TCM（Task Component Model）技术应用规范

版本：2013/3/26（重制 2 修订 19）

此规范用于TCM标准化开发，是TCM SDK的重要组成部分。

1. 起始

TCM是以高性能、跨语言为基本要求，在整个软件生命周期中，降低架构耦合程度为目的，设计研发的中间件。基于TCM技术，您可以快速搭建一个组件化的软件开发框架。

以下“组件”皆代表“基于TCM的组件”。

开发和移植组件之前，请准备相关技术文档。

在文档中，应当说明组件的通用特性、功能清单，并对每一个功能做详尽描述。

2. 组件物理形式与约束

组件的物理形式是一系列文件的集合。

包括：

[组件标识].dll（标准C++动态链接库，×1）

[组件标识].tcm.xml（组件配置文件，×1）

[组件标识].[功能标识].tcm.png（组件图标组，×m）

[组件标识].[附属程序集标识].dll/exe（组件附属程序集，×n）

组件标识和附属程序集标识仅限于字符集合[A-Za-z0-9]。

3. 组件发布器

开发人员请直接使用 *[组件标识]* 命名组件项目。

开发人员执行生成操作后，请使用SDK提供的**组件发布器**进行组件的配置和发布。

（您也可以手动配置，但是建议避免这种情况）

组件发布器在首次使用之前，可能会要求配置发布目录。

发布操作会检查组件完整性、检测DLL导出函数、执行重命名、更新发布目录。

发布成功后，组件二进制文件、配置文件、资源文件等组件内容，将按照开发人员的预先设置，存储到指定的发布目录。

更多详细内容，请参考组件发布器的用户手册。

4. 组件构造

逻辑上，一个组件中可以包含多种功能（function）。

原则上，一个设计良好的组件应当能够**在依赖有限资源并独立于其他组件**的情况下，正常运作，或至少提供完整运行时反馈。

请自行考量并设计功能的粒度。

虽然拆分功能可以提高复用能力，但也不要将功能拆分过细。

请细致地设计功能的输入与输出。

每个功能必须使用一个**所在组件内唯一**的整数ID进行标识。建议使用正整数。

5. 开发基础

基于TCM开发组件需要至少引用 tcm_cdev.h 头文件。

tcm_cdev.h包含一个开关宏“TCM_COMPILER_CDEV”，用于屏蔽不用于组件开发的功能。

您可以人工关闭该宏获得完整功能。

您也可以通过引用其他头文件获得更多的功能。

TCM的所有功能都封装在“tcm”命名空间内。

6. 组件接口规范

一个组件（可能）需要包含以下C风格导出函数。

1) 主调函数（必须）

主调函数内，应当实现且仅实现功能的跳转（或调用）。

可以将具体跳转流程写入文档以作提示，不宜在主调函数内直接实现具体功能。

函数签名：*UINT Run (int function, Envelope* envelope, Context* context)*

完整的写法如下：

```
#include "tcm_cdev.h"

using namespace tcm;

extern "C" __declspec(dllexport)

UINT Run(int function, Envelope* envelope, Context* context);
```

关于输入参数

function是功能的标识。

envelope是用于参数交换的信封对象。关于信封对象详见后文。

context是当前功能的执行上下文。关于上下文对象详见后文。

关于返回码

返回码用于描述功能的执行状态，不是功能本身的输出参数。

如果用户希望返回**与组件功能相关**的信息，请将这些信息作为输出参数，

装填在信封对象中，执行传递。

以下是TCM为返回码定义的含义：

返回值	含义
0	未定义的错误
1	正常执行完成
2	用户取消，通知方式
3	用户取消，强制方式
4	功能不存在
5	没有执行任何功能
[6,DWORD_MAX]	保留，用于TCM未来的扩展定义

7. 组件用途

基于TCM的组件善于执行没有交互或轻量交互的可自动化的任务。

一般，请不要在设计TCM组件时，使用交互频繁的用户界面。

因为，这会为功能粒度设计引入潜在难度。

8. 避免重复异步

TCM及TCM的扩展中间件都可能实现TCM组件的调度和监控功能。

根据TCM平台开发规范，实现这些功能已经考虑异步任务调度。

组件的功能中，一般情况请**不要**使用异步（具备潜在多线程可能的）代码。

如果存在IO操作，一般建议使用同步IO。

如果一定要在IO时，异步执行计算任务，可以将IO与计算分离成不同的功能。

友情提示：

多线程技术在大多数情况下**不能**提高计算过程本身的效率。

如有需要，应该专门设计并行计算的方案。

考虑到TCM本身可能对线程进行并行调度优化，所以建议仅对核心计算代码进行并行改造。

9. 组件状态通信

组件使用上下文对象（Context）操作执行状态。上下文对象是线程安全的。

通过上下文对象，组件可以广播执行进度更新，也可以监听平台发布的控制码。

控制码是一种“君子协定”，实质是来自调用方的控制消息，控制码本身没有强制能力。

组件应当主动监听并响应控制码，以实现合理协调的受控行为。

注意，TCM提供了一些具有强制能力的API。

组件自身不作为时，将可能被TCM运行时强制控制，以避免进一步的调度失败。

但是，强制措施容易导致功能的执行出现潜在问题。

所以，请主动实现受控行为。

上下文对象中定义了3个控制码，如下表，表中还给出了建议动作。

标识符	值	含义
TCM_CTRL_NULL	0	没有特殊的控制指令 建议组件忽略
TCM_CTRL_PAUSE	1	调用方通知组件暂停 建议组件间歇睡眠，并监听恢复通知
TCM_CTRL_RESUME	2	调用方通知组件继续 建议组件立刻唤醒，继续执行
TCM_CTRL_CANCEL	3	调用方通知组件取消 建议组件停止功能，并执行清理

*如果需要更多的受控方式，请将需求告知TCM的设计开发人员以商议更新。

上下文对象面向组件开发提供以下常用方法。

签名	说明
UINT GetCtrlCode()	获取来自调用方的控制码
float GetProgress()	获取当前执行进度
void SetProgress(float value)	设置当前执行进度。value范围：[0,100]
void SwitchPause()	切换真实执行状态的运行和暂停

示例：

监听并响应清理操作

```
if(context->GetCtrlCode() == TCM_CTRL_CANCEL) {
    //执行取消执行的清理
    //执行终结代码
}
```

监听并响应暂停操作

```
if(context->GetCtrlCode() == TCM_CTRL_PAUSE) {
    context->SwitchPause();
    while(true) {
        Sleep(100);
        if(context->GetCtrlCode() == TCM_CTRL_RESUME) {
            context->SwitchPause();
            break;
        }
        //执行暂停期间其他操作
    }
}
```

获取和设置进度

```
context->SetProgress(i * 100.0f / total);
cout<<context->GetProgress()<<endl;
```

10. 组件参数通信

组件与调用方使用信封对象（Envelope）交换参数。信封对象由TCM构造。

大多数情况，调用方通过TCM，预生成符合功能要求的参数信封。

其中，功能需要的内存托管于TCM。

TCM支持全部数据类型。

但这并不意味着TCM支持任意类型，具体见下表。

类型标识符	描述
TCM_DT_POINTER	指针
TCM_DT_INT8	8位整数
TCM_DT_UINT8	无符号8位整数
TCM_DT_INT16	16位整数
TCM_DT_UINT16	无符号16位整数
TCM_DT_INT32	32位整数
TCM_DT_UINT32	无符号32位整数
TCM_DT_INT64	64位整数
TCM_DT_UINT64	无符号64位整数
TCM_DT_REAL32	32位浮点数
TCM_DT_REAL64	64位浮点数
TCM_DT_BOOL	布尔值
TCM_DT_CSTRA	多字节常字符串
TCM_DT_CSTRW	宽字节常字符串

关于用户对象和数组

用户的自定义对象和数组应当使用指针替代。

向信封对象写入这些数据时，**可以**同时提供该对象的实际大小（以字节为单位）。

当开发人员提供对象大小（非0）时，TCM将执行**浅拷贝**，在内存中生成对应对象的浅表副本，并记录此指针。这样，即便原有对象由于某些原因失效，写入信封中的对象依然有效。但是TCM并不关心原先的对象，所以一些开发失误可能导致内存泄露。

如果开发人员不提供对象的实际大小（0），TCM将仅写入引用。

这种情况下请自行保证该对象的生命周期和引用安全。

这时，如果外部在TCM依然占据该对象引用时释放了对象，则可能引发严重访问冲突。

关于转型读写支持

TCM提供有限的转型读写支持，可以在读写参数时 序列化到字符串 或 从字符串反序列化数据。

但是，组件开发API中默认屏蔽了这一功能。

这要求组件开发过程中使用明确类型的读写方法，避免使用转型读写。

信封对象面向组件开发人员提供以下常用方法。

签名	说明
`int GetParamTotal()`	获取参数数量
`bool GetInState(int id)`	获取参数方向
`T Read(int index)`	读取参数
`void Write(int index, T value)`	写入参数
`void Write(int index, LPVOID value, int size)`	写入对象（重载）
`void Write(int index, PCSTR value)`	写入多字节常字符串（重载）
`void Write(int index, PCWSTR value)`	写入宽字节常字符串（重载）

示例：

从信封中读出索引为 0 的 32位整型参数

```
int a = envelope->Read<int>(0);
```

向信封中写入索引为 5 的 32位浮点型参数

```
float a = 1.5f;
envelope->Write(5,a);
```

向信封中用两种方式写入一个对象，索引为 3 和 4

```
class TestA {
public:
    int a;
    float b;
    long c;
};

void Func(Envelope* envelope) {
    TestA* obj_a = new TestA();
    envelope->Write(3,obj_a,sizeof(obj_a)); //写入对象的浅表副本
    envelope->Write(4,obj_a,NULL); //仅写入对象的引用
}
```

11. 其他常用函数

TCM为开发人员提供了一系列工具函数。

签名	说明
PCWSTR MbToWc(PCSTR src, UINT codepage)	转换多字节字符串到宽字节字符串
PCSTR WcToMb(PCWSTR src, UINT codepage)	转换宽字节字符串到多字节字符串
PCSTR FixEncoding(PCSTR src)	进行字符串编码修复
PCSTR ValueToStrA(T value)	转换数值到字符串
PCWSTR ValueToStrW(T value)	转换数值到字符串（Unicode）
T* VectorToArray(vector<T>& src)	转换vector容器到定长（堆）数组
int GetTypeUnit(int type)	获得TCM类型的单位大小
void Assert(bool state, LPVOID clear)	根据断言结果执行清理
PCSTR CopyStrA(PCSTR src)	复制一个字符串
PCWSTR CopyStrW(PCWSTR src)	复制一个字符串（Unicode）
PCSTR GetRandomStrA(int len)	获得一个随机HEX字符串
PCWSTR GetRandomStrW(int len)	获得一个随机HEX字符串（Unicode）
void ShowMsgbox(T value, PCWSTR caption)	显示简易的信息框，以数值为内容
void ShowMsgbox(PCSTR value, PCSTR caption)	显示简易的信息框，自定义内容
void ShowMsgbox(PCWSTR value, PCWSTR caption)	显示简易的信息框，自定义内容（Unicode）
PCWSTR GetAppDir()	获取当前应用程序目录
PCWSTR GetAppName()	获取当前应用程序名称

12. 运行时解耦

TCM提供一个宿主程序（tcm_host.exe）调用具体的TCM组件功能。

通过使用宿主程序调用TCM组件功能，您可以获得以下好处：

i) 调用方可以不依赖tcm_bridge.dll

ii) 调用方不会因为组件功能的崩溃而崩溃

iii) 调用方可以更方便的定制调度逻辑

通过使用宿主程序调用TCM组件功能，会导致以下问题：

i) 调用方与组件之间更难进行复杂对象通信

ii) 调用方与组件之间的通信效率根据具体的方案不同会有不同程度降低

使用宿主可以调用命令行：tcm_host <任务描述文件>

更丰富的命令行开关可以通过tcm_host本身查看。

任务描述文件是描述一次远程调用的全部信息的文件。

具体可以参考任务描述文件规范，也可以使用SDK的工具生成任务描述文件。

13. 多语言支持

管理语言包

14. 扩展数据结构

标志和字典