rdv2 API使用说明

rdv2 API库的目标是为了简化VDR仿真模型的开发，使之能够根据模型功能的需求，快速构建软件，实现与仿真平台其他节点之间的数据通信。

# 术语

## 仿真平台

指由若干仿真节点组成的分布式仿真系统，各仿真节点位于同一局域网内，承担不同角色，通过VDR交换数据，共同完成某一仿真任务。

## VDR

VDR (Virtual Data Repertory)，虚拟数据仓库，是各仿真节点之间数据交换与同步后，在各节点内存中形成的虚拟数据空间。其属性主要包括域（Region）和ID。

## VDR客户端

VDR客户端是参与仿真运行的独立软件单位，通过VDR完成与其他节点的数据通信。创建VDR客户端时需要指定VDR属性（域、ID、组播地址和端口）及本机属性（本机地址和端口）。

## 数据集与数据

VDR通过数据集来实现对关联数据的逻辑管理，数据集属性包括名称、VDR域。数据属性包括名称、类型、数据集域、数据集ID。

## 节点与接入点

这里说的节点（Node）不是仿真节点，而是VDR客户端的数据节点，其属性包括名称、VDR域和节点类型。创建某个数据节点，并将之与某个数据集连接，可以方便接入点的使用。

接入点（Access Point）是VDR客户端用于进行网络数据交互的数据访问接入点，创建后将其与指定域的VDR连接。VDR的数据读写操作都是通过接入点完成的，数据读操作只需要知道数据名称即可。在取得某个数据节点的授权后，即可对此数据节点连接的数据集中的数据进行写入操作（也可以通过直接附加到某个数据集来取得对该数据集中数据的写入权限）。

# 接口调用

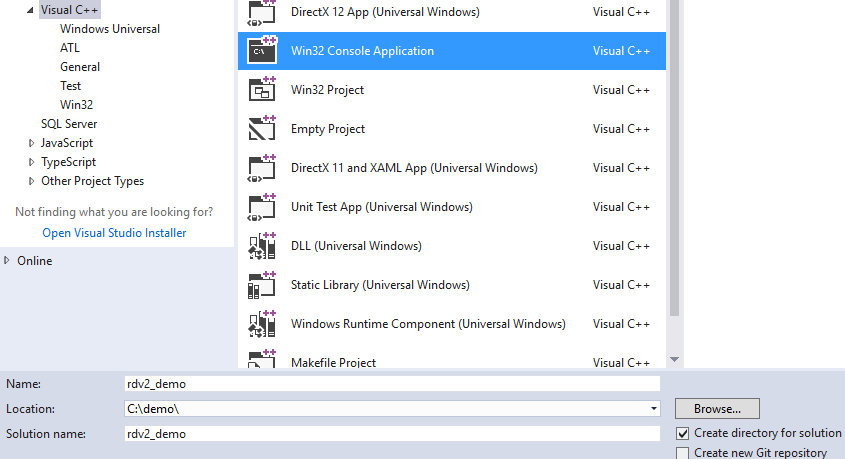
## 开发依赖

rdv2的开发依赖所需要的头文件在include文件夹内，动态库和输入库在lib\x64文件夹内。

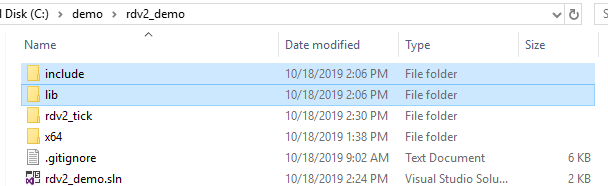
## 新建项目

### Visual Studio2017

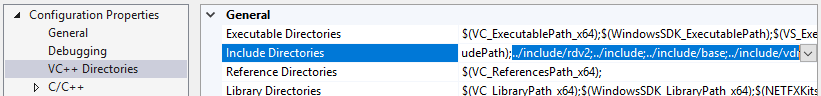
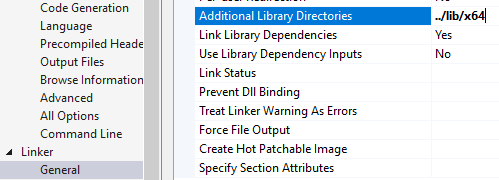
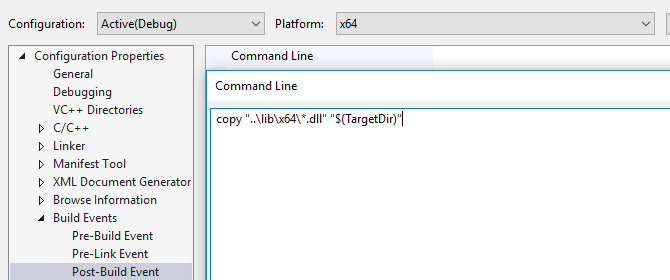
新建Win32 Console Application项目，Solution Platforms选用x64：

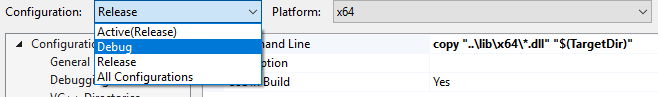


将开发依赖所需要的头文件和库拷贝至项目文件夹：



修改项目rdv2\_demo属性：

1. Include Directories  
   添加../include/rdv2;../include;../include/base;../include/vdr四个路径。
2. Additional Library Directories  
   添加../lib/x64至附加库文件夹。
3. Post-Build Event  
   在Command Line中添加：copy "..\lib\x64\\*.dll" "$(TargetDir)"，注意两个路径的引号之间有一个空格。
4. Configuration: Debug & Release



如果客户端项目需要发布Debug和Release两个不同的版本，则Debug和Release的配置中上述三个设置都需要同样设定。

## API接口

为了方便客户使用，通过rdv2对象可以同时调用rdv2接口和VDR原始接口。rdv2接口命名方式为add\_xxx形式，而VDR原始接口命名方式为AddXxx形式。确保调用的一致性，最好不要混用接口，尽可能使用rdv2接口。如果您对接口有新需求或有任何问题，请联系：[13611050407@139.com](mailto:13611050407@139.com)。

## 创建VDR仿真模型

按照VDR模型的实际需求，加入到仿真平台的节点常用任务通常如下：1. 操作VDR；2. 写入数据； 3. 读取数据； 4. 获取周期性时钟节拍； 5. 监听数据更新；6. 响应运行控制指令。下面就上述几种使用场景分别示例，实际应用则是多个场景的组合。

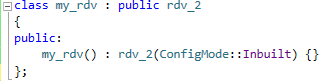
创建rdv2对象有两种方式：直接创建；或定义rdv2的派生类，然后创建派生对象。其中的区别在于后者可以通过覆写on\_xxx函数，实现对时钟回调、运行控制指令的定制响应方式。

### rdv2的配置接口

rdv2提供了三种配置接口：1. 配置类；2. XML配置文件（rdv2\_config.xml）；3. INI配置文件（rdv2\_config.ini）。客户端程序使用三种不同的配置接口生成或加载配置后，在启动连接前，都可以通过调用配置接口的set、get来读写配置项，其中的数值类型均为UINT16（见示例项目rdv2\_tick）。若使用配置文件，且指定的文件不在当前工作目录中，则加载异常。



#### 配置类



配置类不使用外部配置文件。下图是配置类各配置项的默认值，如有需要，在调用startup()连接VDR前，可以根据实际运行情况调用set来设置某些配置项的值。



#### INI配置文件



使用INI配置文件，如果设定global\_ini，则使用指定的全局配置文件覆盖部分设置。（注意：要正确设置你的应用程序与全局配置文件的相对路径，使用/或\\作为路径分隔符）



#### XML配置文件

XML配置文件是rdv2缺省配置方式，默认读取当前目录中的rdv2\_config.xml，如果设定global\_ini，则使用指定的全局配置文件覆盖部分设置。（注意：要正确设置你的应用程序与全局配置文件的相对路径，使用/或\\作为路径分隔符）



### 连接VDR

使用某种配置方式创建rdv2对象（或自定义的rdv2派生对象）后，若当前各配置项数据无需修改，就可以调用startup连接VDR了。

bool startup(*uint64\_t* time\_out\_ms = 5000, bool retry = true)调用参数决定了如果连接VDR服务器失败时的流程控制：time\_out\_ms设定重新连接VDR服务器的时间间隔（毫秒），retry设置是否重连，如果传入false不尝试重新连接，则startup调用直接返回false，此时程序需要检测返回值以确认连接是否成功。如果传入true，startup只有在连接VDR服务器成功后才会返回，否则就会循环重新连接，这时就无需检测返回值，因为只有连接成功调用才会返回。

常用的VDR操作如下：添加数据集、添加数据、添加节点、创建访问接入点。（见示例项目rdv2\_operate）

#### 添加数据集

add\_dataset向VDR中添加指定名称的数据集，参数2指定若VDR中已有同名的数据集时是否覆盖，缺省为覆盖。添加成功则返回此数据集的访问接入点，否则返回nullptr。

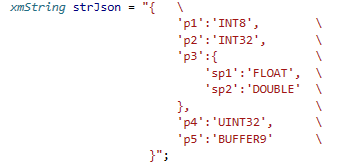
#### 添加数据

add\_data向指定的数据集中添加数据，数据类型包括：各种数值类型、字符串、字节数组、定制结构数据、二进制数据块。如果指定名称的数据集不存在，或指定名称的数据已经存在，则调用异常。成功添加数据后返回此数据的访问接入点。（注意：只能向自己创建的数据集中添加数据。）

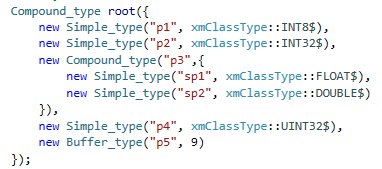
下面重点介绍rdv2接口封装的两种数据。

1. **定制结构数据**

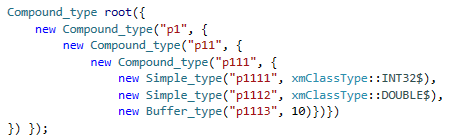
VDR的原始接口是通过字符串描述定制数据结构，难于输入，容易出错，特别是对于多层嵌套的数据结构。



rdv2则通过Compound\_type的嵌套定义，组合Simple\_type和Buffer\_type可以简单明了地声明多层嵌套的数据结构。

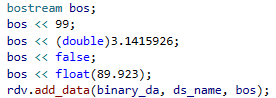


即使是较深的层次嵌套也可以很方便地定义、阅读，而且是语法验证的。



1. **二进制数据块**

二进制数据块的接口是rdv2新增的，主要是方便用户读写包含各种类型数据的块数据，支持不同字节序方式的数据写入和读取。数据支持：各种数值数据、布尔、字符串、字节数组、vector<uint8\_t>。



#### 添加节点

读写VDR数据集的数据有两种方式：通过节点或数据集。add\_node添加指定名称的节点，并连接到指定名称的数据集。若数据集不存在或已有同名的节点，则调用异常。添加节点成功则返回该节点的访问接入点，否则返回nullptr。link\_node将指定的数据集连接至已添加的节点上。

#### 创建访问接入点

VDR原始接口通过访问接入点读写数据，rdv2接口大大简化了数据的读写，提供了set\_value和get\_value，基本上不再需要创建访问接入点。这个接口主要是用于兼容VDR原始接口。访问接入点是基于数据集名称的，attach指定的名称可以是数据集名称（参数2==true），也可以是数据名称（参数2==false）。如果调用使用数据名，返回的是该数据所属数据集的访问接入点，VDR并不存在针对某个数据的访问接入点。

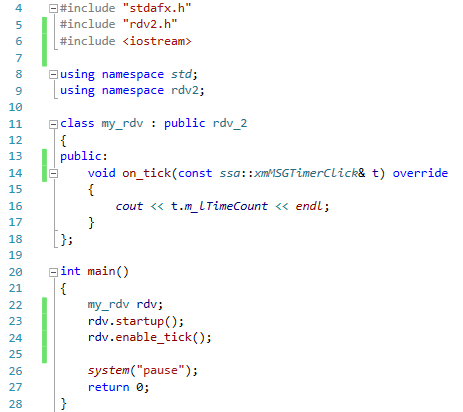
### 写入数据

rdv2接口提供了set\_value写入指定名称的数据，或set\_value\_n通过创建的节点来写入指定名称的数据，其调用返回值指示数据写入是否成功。（见示例项目rdv2\_set\_value和rdv2\_set\_value\_n）

### 读取数据

rdv2接口提供了get\_value读取指定名称的数据，或get\_value\_n通过创建的节点来读取指定名称的数据。对于数值类型数据的读取，调用返回类型为std::pair，其first指示数据读取是否成功，second为数据读取成功时的数值，读取失败时为数据类型的缺省值；对于其他数据类型，如字符串、字节数组、结构数据、二进制数据块，其调用返回值指示数据读取是否成功。（见示例项目rdv2\_set\_value和rdv2\_set\_value\_n）

### 获取周期性时钟节拍



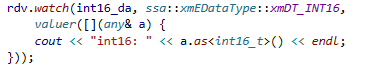
用户通过创建rdv\_2的派生类，覆写on\_tick函数来实现时钟节拍的回调处理。在startup启动VDR后，enable\_tick打开时钟回调。（见示例项目rdv2\_tick）

### 获取平台数据更新通知

#### 使用rdv2简化接口

VDR模型有时需要获取特定数据的更新通知，rdv2提供watch接口简化VDR的敏感数据回调操作（调用watch注册数据更新通知时，注册的数据必须已经添加到VDR上，否则调用异常，其返回值指示注册是否成功）。如何定义更新处理例程，rdv2提供了三种定义数据处理函数的方式：lambda函数、全局函数和函数对象。（见示例项目rdv2\_watch）

1. lambda函数



使用lambda函数的好处是数据更新注册和处理的代码在一起，方便阅读，而且通过捕获列表可以方便地取用其作用域内的任意变量。

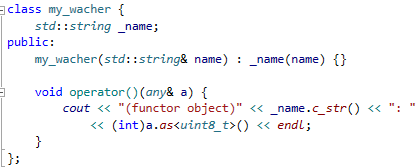
1. 全局函数





由于隐藏的this指针的存在，不能使用类成员函数，只能使用全局函数或类静态函数。

1. 函数对象





函数对象相对复杂一点，但好处是可以通过定义类成员变量，保存任何状态，适用于复杂的数据处理。

若模型不再需要接收某个数据的更新通知，调用unwatch取消。

#### 使用VDR接口

rdv2同时提供了VDR原始敏感数据回调接口，用于某些特殊情形。使用此接口需要创建rdv\_2派生类，覆写on\_data\_update函数，在此函数中对所有关心的数据更新进行处理。调用dispatch来向VDR平台声明哪些数据的更新需要触发回调（若传入的数据名不存在，则调用异常）。（见示例项目rdv2\_dispatch）

### 运行控制指令

运行控制指令的数据名称及各指令值的设定在配置中，可以通过修改配置文件或直接调用配置类接口实现修改。（见示例项目rdv2\_cmd）

#### 接收运行控制指令

接收运行控制指令非常简单: 1. 调用listen\_cmd，传入参数true（传入false则不再接收运行控制指令），调用时平台必须存在运行控制指令数据名，否则调用失败； 2. 创建rdv2的派生类，覆写相应的控制指令响应函数。

#### 发送运行控制指令

发送运行控制指令前需要先向平台注册控制指令数据所在的数据集，register\_cmd使用参数指定的数据集名称来完成注册，若该名称的数据集不存在，则会添加到VDR中。若运行控制指令数据名已经存在，则调用异常；调用返回值指示指令注册是否成功。需要发送指令时，调用send\_cmd，参数2在发送设定仿真步长指令时使用，返回值指示发送是否成功。

## 增强接口

### 模型应用配置数据

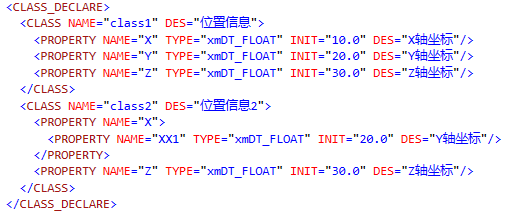
每个VDR模型都有一个配置文件，用于描述模型使用的数据及结构，其中<OUTPUT\_DATA>配置的输出数据需要注册到平台。rdv2提供接口解析模型的配置文件，将配置数据注册至平台。（见示例项目rdv2\_model\_conf）

#### 注册数据集和节点



接口将使用根节点的NODENAME属性值向平台注册节点和数据集，其中<OUTPUT\_DATA>配置的输出数据将添加至此数据集中。

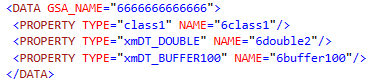
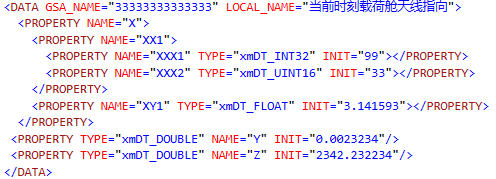
#### 定制结构数据



<CLASS\_DECLARE>用于定制结构数据，方便重复使用。PROPERTY定义数据项，允许多级嵌套，支持所有数值类型和BUFFER，TYPE="xmDT\_BUFFER100"表示长度为100，若没有指定则为512。此处定义的CLASS可以用于随后的<OUTPUT\_DATA>定义中，但不能用于CLASS自身的定义（class2的定义中不能使用class1）。

#### 定义输出数据

<DATA\_DECLARE><OUTPUT\_DATA><DATA>用于定义模型的输出数据。<DATA>定义数据，PROPERTY定义数据项，允许多级嵌套，支持所有数值类型和BUFFER，还可以使用之前定义的CLASS。



#### 配置数据导入平台

调用接口函数*export\_model\_conf*，参数为模型配置文件名，若不指定参数2，则将<OUTPUT\_DATA>定义的所有<DATA>数据将注册到平台，完成数据的初始化；若指定参数2，则将参数2列表中每一个标签里定义的所有<DATA>数据注册到平台。

#### 获取模型配置文件相关数据

1. model\_inst\_name获取配置文件根节点INSTNAME属性值；
2. model\_node\_name获取配置文件根节点NODENAME属性值；
3. inst\_serv\_entry获取配置文件根节点INST\_SERV\_ENTRY属性值；
4. local2gsa获取配置文件中所有<DATA>元素的LOCAL\_NAME到GSA\_NAME的映射表；
5. gsa2local获取配置文件中所有<DATA>元素的GSA\_NAME到LOCAL\_NAME的映射表。（若某个名称没有设定，则二者相同）
6. parse\_data解析配置文件某个标签里的所有<DATA>元素，返回一个数据名到数据类型对象指针的映射表，数据类型类基于Composite设计模式，通过递归其子节点列表可以访问到所有嵌套的PROPERTY定义。

### 二进制数据流

rdv2提供了一个数据流化接口，方便用户以指定的字节序逐个写入或读取不同类型的数据（数值类型、布尔、字符串、字节数组、字节vector）。bostream用于各数据的流化，调用其buffer()接口获取其内部数据区的首地址和数据字节数；bistream可以将任意数据块装入输入流（通过字节数组的首指针和尾后指针、首指针和字节长度、字节vecotr对象），然后将数据逐一取出。（见示例项目rdv2\_bstream）

### 理解VDR数据集的运作方式

VDR核心的功能就是数据交换，更新数据集将导致该数据集上已有数据清零。rdv2提供的接口有2个会更新数据集：

1. add\_data向数据集添加某个数据；
2. *register\_cmd*注册运行指令数据。

为了防止数据集更新，在调用上述接口前，可以先调用begin\_update，之后调用上述接口时将不会更新数据集（注意，add\_data返回nullptr）。添加完某个数据集的所有数据之后，调用end\_update更新数据集（此时所有添加的数据才会真正在该数据集上分配内存）。然后就可以正常读写数据了。export\_model\_conf会将模型配置文件中定义的数据注册到VDR上，由于涉及到初始数据的写入，所以不需要进行上述操作。