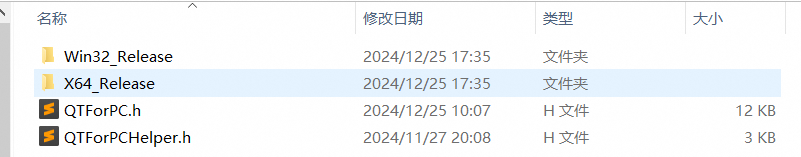
# **QTForPC 接入文档**

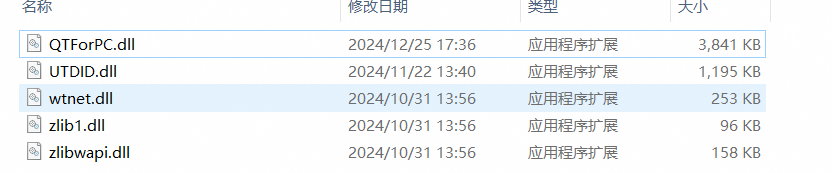
# **引入&配置SDK：**

使用SDK必备操作，缺少任一操作都会造成数据的缺失。

## **SDK基本信息**

|  |  |
| --- | --- |
| 文件名称 | md5 |
| QTForPC.dll | 32 ：ec8a706ad18d218fd5352af85932fc63 |
| 64 ：c70a71c576b648cd31fadb660fa23308 |
| UTDID.dll | 32 ：e5cb460a15b4e76f036245a00b9aea59 |
| 64 ：f7d13897950a16dd7d916e0d51611b05 |
| wtnet.dll | 32：4e8a4fa0758d730735f5b077adc60337 |
| 64：1fa74359bf445093798eea74416954dc |
| zlib1.dll | 32：6094af9aadbed77d042b1009a807a3da |
| 64：c6c4d64d406efcb235e11a8f81003a87 |
| zlibwapi.dll | 32：d5c16da55774777e4fb5558c1145171e |
| 64：959553f1592e6e1522a13544d4f41785 |
| QTForPC.h | 7217bbd6cd5d581e6fcd9b869d470c25 |
| QTForPCHelper.h | 8d11dc71c7ba9d47baf4d08479fed6c6 |



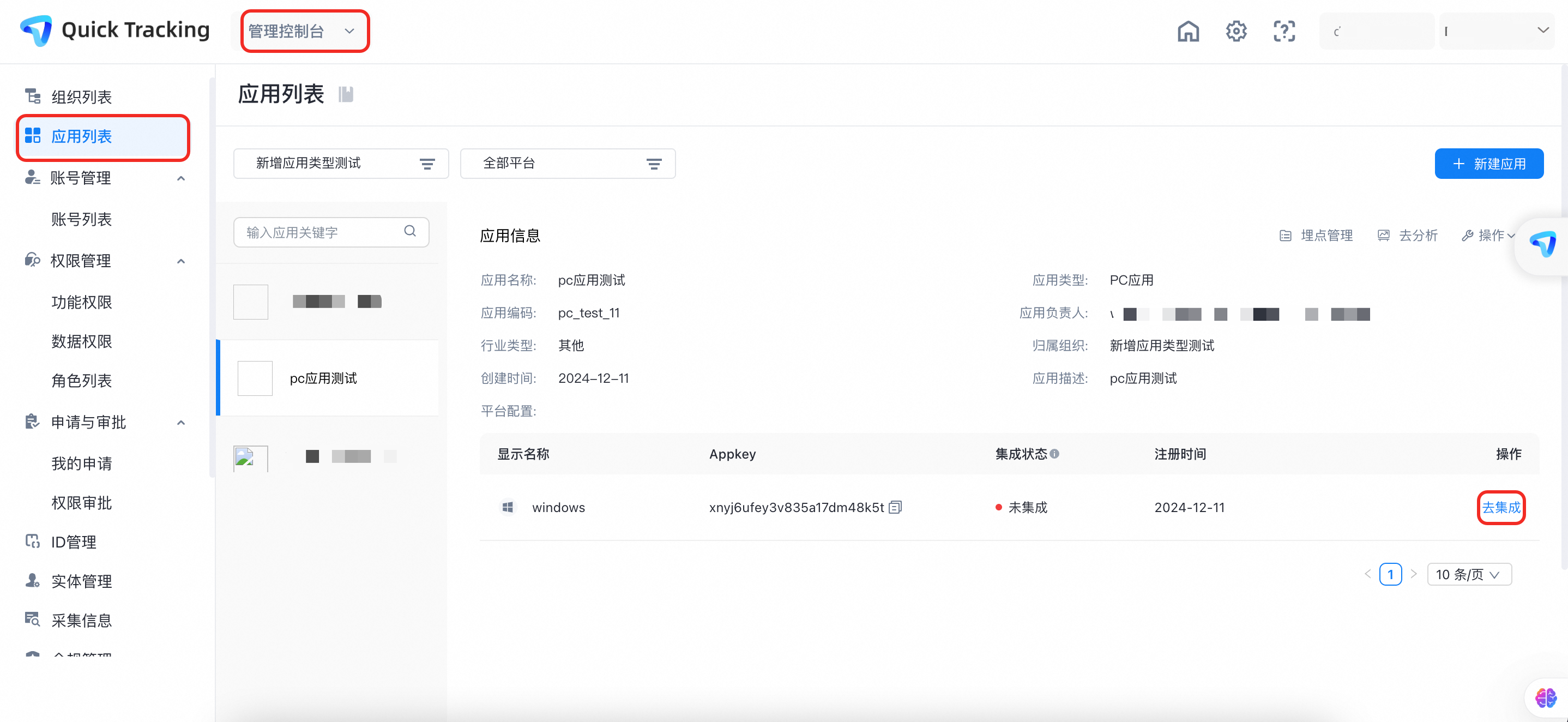


## **1 快速集成**

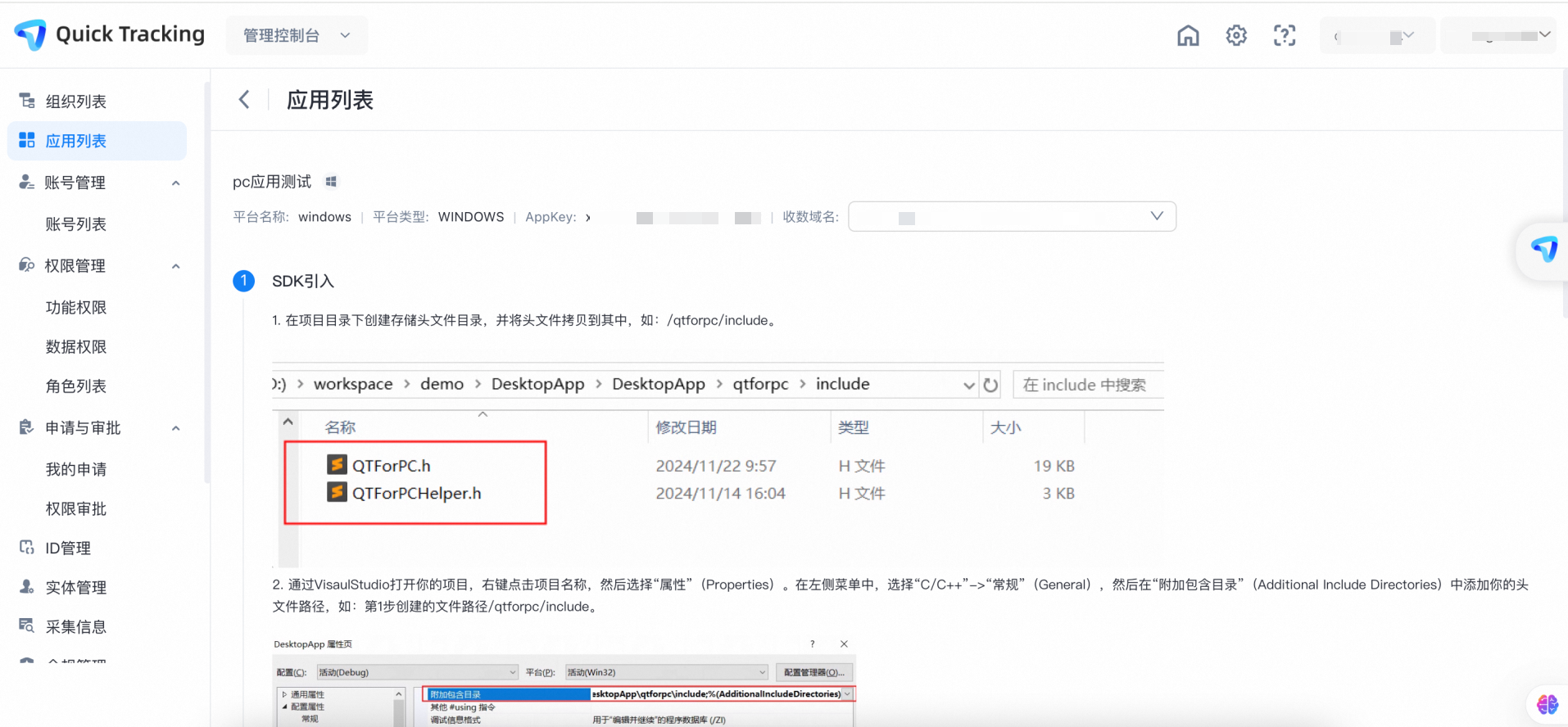
在QuickTracking后台，为每一个App生成了专属的集成代码，可以根据产品内的引导进行集成。

### **1.1 产品路径**

Quick Tracking「首页」-「管理控制台」，在「管理控制台」进入「组织列表」功能，组织列表展示当前已创建的组织，找到应用归属组织，点击「管理应用」功能跳转到「应用管理]页面，在已创建好的应用右侧点击去「去集成」。

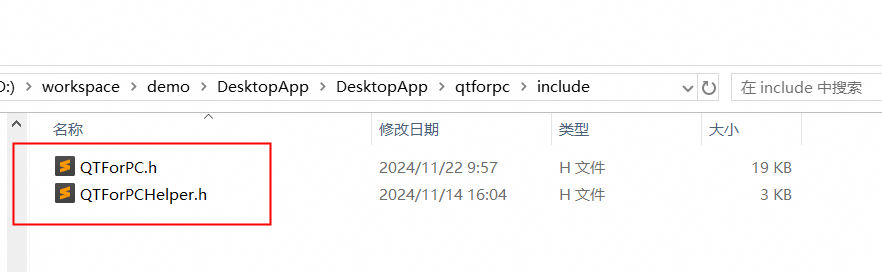


点击「去集成」按钮跳转到新页面可以获取专属集成代码，点击下一步按引导进入到「集成校验」，扫码唤起APP，确认集成是否成功。

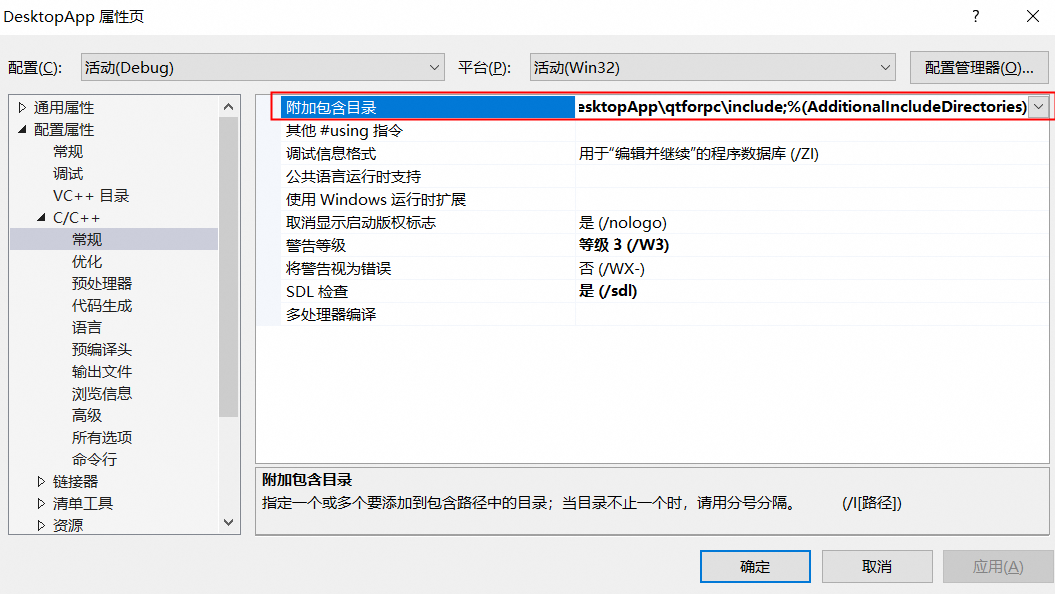


## **2 SDK引入**

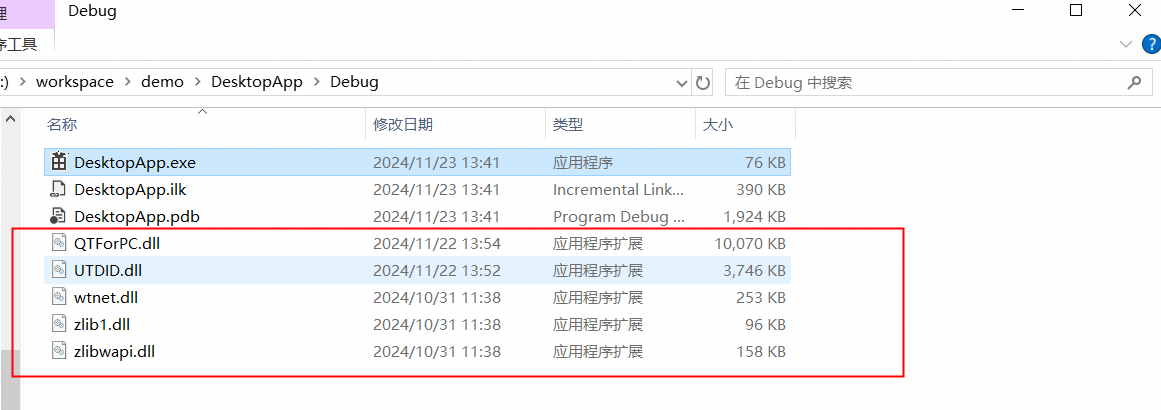
1. 在项目目录下创建存储头文件目录，并将头文件拷贝到其中，如：/qtforpc/include。



1. 通过VisaulStudio打开你的项目，右键点击项目名称，然后选择“属性”（Properties）。在左侧菜单中，选择“C/C++”->“常规”（General），然后在“附加包含目录”（Additional Include Directories）中添加你的头文件路径，如：第1步创建的文件路径/qtforpc/include。



1. 将DLL文件拷贝到与对应平台可执行文件相同的目录下（QTForPC.dll、UTDID.dll、wtnet.dll、zlib1.dll、zlibwapi.dll），如32位Release版DLL文件需要拷贝到对应32位Release版执行文件目录下，使得在运行时可以找到该DLL。



1. 在你的代码中包含头文件，例如：。确保正确调用 DLL 提供的 API。



1. 编译你的项目并运行测试，确保集成正确且功能符合预期。

# **埋点API**

## **0 埋点须知**

### **传入的参数不支持单引号、数据类型等特殊字符串，否则可能导致事件落库失败而使数据丢失。**

### **使用中文参数时需要文件编码是合法的 Unicode(UTF-8 无签名)编码。**

### **除日志开关接口，其他接口的使用需要在调用initQTPC之后使用才能正常生效。**

//自定义属性以及全局属性的key和value超过上限则无法设置成功，自定义属性字符串数组最大size为100超过上限则截断，全局属性集合最大size为50超过上限则无法继续插入。  
const size\_t kStringPropertyValueMaxLength = 4096; //自定义属性以及全局属性value的字符串上限  
const size\_t kStringPropertyKeyMaxLength = 1024; //自定义属性以及全局属性key的字符串上限  
const size\_t kStringArrayValueMaxSize = 100; //自定义属性字符串数组长度的上限  
const size\_t kStringMapValueMaxSize = 50; //全局属性集合的上限  
const size\_t kStringEventCodeMaxLength = 500; //事件编码最大的长度

## **1 初始化**

### **1.1 调用initQTPC**

* AppKey：appKey （必传参数）
* ProcessName：进程名称（必传参数）
* TrackDomain：收数域名 （必传参数）
* param：其他配置参数 （可选参数）（appVersion：app版本号；channel：app渠道号；maxCacheDay：本地日志最大缓存时长（以天为单位，默认为3天，最大为366天。最小为1天）；maxReportDataSize：聚合上报数据最大条数（默认为300条，最大为300条，最小为1条）

//param参数使用QT提供的QT::Helper::QT\_Serializable方法进行简单序列化  
QTFORPC\_API QT\_VOID initQTPC(QT\_CSTR strAppKey, QT\_CSTR strProcessName, QT\_CSTR strTrackDomain, QT\_MAP param);

示例：

QT\_INTERFACE\_PTR qtInterface = NULL;  
try  
{  
 qtInterface = loadQTDLL(L"QTForPC.dll");  
if (!qtInterface)  
{  
fprintf(stderr, "Error loading dll: %lx\n", -1);  
return -1;  
}  
  
 map<string, string> param;  
param["appVersion"] = "testVersion";  
param["channel"] = "testChannel";  
param["maxCacheDay"] = "60";  
param["maxReportDataSize"] = "100";  
std::string sparam;  
QT::Helper::QT\_Serializable(sparam, param);  
  
 qtInterface->initQTPC("appKey", "processname", "http://xxxxx", sparam.c\_str());  
 ......  
  
}  
catch (...)  
{  
 return -1;  
}  
  
system("pause");  
freeQTDLL(qtInterface);

### **1.2 设置用户信息**

* userID：用户id（必传参数）
* nick：用户昵称（可选参数，没有可传空字符串）

// 登录  
QTFORPC\_API QT\_VOID onProfileSignIn(QT\_CSTR userID, QT\_CSTR nick);  
// 登出  
QTFORPC\_API QT\_VOID onProfileSignOff();

示例：

qtInterface->onProfileSignIn("userId", "userNick");  
  
qtInterface->onProfileSignOff();

### **1.3 反初始化**

* 调用finiQTPC停止数据落库
  + cb：回调方法（必传参数）

// 反初始化QT  
QTFORPC\_API QT\_VOID finiQTPC(QT\_CB cb);

示例：

void callBackFunc()  
{  
fprintf(stderr, "finiQTPC callback\n");  
}  
  
qtInterface->finiQTPC(callBackFunc);

### **1.4 设置自定义设备id**

* 调用setCustomDeviceId设置自定义设备ID
  + deviceId：自定义设备（必传参数）

// 设置自定义设备ID  
QTFORPC\_API QT\_VOID setCustomDeviceId(QT\_CSTR deviceId);

示例：

string customDeviceId = "testId";  
qtInterface->setCustomDeviceId(customDeviceId.c\_str());

## **2 开始埋点**

### **2.1 页面事件**

* 在页面即将展现时调用onPageStart
  + pageObj：建议使用容器对象WNDCLASS（必传参数）
  + pageName：页面编码（必传参数）
* 在页面消失之前调用updatePageProperties更新参数
  + pageObj：需要传入onPageStart相同的pageObj（必传参数）
  + pageProperties：页面参数，对应页面事件日志中的事件属性（可选参数），参数需要传入json字面量模板字符串
* 在页面即将消失时调用onPageEnd
  + pageObj：需要传入onPageStart相同的pageObj（必传参数）
* 跳过容器skipPage
  + pageObj：需要传入onPageStart相同的pageObj（必传参数）

**注：页面事件埋点必须按照先onPageStart后onPageEnd成对使用才能确保数据正常。**

// 页面开始展现  
QTFORPC\_API QT\_VOID onPageStart(QT\_VSTR pageObj, QT\_CSTR pageName);  
// 页面开始消失  
QTFORPC\_API QT\_VOID onPageEnd(QT\_VSTR pageObj);  
// 设置页面事件参数  
QTFORPC\_API QT\_VOID updatePageProperties(QT\_VSTR pageObj, QT\_MAP pageProperties);  
// 跳过某个容器，用于多容器嵌套  
QTFORPC\_API QT\_VOID skipPage(QT\_VSTR pageObj);

示例：

WNDCLASS \*wndClass = new WNDCLASS();  
qtInterface->onPageStart(wndClass, "firstPageName");  
std::string cusp = R"({  
 "param\_str": "hello c++",  
 "param\_num": 1900,  
 "param\_bool": true,  
 "param\_list": ["hello", "world", "c++"]  
})";  
qtInterface->updatePageProperties(wndClass, cusp.c\_str());  
qtInterface->onPageEnd(wndClass);  
  
WNDCLASS \*wndClass1 = new WNDCLASS();  
qtInterface->onPageStart(wndClass1, "secondPageName");  
qtInterface->onPageEnd(wndClass1);  
  
WNDCLASS \*wndClass2 = new WNDCLASS();  
qtInterface->onPageStart(wndClass2, "thirdPageName");  
qtInterface->onPageEnd(wndClass2);  
  
WNDCLASS \*wndClass3 = new WNDCLASS();  
//跳过FourthPageName的上一个页面，也就是thirdPageName  
qtInterface->skipPage(wndClass3);  
qtInterface->onPageStart(wndClass3, "FourthPageName");  
qtInterface->onPageEnd(wndClass3);

### **2.2** 埋点**事件**

* 点击时调用trackEvent或trackEventWithPageName
  + pageObj：建议使用容器对象WNDCLASS（必传参数）
  + id：事件编码（必传参数，不能传递以”$$\_“开头的参数作为id）
  + customProperties：事件属性（可选参数）
  + pageName：页面编码（必传参数）

**注：使用trackEvent或trackEventWithPageName需要保证在onPageStart后以及onPageEnd前调用才能保证数据准确性。**

//埋点事件  
QTFORPC\_API QT\_VOID trackEvent(QT\_VSTR pageObj, QT\_CSTR id, QT\_MAP customProperties);  
//埋点事件带页面名称  
QTFORPC\_API QT\_VOID trackEventWithPageName(QT\_VSTR pageObj, QT\_CSTR id, QT\_MAP customProperties, QT\_CSTR pageName);

示例：

WNDCLASS \*wndClass = new WNDCLASS();  
qtInterface->onPageStart(wndClass, "PageName");  
std::string cusp = R"({  
 "param\_str": "hello c++",  
 "param\_num": 1900,  
 "param\_bool": true,  
 "param\_list": ["hello", "world", "c++"]  
})";  
qtInterface->trackEventWithPageName(wndClass, "test\_event", cusp.c\_str(), "PageName");  
qtInterface->onPageEnd(wndClass);

### **2.3 前后台事件**

* 在切到前台时调用enterForeground/enterForegroundWithPageName
  + pageObj：建议使用容器对象WNDCLASS（必传参数）
  + customProperties：业务参数（可选参数）
  + pageName：页面编码（必传参数）
* 在切到后台时调用enterBackground/enterBackgroundWithPageName

**注：使用enterForeground/enterForegroundWithPageName或enterBackground/enterBackgroundWithPageName需要保证在onPageStart以及和onPageEnd前调用才能保证数据准确性。**

// 进入前台  
QTFORPC\_API QT\_VOID enterForeground(QT\_VSTR pageObj, QT\_MAP customProperties);  
// 进入前台带PageName  
QTFORPC\_API QT\_VOID enterForegroundWithPageName(QT\_VSTR pageObj, QT\_MAP customProperties, QT\_CSTR pageName);  
// 进入后台  
QTFORPC\_API QT\_VOID enterBackground(QT\_VSTR pageObj, QT\_MAP customProperties);  
// 进入后台带PageName  
QTFORPC\_API QT\_VOID enterBackgroundWithPageName(QT\_VSTR pageObj, QT\_MAP customProperties, QT\_CSTR pageName);

示例：

WNDCLASS \*wndClassForeground = new WNDCLASS();  
qtInterface->onPageStart(wndClassForeground, "StartPageName");  
std::string start\_cusp = R"({  
 "param\_str": "hello c++",  
 "param\_num": 1900,  
 "param\_bool": true,  
 "param\_list": ["hello", "world", "c++"]  
})";  
qtInterface->enterForeground(wndClassForeground, start\_cusp.c\_str());  
qtInterface->onPageEnd(wndClassForeground);  
  
WNDCLASS \*wndClassBackground = new WNDCLASS();  
qtInterface->onPageStart(wndClassBackground, "EndPageName");  
std::string end\_cusp = R"({  
 "param\_str": "hello c++",  
 "param\_num": 1900,  
 "param\_bool": true,  
 "param\_list": ["hello", "world", "c++"]  
})";  
qtInterface->enterBackground(wndClassBackground, end\_cusp.c\_str());  
qtInterface->onPageEnd(wndClassBackground);

### **2.4 用户属性上报**

* 用户属性上报调用uploadUserProfile
  + pageObj：建议使用容器对象WNDCLASS（必传参数）
  + customProperties：业务参数（必传参数）

**注：使用uploadUserProfile需要保证在onPageStart后以及onPageEnd前调用才能保证数据准确性。**

// 用户属性上报  
QTFORPC\_API QT\_VOID uploadUserProfile(QT\_VSTR pageObj, QT\_MAP customProperties);

示例：

WNDCLASS \*wndClass = new WNDCLASS();  
qtInterface->onPageStart(wndClass, "PageName");  
std::string cusp = R"({  
 "age": 18,  
 "level": "王者",  
 "name": "coolboy",  
})";  
qtInterface->uploadUserProfile(wndClass, cusp.c\_str());  
qtInterface->onPageEnd(wndClass);

### **2.5 全局属性**

* 调用registerGlobalProperty设置全局属性
  + key：设置的全局属性key（必传参数）
  + value：设置的全局属性value（必传参数）
* 调用unregisterGlobalProperty删除全局属性
  + key：要删除的全局属性key（必传参数）
* 调用getGlobalProperty获取指定key全局属性
  + key：要获取的全局属性key（必传参数）
  + value：要获取的全局属性value（必传参数）
* 调用getGlobalProperties获取全部全局属性
  + properties：要获取的全部全局属性（必传参数）

**注：设置全局属性是异步操作，在registerGlobalProperty使用后马上调用getGlobalProperty/getGlobalProperties可能会导致属性获取失败。**

// 设置全局属性  
QTFORPC\_API QT\_VOID registerGlobalProperty(QT\_CSTR key, QT\_CSTR value);  
// 根据Key获取全局属性  
QTFORPC\_API QT\_VOID getGlobalProperty(QT\_CSTR key, QT\_STR value, QT\_INT size);  
// 获取全部全局属性  
QTFORPC\_API QT\_VOID getGlobalProperties(QT\_STR properties, QT\_INT size);  
// 根据Key删除全局属性  
QTFORPC\_API QT\_VOID unregisterGlobalProperty(QT\_CSTR key);

示例：

qtInterface->registerGlobalProperty("key", "value");  
  
qtInterface->unregisterGlobalProperty("key");  
  
// 大小根据需求设定  
char value[64] = { 0 };  
qtInterface->getGlobalProperty("key", value, 64);  
string myValue = value;  
  
// 大小根据需求设定  
char properties[128] = { 0 };  
qtInterface->getGlobalProperties(properties);  
map<string, string> myProperties = QT::Helper::QT\_DeserializePerson(properties, 128);

### **2.6 ABTest**

* 调用initABTest设置全局属性
  + serverUrl：服务端地址（必传参数）
  + options：属性参数（必传参数）
    - experiment\_file\_path : 字符串类型，必传参数，AB实验结果缓存文件
    - update\_interval\_seconds：int类型，可选参数，AB实验轮询间隔，默认10分钟
* 调用fetchABTestFromCache
* 调用fetchABTestFromServer
* 调用fetchABTestFromServerThenCache
  + paramName：参数名称（必传参数）
  + valueType：值类型（必传参数）
  + defaultValue：默认值（必传参数）
    - 如果valueType是JSON类型，defaultValue需要传递json字符串字面量形式，例如：R"({"a":1})"
  + handler：回调方法（必传参数）
  + timeout\_seconds：超时时间（可选参数）

//ABTest 初始化  
QTFORPC\_API QT\_VOID initABTest(QT\_CSTR serverUrl, QT\_MAP options);  
//ABTest 请求缓存实验  
QTFORPC\_API QT\_VOID fetchABTestFromCache(QT\_CSTR paramName, QT\_CSTR valueType, QT\_MAP defaultValue, QT\_AB\_CB handler);  
//ABTest 请求服务端实验  
QTFORPC\_API QT\_VOID fetchABTestFromServer(QT\_CSTR paramName, QT\_CSTR valueType, QT\_MAP defaultValue, QT\_AB\_CB handler, QT\_INT timeout\_seconds);  
//ABTest 先请求本地缓存实验结果，如果没有再请求服务端实验结果  
QTFORPC\_API QT\_VOID fetchABTestFromCacheThenServer(QT\_CSTR paramName, QT\_CSTR valueType, QT\_MAP defaultValue, QT\_AB\_CB handler, QT\_INT timeout\_seconds);

示例：

std::string abOptions = R"({  
 "experiment\_file\_path": "D:\\xxxx\\abcache\\ab\_cache.txt",  
 "update\_interval\_seconds": 10  
})";  
  
qtInterface->initABTest("http://xxxxx", abOptions.c\_str());  
  
qtInterface->fetchABTestFromCache("y\_number", "NUMBER", "1", [](const string &result){  
 std::cout << result << std::endl;  
});  
  
qtInterface->fetchABTestFromServer("y\_json", "JSON", R"({"a":1})", [](const string &result){  
 std::cout << result << std::endl;  
}, 3);  
  
qtInterface->fetchABTestFromServerThenCache("y\_boolean", "BOOLEAN", "false", [](const string &result){  
 std::cout << result << std::endl;  
}, 3);

## **3 调试**

### **3.1 实时调试模式**

* 调用turnOnRealTimeDebug开启实时调试模式，上报数据间隔变为三十秒
  + configs：配置参数（必传参数）
* 调用turnOffRealTimeDebug关闭实时调试

/\*\* 开启实时调试模式\*/  
QTFORPC\_API QT\_VOID turnOnRealTimeDebug(QT\_MAP configs);  
/\*\* 关闭实时调试 \*/  
QTFORPC\_API QT\_VOID turnOffRealTimeDebug();

示例：

map<string, string> configs;  
configs["debug\_key"] = "test";  
std::string sconfig;  
QT::Helper::QT\_Serializable(sconfig, configs);  
qtInterface->turnOnRealTimeDebug(sconfig.c\_str());  
  
qtInterface->turnOnRealTimeDebug();

### **3.2 日志**

* 调用setQTLog开启日志功能，需要在initQTPC之前调用
  + strAppKey：appkey（必传参数）
  + strProcessName：进程名（必传参数）
  + type：日志输出方式（必传参数）

// 日志打印 0:无 1:标准输出stdout 2:文件输出%appdata%/qtforpc/appkey/  
QTFORPC\_API QT\_VOID setQTLog(QT\_CSTR strAppKey, QT\_CSTR strProcessName, LOG\_PRINT\_TYPE type);

示例：

//日志打印  
qtInterface->setQTLog("47130293", "testprocessname", LOG\_PRINT\_TYPE::LOG\_PRINT\_TYPE\_FILE);  
//初始化  
qtInterface->initQTPC("appKey", "processname", "http://xxxxx", sparam.c\_str());

### **3.3 开启快速上传模式**

* 调用setFastUploadMode开启快速上传模式，上报数据间隔变为每秒一上报，默认是三秒一上报

/\* 开启快速上传模式, \*/  
QTFORPC\_API QT\_VOID setFastUploadMode();

示例：

qtInterface->setFastUploadMode();

## **4.常见问题**

windows10.x 版本获取操作系统版本不准确问题<https://www.jianshu.com/p/ddb029101e3c>