|  |
| --- |
| iFish |
| 自动更新库客户端示例 |
| [文档副标题] |

|  |
| --- |
| 木鱼  2014-10-19 |

**注意：本文档仅包含基本使用样例，打包工具使用以及更全面的API文档请参见另一个手册！**

**本手册所有调用范例均使用DLL引用方式，如果您需要以EXE方式直接启动独立的进程更新，请参考另一手册！**

本升级库的主页位于：<http://www.fishlee.net/soft/simple_autoupdater/>

反馈论坛： <http://bbs.fishlee.net/forum-63-1.html>

作者： 木鱼 [ifish@fishlee.net](mailto:ifish@fishlee.net)

微博： <http://t.qq.com/ccfish/> <http://weibo.com/imcfish>/

官方主页： <http://www.fishlee.net/>

QQ群： [②群 216126338（高级群）](http://qun.qq.com/air/#134546850/join) [①群 **134546850**（超级群）](http://qun.qq.com/air/#134546850/join)

**请确保在进行如下操作前已经引用了相关的命名空间：**

using FSLib.App.SimpleUpdater;

1. **基本使用方式**：全部使用默认参数更新，不需要自己进行任何额外的处理。

**Updater**.CheckUpdateSimple("http://你的服务器地址/路径/{0}", "xml文件名，一般是 update\_c.xml 或 update.xml");

特点：简单，不用做任何多余的处理。

1. **捕捉更新时间方式：**

var updater=**Updater**.CreateUpdaterInstance("http://你的服务器地址/路径/{0}", "xml文件名，一般是 update\_c.xml 或 update.xml");

updater.Error += (s, e) =>

{

**MessageBox**.Show("更新发生了错误：" + updater.Context.Exception.Message);

};

updater.UpdatesFound += (s, e) =>

{

**MessageBox**.Show("发现了新版本：" + updater.Context.UpdateInfo.AppVersion);

};

updater.NoUpdatesFound += (s, e) =>

{

**MessageBox**.Show("没有新版本！");

};

updater.MinmumVersionRequired += (s, e) =>

{

**MessageBox**.Show("当前版本过低无法使用自动更新！");

};

**Updater**.CheckUpdateSimple();

特点：能知道发生了什么事情，但依然不需要自己手动处理相关逻辑。更多的事件，可以参考API手册，在正式更新之后触发的事件将无法使用此方式捕捉。

1. **自主逻辑模式**

var updater=**Updater**.CreateUpdaterInstance("http://你的服务器地址/路径/{0}", "xml文件名，一般是 update\_c.xml 或 update.xml");

updater.Error += (s, e) =>

{

**MessageBox**.Show("更新发生了错误：" + updater.Context.Exception.Message);

};

updater.UpdatesFound += (s, e) =>

{

**MessageBox**.Show("发现了新版本：" + updater.Context.UpdateInfo.AppVersion);

//确认更新？

updater.StartExternalUpdater();

};

updater.NoUpdatesFound += (s, e) =>

{

**MessageBox**.Show("没有新版本！");

};

updater.MinmumVersionRequired += (s, e) =>

{

**MessageBox**.Show("当前版本过低无法使用自动更新！");

};

updater.Context.EnableEmbedDialog = false;

updater.BeginCheckUpdateInProcess();

特点：能知道发生了什么事情， 并且自己能控制启动更新的时机，并且可以完全使用自己的UI。

1. **多服务器更新模式**

var updater=**Updater**.CreateUpdaterInstance(

new UpdateServerInfo[]

{

new **UpdateServerInfo**("http://你的服务器地址1/路径/{0}", "xml文件名，一般是 update\_c.xml 或 update.xml"),

new **UpdateServerInfo**("http://你的服务器地址2/路径/{0}", "xml文件名，一般是 update\_c.xml 或 update.xml")

//...其它服务器地址

});

**Updater**.CheckUpdateSimple();

多服务器模式下，更新客户端会依次使用指定的服务器，当指定的服务器失效时，会自动切换下一个可用服务器。

1. **确保更新才启动软件模式**

此模式要求检测更新，确认当前版本是最新之后才会继续引导软件。

var updater = **Updater**.CreateUpdaterInstance("http://你的服务器地址1/路径/{0}", "xml文件名，一般是 update\_c.xml 或 update.xml");

updater.EnsureNoUpdate();

updater.EnsureNoUpdate() 方法会阻塞当前线程，直到检测更新操作完成。这个方法具有两个重载，可传入多个不同的委托回调，完成指定的操作。默认无参数调用模式将会使用内置的检查更新提示，是否更新也不强制，根据升级包内部提供的信息来定。通过更改重载可以实现强制升级的效果，具体内容可参考智能提示。

**以上均为基本模式。两大关键点在于：**

1. 创建更新客户端
2. 启动检测时机

**在这之间做的事情，可以自由发挥，任意组合。**

在以上的示例中，**Updater**.CheckUpdateSimple 将会使用 **Updater**.CreateUpdaterInstance 创建一个默认的客户端。如果之前已经创建过客户端，则 **Updater**.CheckUpdateSimple 方法将会使用这个已经存在的客户端。

需要注意的是， **Updater**.CreateUpdaterInstance 方法仅能创建一次，也就是说，重复地调用会导致出异常。