q_DECLARE_INTERFACE(IFirmware, "org.kde.atelier.core.firmware")
```

Exemplo em [atcore/src/core/ifirmware.h](#)

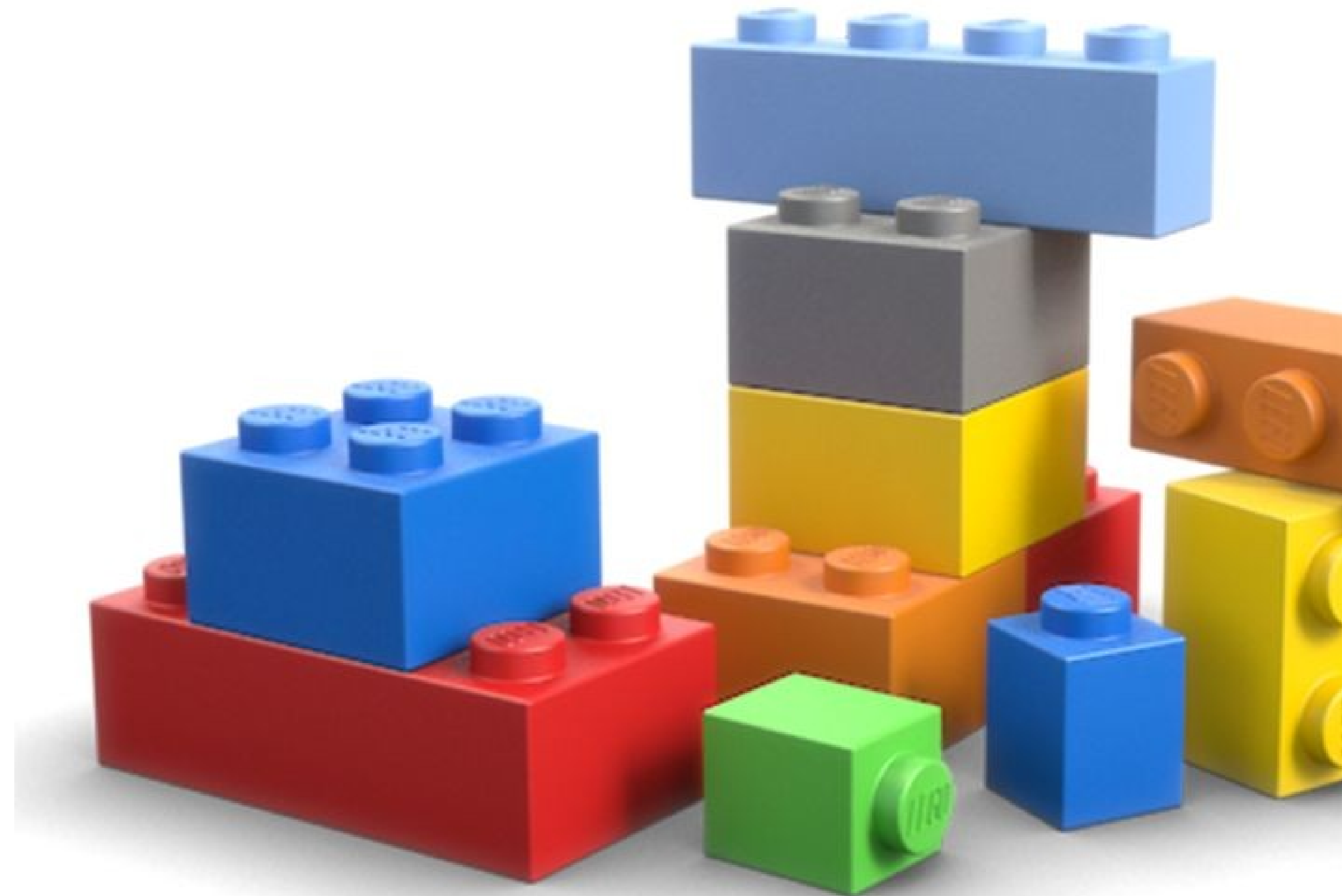
Plugins no Qt

...

Carregar plugins com QPluginLoader

QPluginLoader é a classe Qt responsável por carregar e instanciar os plugins.

No geral, os plugins residem em um mesmo diretório onde os arquivos são iterados e verificados se são plugins carregáveis ou não.



Plugins no Qt

...

...

```
void AtCore::loadFirmwarePlugin(const QString &fwName)
{
    qDebug(ATCORE_CORE) << "Loading plugin: " << d->plugins[fwName];
    if (d->plugins.contains(fwName)) {
        d->pluginLoader.setFileName(d->plugins[fwName]);
        if (!d->pluginLoader.load()) {
```

...

Exemplo em [atcore/src/core/atcore.cpp](#)

Plugins no Qt

...

...

```
foreach (QString fileName, pluginsDir.entryList(QDir::Files)) {  
    QPluginLoader loader(pluginsDir.absoluteFilePath(fileName));  
    QObject *plugin = loader.instance();  
    if (plugin) {  
        populateMenus(plugin);  
        pluginFileNames += fileName;  
    }  
}
```

...

Exemplo em [plugandpaint/app/mainwindow.cpp](#)

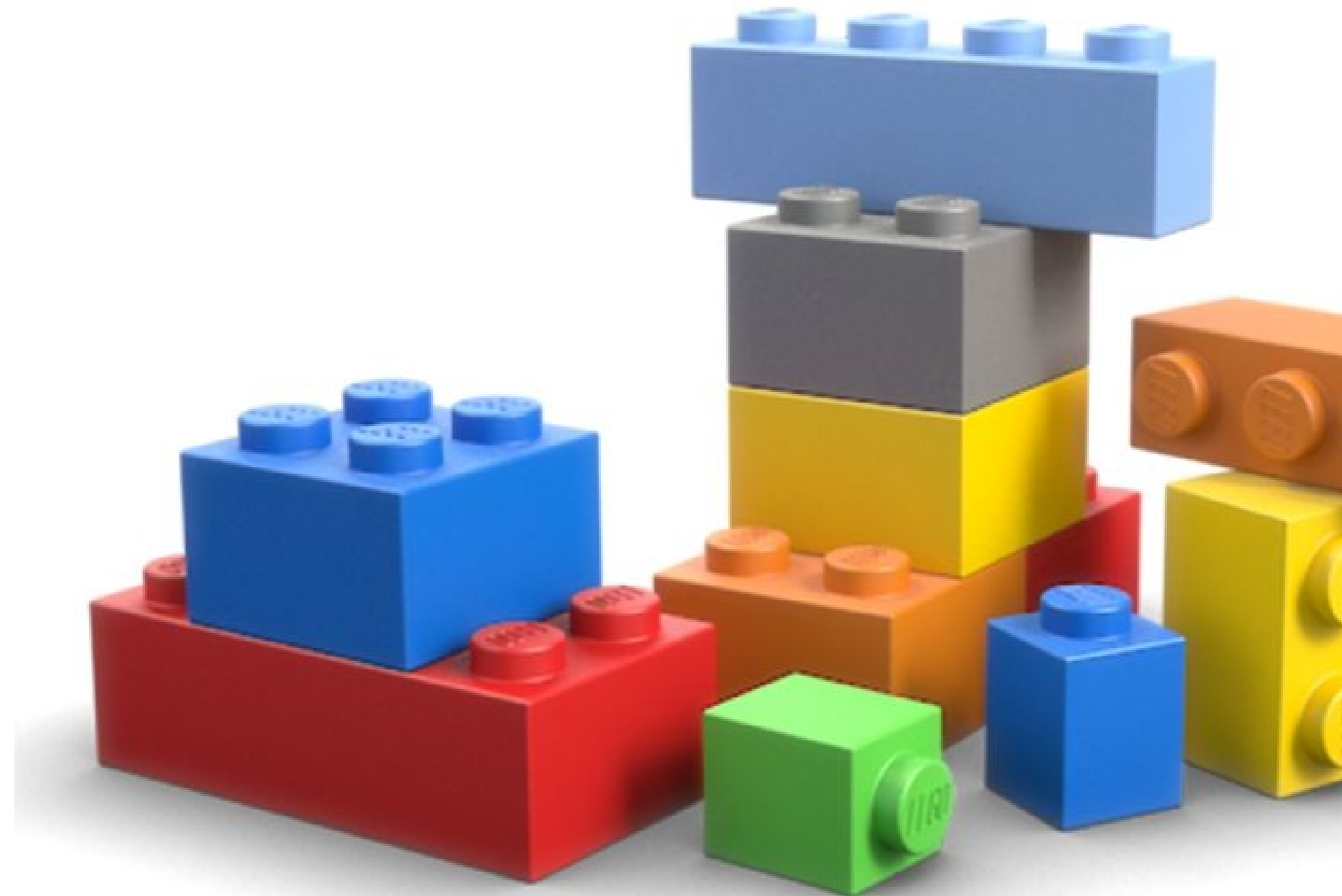
Plugins no Qt

...

Testar plugins com `qobject_cast()`

Utilizando `qobject_cast` é possível verificar se determinado plugin é uma instância da interface necessária para implementar um plugin.

Dessa forma é possível verificar se o plugin é válido ou não.



Plugins no Qt

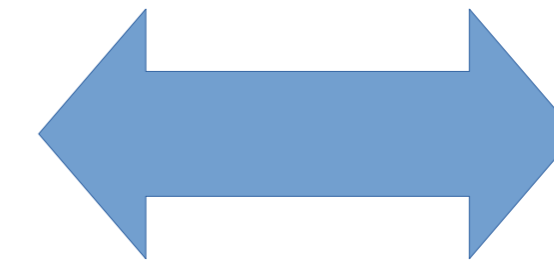
...

No lado do plugin, é necessário:

- Declarar uma classe do plugin que seja `QObject` e herde a interface;
- Usar `Q_INTERFACES()`;
- Exportar com `Q_PLUGIN_METADATA()`.



Interface com plugins



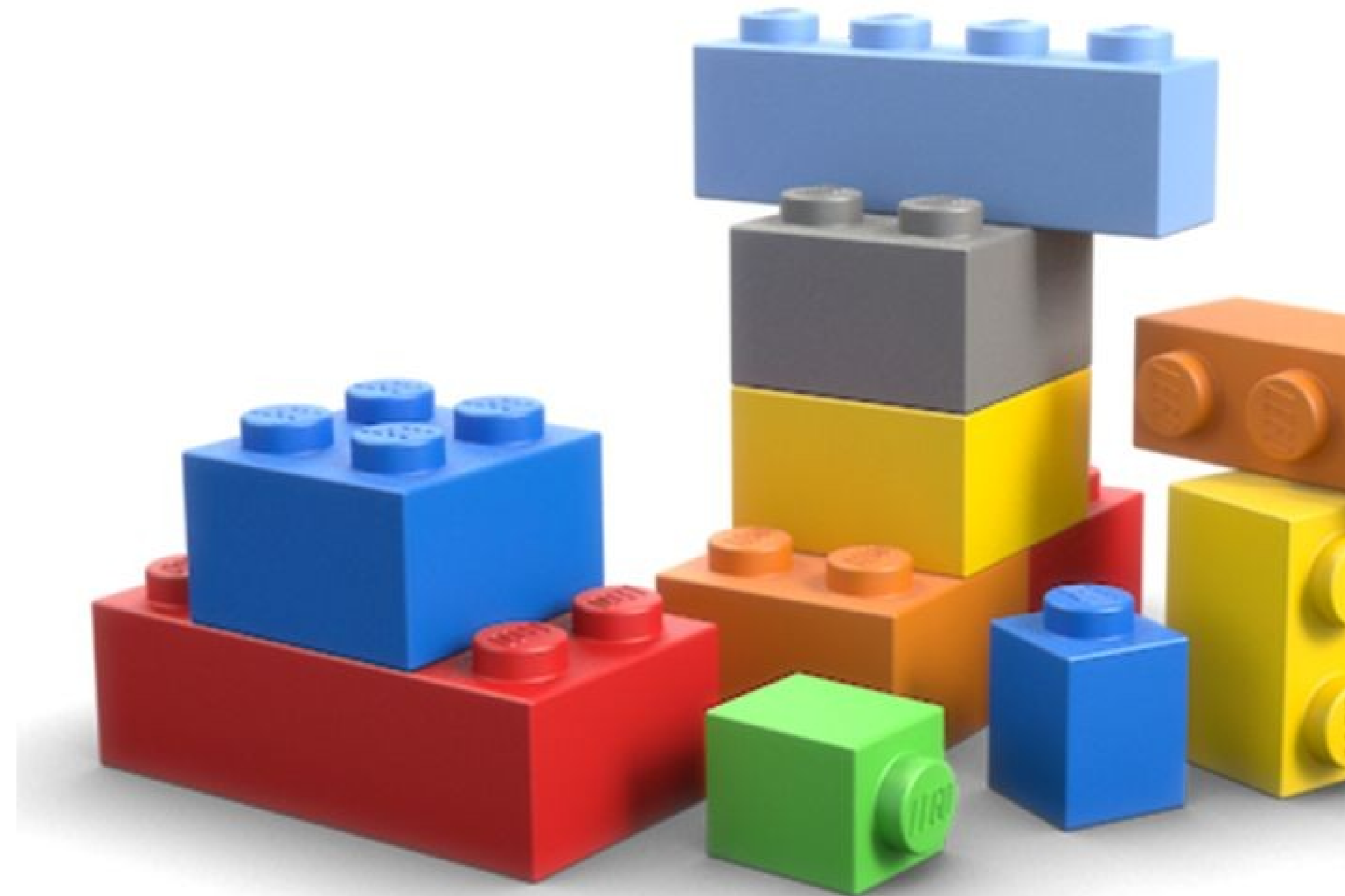
Plugins

Plugins no Qt

...

Declarar uma classe do plugin que seja QObject e herde a interface

Para criar o plugin, é necessário implementar a interface definida na aplicação principal e utilizar QObject.

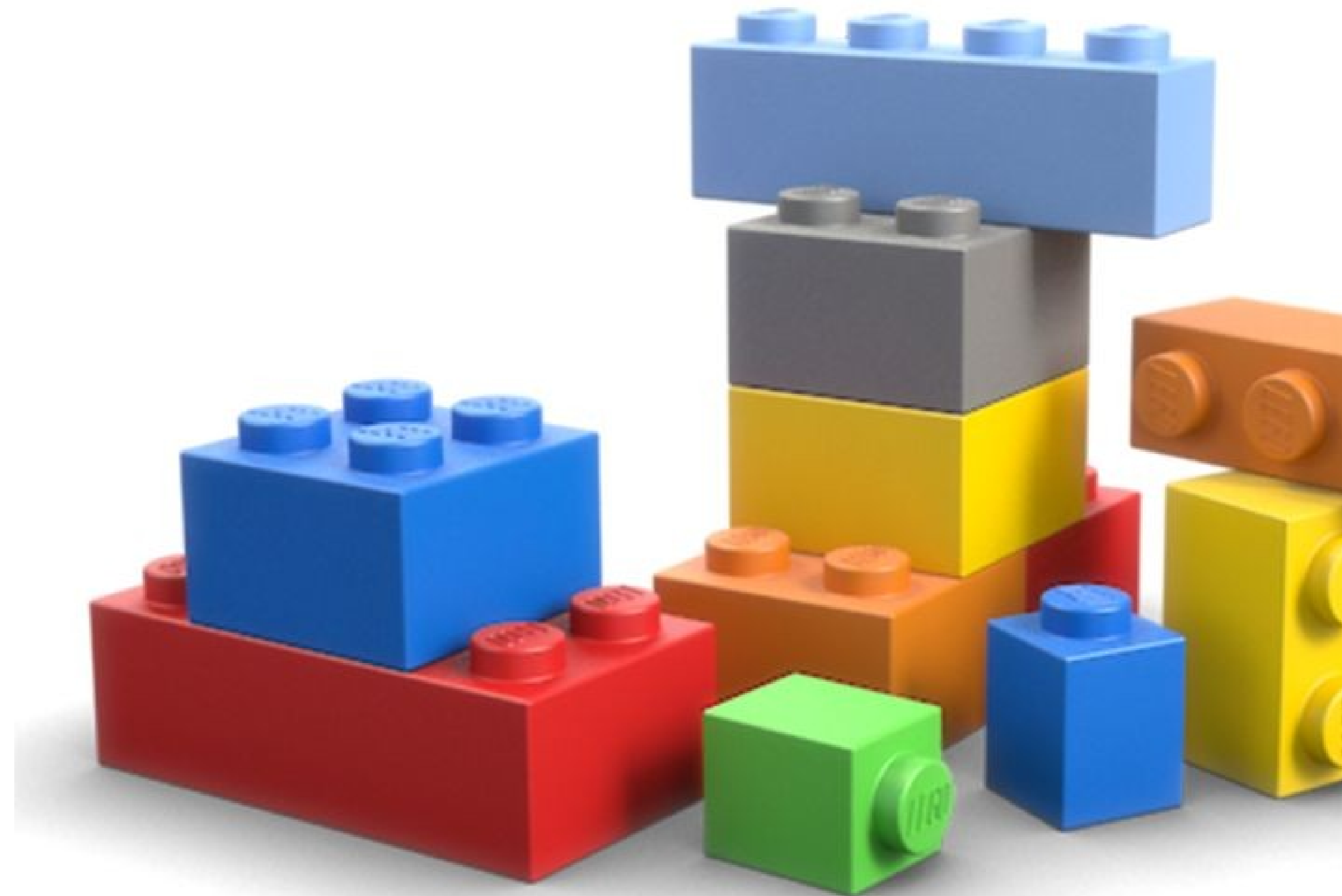


Plugins no Qt

...

Usar `Q_INTERFACES()`

`Q_INTERFACES()` serve para avisar ao moc que tipo de interface aquele plugin está implementando.



Plugins no Qt

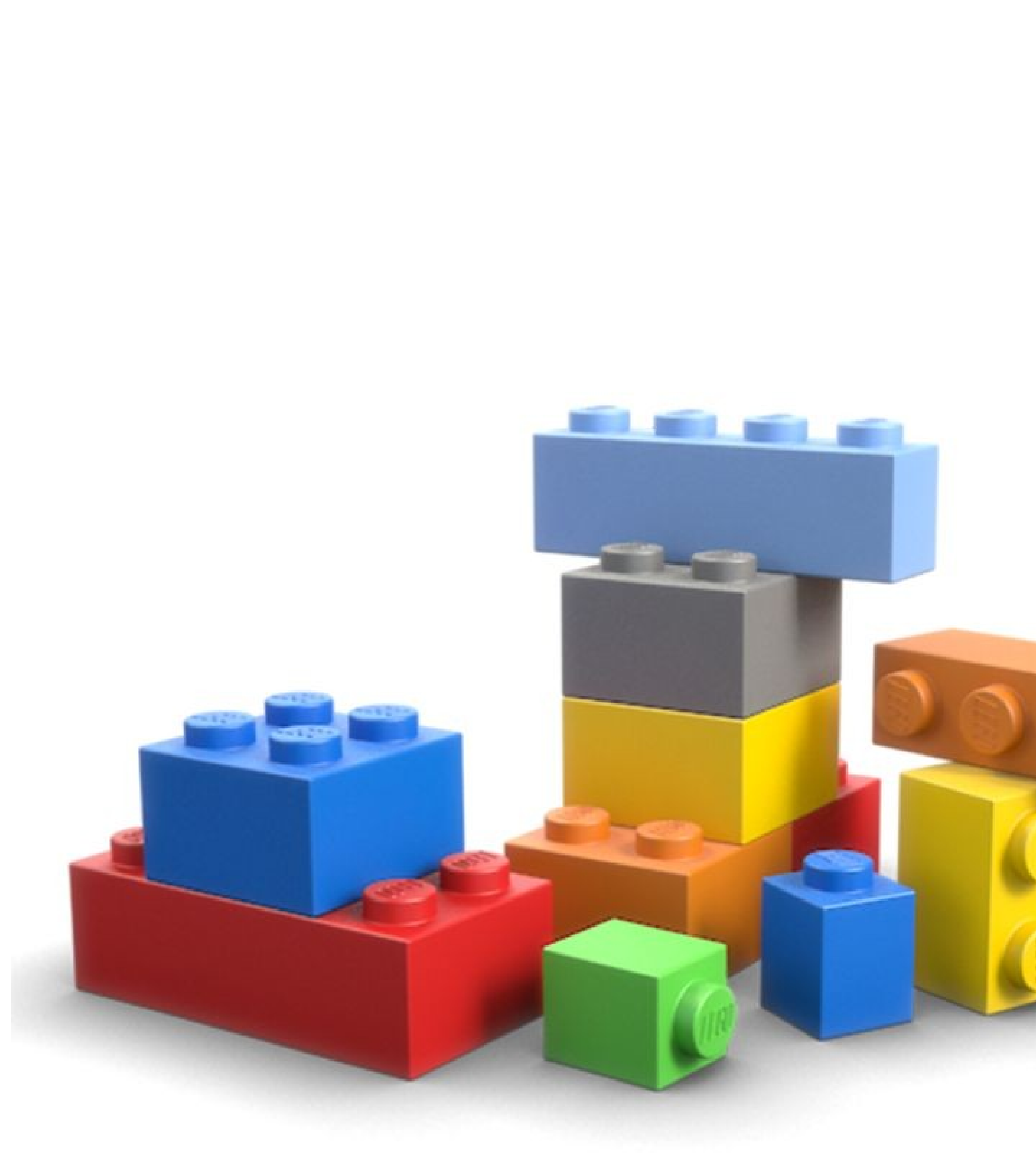
...

Exportar com

`Q_PLUGIN_METADATA()`

`Q_PLUGIN_METADATA()` é utilizado para declarar o identificador da interface implementada (IID) e adicionar metadados do plugin implementado.

Esses metadados são declarados em um arquivo json e inseridos no plugin em tempo de compilação.



Plugins no Qt

...

...

```
class RepetierPlugin : public IFirmware
```

```
{
```

```
    Q_OBJECT
```

```
    Q_PLUGIN_METADATA(IID "org.kde.atelier.core.firmware")
```

```
    Q_INTERFACES(IFirmware)
```

```
...
```

Exemplo em [atcore/src/plugins/repetierplugin.h](#)

Plugins no Qt

...

...

```
class BasicToolsPlugin : public QObject,  
                        public BrushInterface,  
                        public ShapeInterface,  
                        public FilterInterface  
{  
    Q_OBJECT  
    //! [4]  
    Q_PLUGIN_METADATA(IID "org.qt-  
project.Qt.Examples.PlugAndPaint.BrushInterface" FILE "basictools.json")  
    ...
```

Exemplo em [plugandpaint/plugins/basictools/basictoolsplugin.h](#)

Plugins no Qt

...

...

```
"Description[zh_TW]": "Scilab 科學程式環境的後端介面 ",  
  "Icon": "scilabbackend",  
  "Id": "Scilab",  
  "License": "GPL",  
  "Name": "Scilab",
```

...

Exemplo em [cantor/src/backends/scilab/scilabbackend.json](#)

3

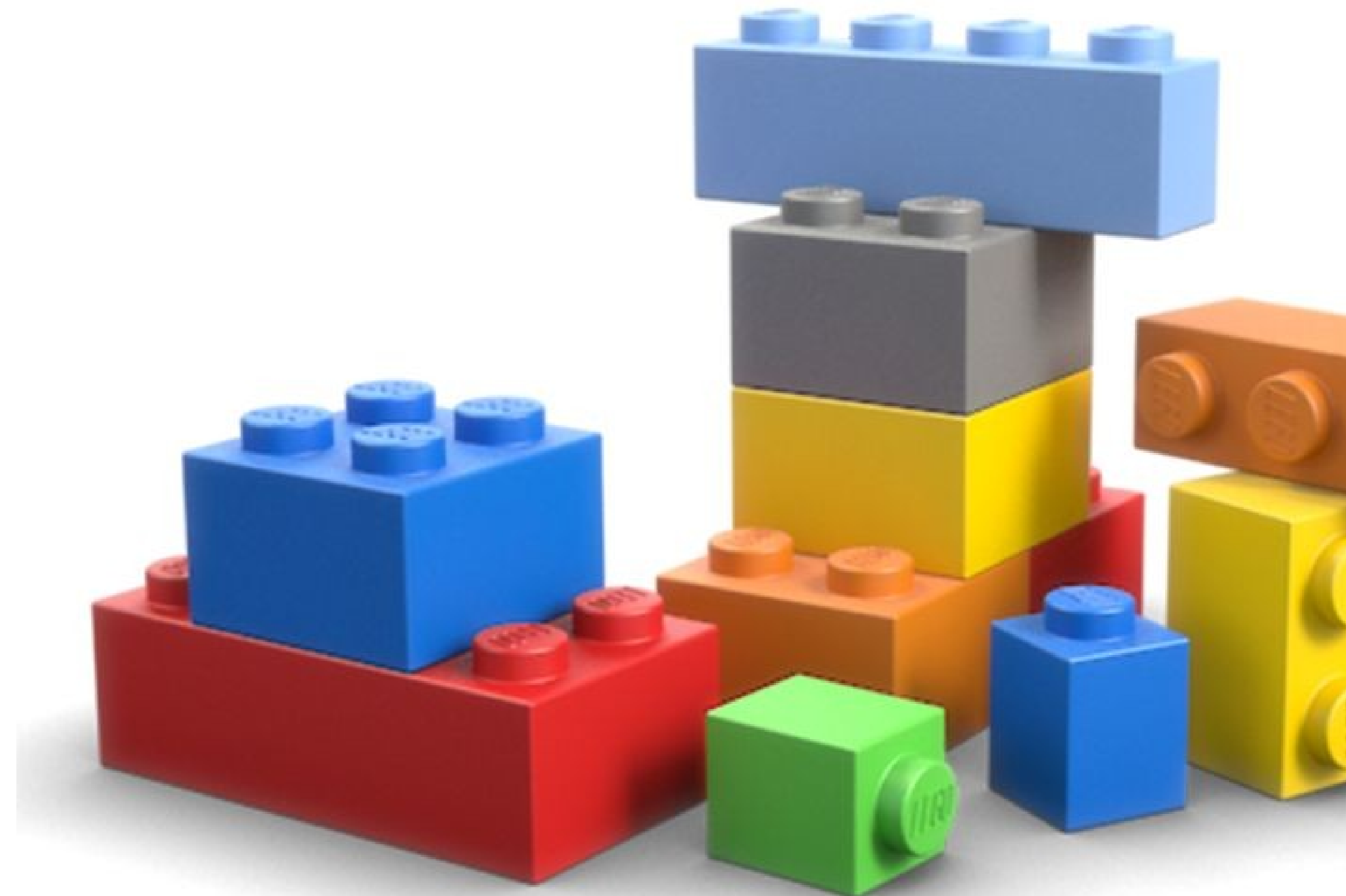
Plugins no KF5

Plugins no KF5

...

A comunidade KDE disponibiliza o KDE Frameworks 5, um conjunto de bibliotecas que estendem o Qt.

O atual release, KDE Frameworks 5 versão 5.51, contém 79 bibliotecas – entre elas, bibliotecas voltadas para desenvolvimento de plugins.



Plugins no KF5

...

▼ **KPluginFactory**

InheritanceChecker

KPluginLoader

KPluginMetaData

KPluginName

KPluginFactory provides a convenient way to provide factory-style plugins

This is used to detect the arguments need for the constructor of plugin classes

This class behaves largely like **QPluginLoader** (and, indeed, uses it internally), but additionally reads the plugin version, as provided by the `K_EXPORT_PLUGIN_VERSION` macro (see **pluginVersion()**) and provides access to a **KPluginFactory** instance if the plugin provides one (see **factory()**)

This class allows easily accessing some standardized values from the JSON metadata that can be embedded into Qt plugins

Represents the name of a plugin intended for **KPluginLoader**

A biblioteca que contém as classes relacionadas com plugins é a **KCoreAddons**

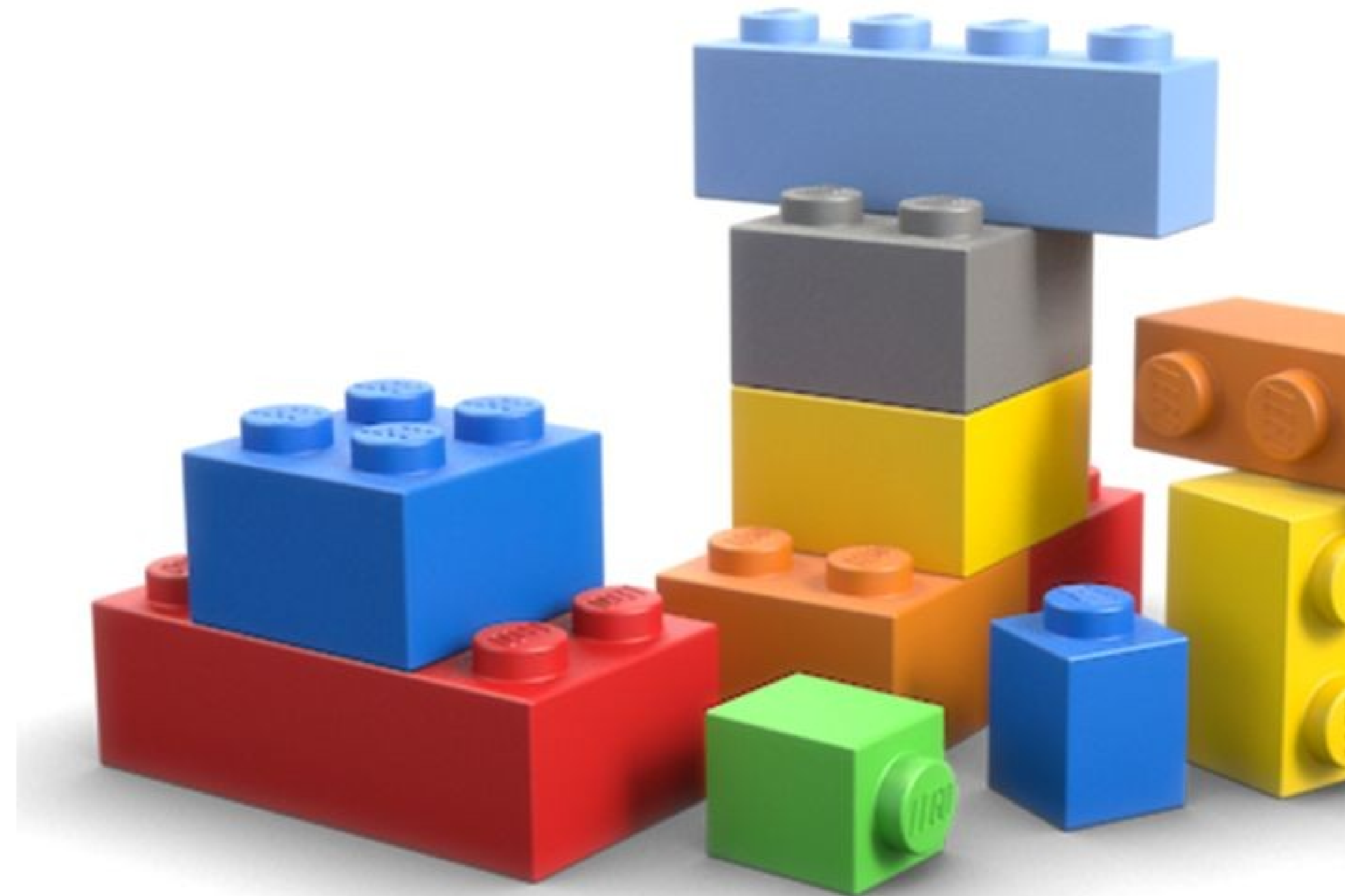
Plugins no KF5

...

KPluginFactory

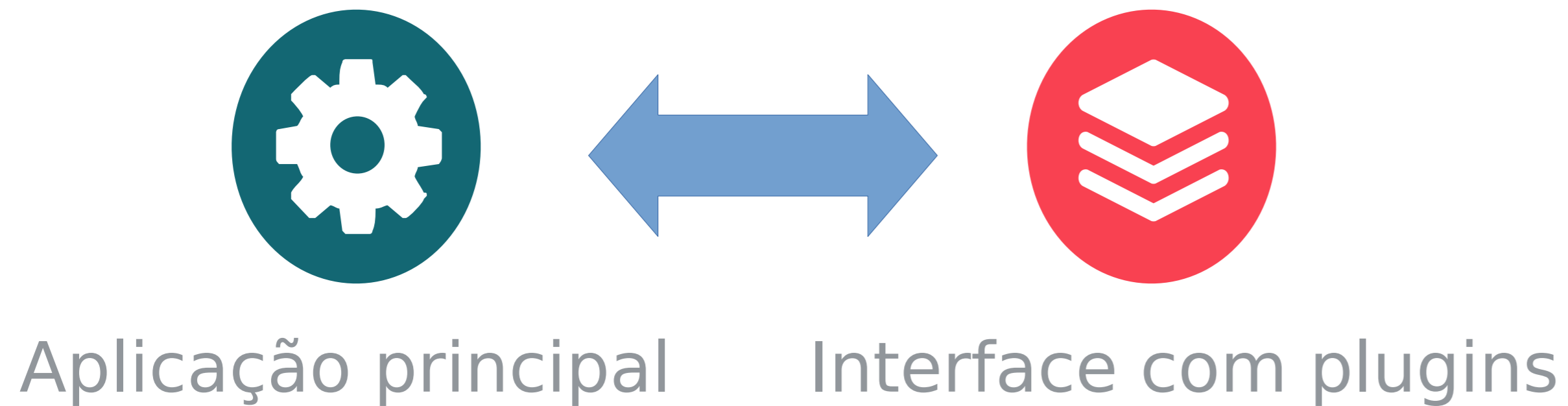
KPluginFactory provê uma maneira conveniente de gerar plugins utilizando o padrão Factory.

Isso modifica a maneira como plugins são implementados em Qt.



Plugins no KF5

...



No lado da aplicação principal, é necessário:

- Definir as interfaces;
- ~~Usar `Q_DECLARE_INTERFACE()`;~~
- Carregar plugins com `QPluginLoader`
 - Ou `KPluginLoader`;
- Testar plugins com `qobject_cast()`.

Plugins no KF5

...

...

```
class CANTOR_EXPORT Backend : public QObject, public KXMLGUIClient
{
    Q_OBJECT
public:
    /**
     * This enum is used to specify the Features, supported by a backend.
     */
    enum Capability{
```

...

Exemplo em [cantor/src/lib/backend.h](#)

Plugins no KF5

...

...

```
KPluginFactory* factory = KPluginLoader(loader.fileName()).factory();  
    Backend* backend = factory->create<Backend>();
```

```
    KPluginMetaData info(loader);  
    backend->d->name=info.name();  
    backend->d->comment=info.description();  
    backend->d->icon=info.iconName();  
    backend->d->url=info.website();  
    backendCache<<backend;
```

...

Exemplo em [cantor/src/lib/backend.cpp](#)

Plugins no KF5

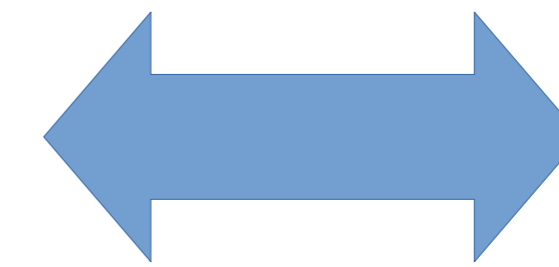
...

No lado do plugin, é necessário:

- Declarar uma classe do plugin que seja QObject e herde a interface;
- Usar `Q_INTERFACES()`;
- Exportar com `Q_PLUGIN_METADATA()`.
- Exportar com `K_PLUGIN_FACTORY`;
 - Ou `K_PLUGIN_FACTORY_WITH_JSON`.
- `#include "myplugin.moc"`



Interface com plugins



Plugins

Plugins no KF5

...

...

```
class ScilabBackend : public Cantor::Backend
```

```
{
```

```
    Q_OBJECT
```

```
public:
```

```
    explicit ScilabBackend( QObject* parent = nullptr, const QList<QVariant>
```

```
args = QList<QVariant>());
```

```
    ~ScilabBackend() override;
```

...

Exemplo em [cantor/src/backends/scilab/scilabbackend.h](https://github.com/KDE/cantor/blob/master/src/backends/scilab/scilabbackend.h)

Plugins no KF5

...

...

```
K_PLUGIN_FACTORY_WITH_JSON(scilabbackend, "scilabbackend.json",  
registerPlugin<ScilabBackend>());  
#include "scilabbackend.moc"
```

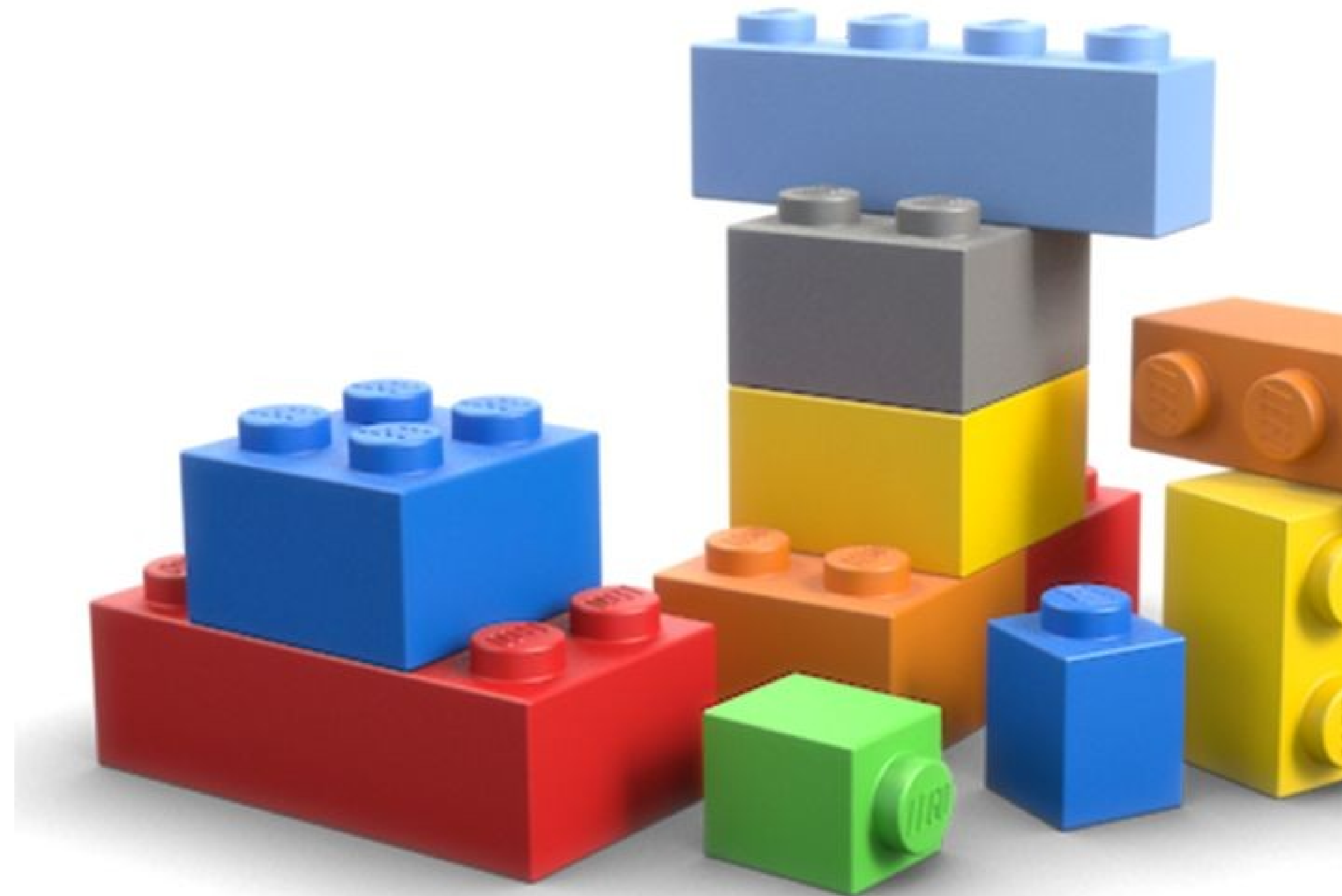
Exemplo em [cantor/src/backends/scilab/scilabbackend.cpp](#)

Plugins no KF5

...

KPluginLoader

Essa classe se comporta como QPluginLoader, mas retorna um objeto factory para a aplicação principal além de já ler por padrão a versão do plugin carregado.

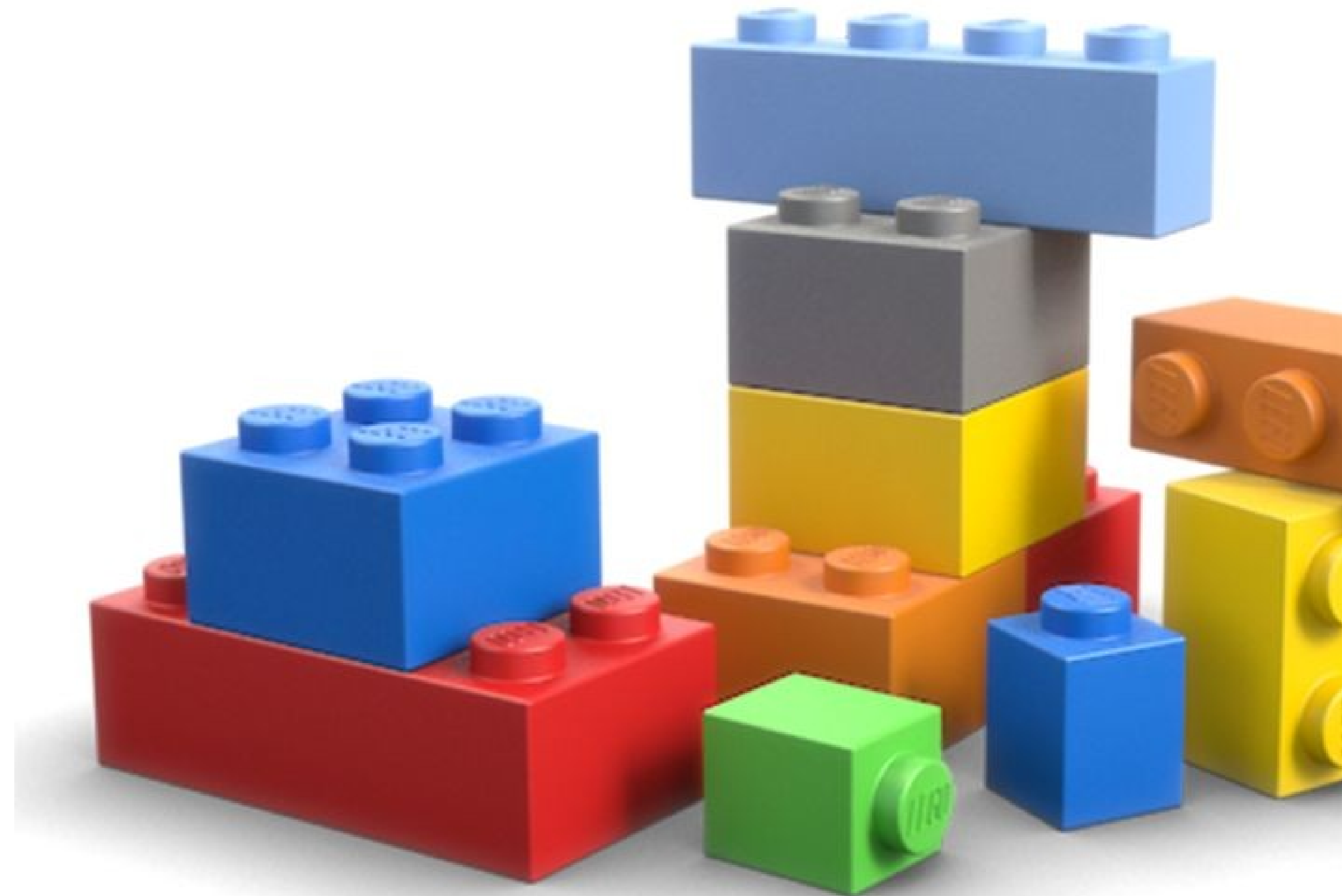


Plugins no KF5

...

KPluginMetadata

KPluginMetadata provê acesso fácil para os campos padrões do arquivo json inserido no plugin.



4

Conclusões

Conclusões

...

1

Plugins são legais para vários tipos de extensão;

2

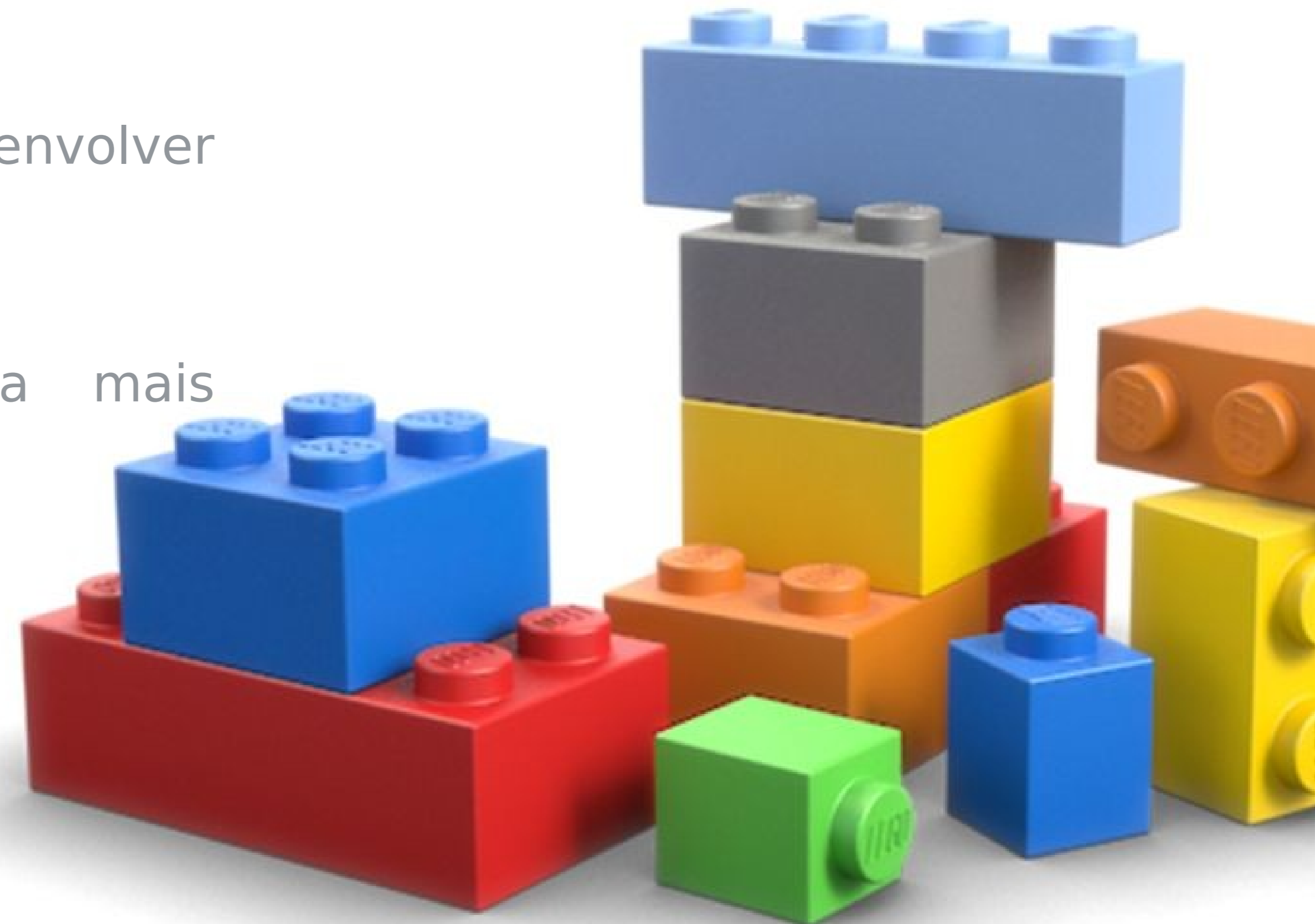
Qt provê uma forma interessante de desenvolver plugins para sua aplicação;

3

O KDE Frameworks 5 provê ainda mais facilidades para essa arquitetura;

4

Entretanto, plugins é uma decisão não apenas técnica, mas também gerencial para projetos.



Qt



Obrigado!
perguntas?

Filipe Saraiva
filipe@kde.org