ICS 03.060
A11
备案号

JR

# 中华人民共和国金融行业标准

JR/T 0103—2014

# 证券交易数据交换编解码协议

Data exchange encoding and decoding protocol for securities trading

2014-02-10 发布　　2014-02-10 实施

中国证券监督管理委员会　　发布

# 目　　次

# 前　言

本标准依据GB/T 1.1-2009给出的规则起草。

本标准由全国金融标准化技术委员会证券分技术委员会（SAC/TC68 SC4）提出。

本标准由全国金融标准化技术委员会（SAC/TC68）归口。

本标准起草单位： 上海证券交易所、上证所信息网络有限公司、深圳证券交易所、深圳证券通信有限公司。

本标准主要起草人：徐广斌、吴韶平、黄越、朱世东、徐乾、王海、徐伟华、巫禄芳。

# 证券交易数据交换编解码协议

## 1 范围

本标准规定了证券交易所交易系统与市场参与者系统之间使用金融信息交换协议（FIX协议）或证券交易数据交换协议（STEP协议）时，证券交易数据交换的编解码协议（Data Exchange Encoding and decoding Protocol for securities trading, 简称DEEP），包括传输数据的编码、解码和会话传输等内容。

本标准适用于证券期货交易所、市场参与者等相关金融机构之间的交易数据交换编解码，其他应用系统可参照执行。

## 2 规范性引用文件

下列文件中的条款通过本标准的引用而成为本标准的条款。凡是注明日期的引用文件，其随后所有的修改单（不包括勘误的内容）或修订版均不适用于本标准，然而，鼓励根据本标准达成协议的各方研究是否可使用这些文件的最新版本。凡是不注明日期的引用文件，其最新版本适用于本标准。

GB/Z 21025-2007　　XML使用指南

## 3 术语和定义

下列术语和定义适用于本文件。

3.1

**编码 encode**

将应用消息流串行化为二进制流的过程。

3.2

**解码 decode**

将二进制流反串行化为应用消息流的过程。

3.3

**模板 template**

本标准的编/解码方法依据该控制结构来进行，其对应上层的FIX/STEP应用消息，通过规定消息中字段的顺序和结构、字段运算，及其使用的二进制编码表示方法来控制与应用消息相对应的二进制流的编码和解码。

3.4

**会话控制 session control**

为消息传输的发起方和接收方之间的一次编码消息交换过程提供的会话发起和控制的手段。

## 4 基本应用模式

本协议的基本应用模式如图1所示，其中编码操作利用两个层次的处理降低消息尺寸：首先，进行字段编码时利用流中数据的相关性消除冗余数据；其次，进行传输编码时，利用自描述的字段长度以及标识字段是否存在的位图进行串行化处理。

如图1，消息编码和解码依据有状态的模板进行。模板所含的字段指令在运行上下文环境中执行，其包括了使用的模板集、应用类型集、字典集、以及字段的初值等。其中，字典是一种有状态结构，用以保存前一个包含该字段的应用消息的该字段的应用值，也即字段的前值。对编码和解码来说，由于字段运算要求消息收发双方能够维护一致的前值和字段状态，因此，会话控制机制要考虑维护收发双方之间模板状态一致性。

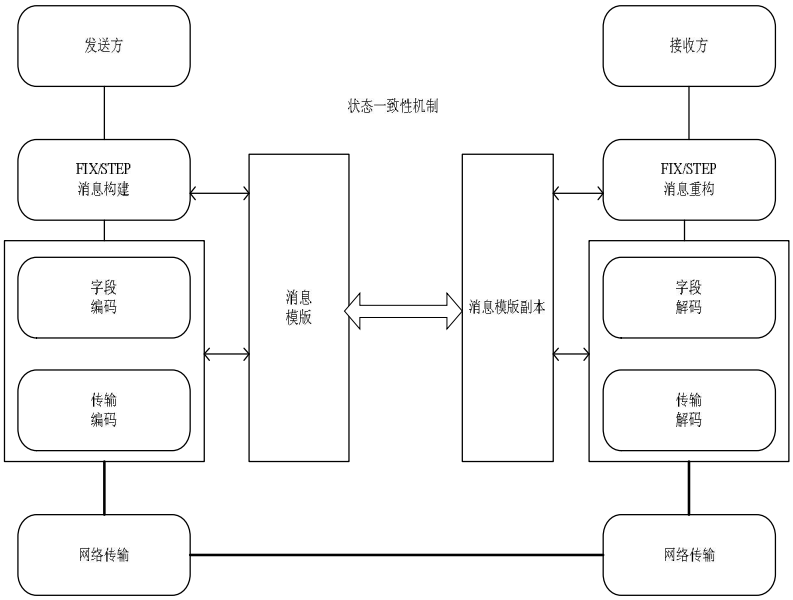


**图 1 协议基本应用模式**

后文主要对本协议的编码操作进行详细描述，鉴于解码为编码的反向操作，对解码的操作不再做赘述。

附录D给出了模板的实例。

## 5 应用类型

应用类型表示上层的FIX/STEP消息或分组的对应类型，本标准不对上层应用类型的具体结构进行规定，但是认为它们可以被映射为以下的抽象应用类型实体:

a) 分组：一个无序的字段的集合的带名实体。

b) 字段：具有名称和类型的实体。其中，名称在一个分组内唯一，类型则可以为基本类型、序列、或者分组中的一种。

c) 序列：包括长度及一个组成体的有序集合。其中，所含的每一组成体均为分组类型。

d) 位组：一个无序的字段的集合的带名实体，用于将 2 个或 2 个以上小于 8 位的字段填充到一个停止位编码字段中。

e) 基本类型：可以为 ASCII 码字符串、Unicode 字符串、32 位无符号整数(UInt32)、32 位整数(int32)、64 位无符号整数(UInt64)、64 位整数(int64)、浮点数、字节向量、布尔、枚举、集合、二进制整数中的一种。

f) 消息：流内最高层的分组。

## 6 字段编码

### 6.1 模板

模板主要用于规定如何将应用类型的一个实例编码为字节流。模板用名称来进行标识，并可以通过名称对另一模板或外部定义的模板进行引用。

为提供人机可读、规范化和完整性高的格式，本标准默认将模板定义为XML格式，并使用RELAX NG XML模式描述语言对模板XML进行描述。

一个模板定义XML文档可以包含单个的模板或多个模板的集合。模板集合必须封装在<td:templates>元素里。

模板包含指令的一个序列，指令的顺序应与数据在流中的位置相对应。指令分为字段指令和模板引用指令两大类。字段指令规定了如何将字段的实例编码成为流，模板引用指令则提供了通过引用其他模板来定义模板的一个部分的方法。

### 6.2 指令上下文

编/解码在指令的上下文中进行，指令的上下文包括：

a) 模板的集合。

b) 当前模板。

c) 应用类型的集合。

d) 当前应用类型。

e) 字典集合。

f) 可选的初值。

当前应用类型在初始时为特殊的“任意”类型。当处理遇到到包含“<td:typeRef>”的元素时当前类型发生改变。新类型适用于该元素所包含的指令。“<td:typeRef>”元素可包含于“<td:template>”、“<td:group>” 和“<td:sequence>” 元素中。

当前模板指正被处理的模板，在流中处理到一个模板标识符时当前模板被更新。对静态模板的引用也可引起当前模板的改变。

### 6.3 字段指令

#### 6.3.1 基本要求

每一字段指令均具有名称和类型：名称标识了当前应用类型中的相应字段；类型则规定了字段的基本编码方法。

可选的“存在(presence)”参数用来说明字段是必选(mandatory)字段还是可选(optional)字段。如果该参数未被指定，则字段为必选。

基本类型字段，也即除了分组、序列、位组之外的字段，可具有一个字段运算符(field operator)，该运算符规定了对字段的特定优化操作。

#### 6.3.2 整数

整数在传输编码中的大小不受限，值的编码和解码方法不受整数大小的影响。但通常上层应用会采用固定长度的整数。整数字段指令因此必须对整数的边界进行指定。元素名称中的数字则指示了整数字段指令的位大小。

前缀“int”表示字段是带符号数，“uInt”则表示字段是无符号数。

### 6.3.3 浮点数

浮点数字段指令由两部分表示：指数(exponent)及尾数(mantissa)。该指令可包含用于整个浮点数的单个字段运算符，或两部分各自使用的两个运算符。如果为尾数和/或指数单独指定了运算符，则为了合成浮点数型数，运算符被单独应用于每一部分。指数和尾数字段运算符的操作数为带符号整数，分别为int32和int64类型。如果没有为整个浮点数指定运算符，或只指定了单个运算符，则操作数为一个浮点数，在传输编码中由一个比例数(Scaled Number)表示。

如果浮点数字段是可选存在的，并具有单独的运算符，则尾数存在与否取决于指数是否存在。

为浮点数字段指令指定的初值将被规范化，相关内容见后文。

### 6.3.4 字符串

字符串字段指令具有一个指示字符串所使用字符集的、可选的“字符集（charset）”参数。协议支持的两种字符集为：ASCII和Unicode，分别由参数值“ascii”和“unicode”表示。如果没有指定该参数，则字符集为ASCII。根据指定的字符集，传输编码中字符串可由ASCII字符串或Unicode字符串表示。如果字符串为Unicode格式，可对可选的”<td:length>”元素进行指定，并可为其基础字节向量的长度前导(length preamble)命名。

### 6.3.5 字节向量

字节向量字段指令表示字段在传输编码中由一个字节向量表示。

可以通过指定“<td:length>”元素将字节向量的长度前导和指定名称关联。“<td:length>”的使用不会影响字节向量在流中的编码方式，而只是作为处理器向应用层报告长度的一个手段。该字段逻辑上为uInt32类型。

### 6.3.6 布尔

布尔类型的字段用一个无符号整数来表示，其中0表示假，1表示真。布尔值可以为位组成员之一，在这种情况下，布尔值占用1位（0/1）。

可选存在的布尔字段使用类似于可空的整数的表示方式，以0表示空，1表示假，2表示真。在位组中，一个可空的布尔字段占2个位。

### 6.3.7 枚举

对于FIX/STEP协议的某些字段，取值范围为一个固定的可选值的集合，枚举类型支持取值范围为固定数量值的单值字段。

举例来说，如果一个字段的值可以为M1, M2, M3, M4四个中的一个，这四个值的编码值可以分别被指定为：0，1，2，3，则字段的编码值可以为0到3中的任一个。

如果该字段是可空的，则可类似于可空整数的定义方法，指定0来表示空值。以具有4个可选值的枚举字段的上例为例，编码值将分别为1，2，3和4，0则保留作为空值的表示。

可以在位组中包含枚举值，以利用最少的位来表示枚举字段中的所有值。

### 6.3.8 集合

当FIX/STEP协议的某些字段取值范围为固定的可选值集合，并且具体字段值可能为多值时，可以使用集合字段指令来进行编码。例如，如果一个字段的值可以为A, B, C, D四个中的一个或多个。这四个值可以分别被指定为1，2，4，8（也即每个值分别由1个位来表示），则可以通过对它们进行相加来形成这四个值的组合的值。例如，A及C的值可以使用1+4=5作为组合的编码字段的值。

如果该字段是可空的，可类似于可空整数的定义方法，指定0来表示空值。可以在位组中包含集合值，以利用最少的位来进行表示。

### 6.3.9 序列

序列字段指令表示应用类型的字段为序列类型，需要重复应用其包括的指令分组以对其所含的每一组成体进行编码。如果该分组中的任一指令需要对应在存在位图中占用一个位，则在传输编码中每一组成体用一个段(segment)表示。

序列具有一个相关长度字段，其包含一个无符号整数，用以表示编码组成体的个数。若流中出现长度字段，则必须紧接在编码的组成体的前面出现。该长度字段可具有名称，类型为uInt32，且可具有字段运算符。其命名方式可为以下两种之一：

a) 隐式(implicit)：名称自动生成，并特定于序列字段的名称。该名称不能与模板中显式指定的字段名冲突。

b) 显式(explicit)：名称在模板定义中显式地指定。

序列可为必选(mandatory)或者可选的(optional)。若序列为可选的，则其长度字段也是可选的。

序列指令由“<td:sequence>”表示。可在任一指令前，可选地包含一个“<td:length>”子元素。该元素规定长度字段的属性。如果该元素具有“名称(name)”参数的话，则为显式命名方式，否则为隐式命名方式。如果没有指定<td:length>元素，则长度字段为隐式命名，且无字段运算符。

### 6.3.10 分组

分组字段指令将一个指令分组和某个名称及存在参数相关联。如果分组中的任一指令需要在存在位图(Presence Map, 或PMAP)中占用一个位，则在传输编码中该分组由一个段表示。

分组字段指令的主要用途是使得可以用存在位图中的单个位来指示一整组的字段存在与否。因此，当前应用类型并不要求具有与分组所对应的概念。

### 6.3.11 位组

位组字段指令结构与分组字段指令类似。但在停止位编码时，隶属于一个位组的2个或2个以上的字段将填充到一个停止位编码字段中。其中的每个位字段具有小于8位的固定大小。位字段支持的类型包括枚举类型，集合类型，2-7位的带符号整数，以及1-7位的无符号整数。新的整数类型分别被命名为Int2-Int7, 以及uInt1-uInt7。1位带符号整数由于只有1位的符号位，故不在考虑之列。

位组使用普通的停止位编码字段的传输表示方式。组内的字段按照从左至右来分配空间。例如，具有A (2位), B (1位) 和 C (1位)共3个字段的位组将在一个停止位编码的字节中占用共4个位。则字段在该字节中的放置为：SAABCxxx，其中S是停止位，xxx是因未使用而必须置为0的3个位。以位组表示的枚举类型字段和集合类型字段将使用最少的位来编码，以表示该类型的所有的值。

一个位组可以使用多个字节，位组中一个单独的字段也可能会跨两个字节的边界。

### 6.3.12 二进制整数

二进制整数字段指令可用来表示不限长度的整数字段值，带有一个长度前导。通过指定“〈td:length〉”元素将长度前导和指定名称关联。前缀“binInt”表示字段是带符号数，“uBinInt”则表示字段是无符号数。

## 6.4 字段运算

### 6.4.1 字典与前值

字段运算符(field operator)规定了字段编码的优化方式。

一些运算符依赖于前值(previous value)，在带名的字典中维护。字典是一个条目集合，每一条目具有名称及特定类型的值。值可为以下三种的状态之一：未定义(undefined)，空(empty)，已定义(assigned)。在协议处理开始时，所有值的状态均为未定义。已定义状态表示前值存在，空状态表示前值不存在。空状态只适用于可选的字段。

字典的名称在字段运算符元素的“字典（dictionary）”参数中指定，如果已有多个在先的“字典”参数，则使用位置最近的元素的该参数。如果没有指定该参数，则使用全局字典。

以下三种字典是预定义的:

a) 模板(template)字典：字典限于当前模板使用。也就是说：当且仅当 T1=T2，模板 T1 中运算符与模板 T2 中运算符共享相同的字典。
b) 类型(type)字典：字典限于当前应用类型使用。也就是说：当且仅当 A1=A2，应用类型 A1 的模板 T1 中的运算符与应用类型 A2 的模板 T2 中的运算符共享相同的字典。
c) 全局(global)字典：字典为全局的。无论模板和应用类型如何，所有运算符共享相同的字典。

其余字典均称为用户自定义(user defined)字典。当且仅当指定的字典名称相同时，两个运算符才共享同一用户自定义字典。

字典可以被显式地重置(reset)。重置一个字典将把其所有条目的状态设置为未定义。

### 6.4.2 初值

初值(Initial Value)由运算符元素的“值(value)”参数指定。该值可通过后文中定义的方法转换为字段的类型。

如果字段为浮点数类型，类型转换得到的值将被正规化。正规化是对尾数和指数进行调整，使得尾数整除10的余数不为0：即尾数%10≠ 0. 例如$100*10^0$ 将被正规化为$1*10^2$ 。如果尾数为0，则正规化浮点数的尾数为0，指数也为0。

### 6.4.3 常量

常量运算符(Constant Operator)表示字段的值均相同，即为初值。

常量字段的值不被传送。

常量运算符适用于所有字段类型。

### 6.4.4 缺省

缺省运算符(Default Operator) 表示字段的值在流中出现，否则为初值。如果字段无初值、且是可选存在的，则当流中字段的值不存在时，字段被认为不存在。

缺省运算符适用于所有字段类型。

#### 6.4.5 拷贝

拷贝运算符(Copy Operator) 表示字段值在流中是可选地存在的。如果值在流中存在，则成为新的前值。

当值在流中不存在，则根据前值的状态分为以下三种情况作处理：

a) 已定义状态：字段值为前值。
b) 未定义状态：字段值为初值，并成为新的前值。如果字段是可选存在的，并且无初值，则该字段被认为不存在，前值变成空状态。
c) 空状态：则字段的值为空。如果字段是可选的，则值被视作不存在。

拷贝运算符适用于所有的字段类型。

#### 6.4.6 递增

递增运算符(Increment Operator) 表示字段的值在流中是可选存在的。如果值在流中存在，则成为新的前值。

当值在流中不存在，则根据前值的状态分为以下三种情况处理：

a) 已定义状态：字段值为前值加 1。计算得到的值成为新的前值。
b) 未定义状态：字段值为初值，并且成为新的前值。如果字段为可选存在的，并且无初值，则字段被视作不存在，同时前值的状态变为空。
c) 空状态：字段值为空。如果字段是可选的，则值被视作不存在。

递增运算符适用于整数、枚举字段类型。

整数的递增为自加1。如果其值已经是类型最大值，则递增后的值为最小值。

#### 6.4.7 差值

##### 6.4.7.1 基本要求

差值运算符(Delta Operator) 表示流中存在差值。如果字段是可选存在的，则差值可为空。如果是这种情况，则该字段值被视作不存在。其余情况，字段值通过差值(delta value)和基值(basic value)的结合得到。

根据前值的状态，基值按照以下方式来确定：

a) 已定义状态：则基值为前值。
b) 未定义状态：如果指令上下文中存在初值，则基值为初值。除此之外，使用由类型决定的缺省基值。
c) 空状态：如果前值为空则错误。

以下小节规定了差值的表示方法、缺省基值，以及如何根据类型进行值的结合。

##### 6.4.7.2 整数的差值

差值在传输编码中由一个整数差值表示。结合值是基值和差值的总和。

缺省的整数基值为0。

##### 6.4.7.3 浮点数的差值

差值在传输编码中以一个比例数字差值表示。通过将差值和基值的指数和尾数部分分别对应相加来计算结合的值。

浮点数的缺省基值为0，缺省基值的指数为0。

#### 6.4.7.4 ASCII 码字符串的差值

差值在传输编码中以ASCII码字符串差值表示。差值的减除长度(subtraction length)规定了要从基值前端或后端移除的字符的数量。当减除长度为负时，从前端移除。差值的字符串部分表示要添加到基值与减除长度的符号所指定的同一端方向上的字符。

减除长度使用“额外减一(excess-1)”的编码方式：解码时，如果值为负，则加1后得到减除的字符数量。这使得可以将负0编码为-1，以使得可在无需移除任何字符的情况下在前端进行增加的操作。

缺省基值为空字符串。

#### 6.4.7.5 Unicode 字符串的差值

Unicode字符串的差值,除了具有差值中字符向量的内容必须为UTF-8类型字节的额外约束外，在结构上与字符向量相同。差值是对编码字节，而不是Unicode字符进行操作，因此差值可能由一个不完整的UTF-8字节序列构成。

#### 6.4.7.6 字节向量的差值

差值在传输编码中由一个字节向量差值表示。差值的减除长度规定了从基值前端或后端移除的字节数量。当减除长度为负时，字节从前端被移除。差值的字节向量部分表示要添加到基值与起始的减除长度的符号所指定的同一端方向上的字节。

减除长度使用与ASCII码字符串情况相同的“额外减一”的编码方式：如果解码时值为负，则加1后得到要减除的字符数量。

缺省的基值为空的字符向量。

### 6.4.8 尾运算

#### 6.4.8.1 基本要求

尾运算符(Tail Operator)表示在流中可选地存在一个尾值。

如果字段为可选存在的，则尾值可为空。若为此情况，则字段值被认为不存在。否则，如果尾值存在，则通过将尾值和基值结合来得到字段值。

根据前值的状态，基值可根据以下方式确定：

a) 已定义状态：则基值为前值。
b) 未定义状态：则如果指令上下文中存在初值，则基值为初值。其余情况，使用基于类型的缺省基值。
c) 空状态：如果指令上下文中存在初值，则基值为初值。其余情况，使用基于类型的缺省基值。

结合的值成为新的前值。

如果尾值在流中不存在，则根据前值的状态，字段值以下方式进行确认：

a) 已定义状态：字段值为前值。
b) 未定义状态：字段值为前值，且成为新的前值。除非该字段为可选存在的，否则如果指令上下文无初值，错误。如果字段为可选存在且无初值，则字段被认为不存在，且前值的状态变为空。
c) 空状态：字段值为空。如果字段是可选地，则值被视为不存在，如果字段是必选的，则错误。

以下定义了尾值的表示方法、缺省的基值，以及如何根据类型进行值的结合。尾运算符只能应用于这些类型上。

#### 6.4.8.2 ASCII 字符串的尾运算

尾值在传输编码中以ASCII码字符串表示。字符串的长度规定了从基值的后端移除的字符的数量。尾值表示附加到剩余字符串上的字符。

如果尾值的长度超过了基值的长度，则结合后的值成为尾值。

缺省的基值为空字符串。

#### 6.4.8.3 Unicode 字符串的尾运算

Unicode字符串的尾值,除了具有差值中字符向量的内容必须为UTF-8类型的字节的额外约束外，在结构上与字符向量的尾值相同。尾值操作是对编码字节而非Unicode字进行操作，因此，尾值可能为一个不完整的UTF-8字节序列。如果结合值不是一个合法的UTF-8 序列的话，则错误。

#### 6.4.8.4 字节向量的尾运算

尾值在传输编码中以字符向量表示。尾值的长度规定了从基值的后端移除的字节的数量。尾值表示附加到剩余字节向量的字节。

如果尾值的长度超过了基值的长度，则结合后的值成为尾值。

缺省的基值为空字节向量。

### 6.5 模板引用指令

模板引用指令(Template Reference Instruction)规定该模板的一部分由另一模板来定义。模板引用可以是静态或者动态的。当指令中规定了名称时，引用为静态的(static)，否则为动态的(dynamic)。

静态引用表示应当将引用的模板作为当前模板来继续处理。静态引用并不需要流中一定有存在位图或模板ID。

动态引用规定了流中存在一个存在位图和模板标识符(template identifier)，应该以该标识符所指示的模板作为当前模板继续进行处理，并在传输编码中以一个段来进行表示。

## 7 名称

模板定义中的名称由两部分组成：命名空间URI (namespace URI)和本地名称。

应用类型、字段和运算符关键字的命名空间URI由“ns”参数指定，它可与本地名称出现在相同的元素中，也可出现在任一在先元素中。如果在先部分已经出现了多个ns参数，则采用位置最近元素的该参数。如果没有指定“ns”参数，则命名空间的URI为空字符串。

模板的命名空间URI由“templateNs”参数指定，其继承方式与ns参数相同。为模板名称单独设置一个参数的原因是消息和文件名常常使用同一个标准化的命名空间,但是模板名称则经常包含在供应商规定的命名空间中。

采用URI的命名空间并不表示其必须指向一个资源。此处使用URI形式仅仅是为了对语法进行规范，同时也推荐使用诸如公司或者组织的URL来生成全局唯一的命名空间。

本地名称由参数“名称(name)”指定。

两个名称相同，当且仅当它们的命名空间标识符和本地名称都相同。

## 8 类型转换

### 8.1 基本要求

如果模板的某个字段的类型与当前应用类型所对应的字段的类型不一致，则对字段进行编码或者解码时，必须进行值的转换。本节对不同类型间进行两两的转换进行了定义。

字节向量只能与字符串进行相互转换，与其余类型的转换均为错误。

## 8.2 字符串转换为其他类型

### 8.2.1 基本要求

下文中的“空白剪切(whitespace trimming)”是指在字符串被解译前将位于起始和结尾处的空白移除的操作。下列十六进制ASCII码的字符被视作空白：20（空格），09（横向制表符），0D（回车），and 0A（换行）。

### 8.2.2 转换为整数

经过空白剪切，字符串被作为’0’—’9’数字的序列诠释。如果为带符号类型，则允许以起始的减号来表示负数。

### 8.2.3 转换为浮点数

经过空白剪切，字符串具有一个整数部分和一个小数部分。可对两者进行指定，或者只对其中之一进行指定。如果对两者都进行了指定，则它们之间必须带有小数点。起始的减号表示是负数。例如，1，1.1，.1，和-0.1都为允许的表达方式。

### 8.2.4 转换为字节向量

字符串被作为偶数个的[0-9A-Fa-f]的十六进制数字解译，其中可能夹带有空白。首先通过去除所有的空白，文字转换为一个字节向量。然后，其中的每对字符被作为表示单字节的十六进制数解译。

### 8.2.5 字符集间的相互转换

ASCII是Unicode的一个子集，因此将ASCII码字符串转换为Unicode字符串较为简单。如果Unicode字符串只包含ASCII字符的话，可以转换为ASCII码字符串。

## 8.3 整数转换为其他类型

### 8.3.1 转换为整数

只要没有精度丢失，不同类型的整数可以进行相互的转换。

### 8.3.2 转换为浮点数

如果可以由一个指数范围为[-63，63] 、尾数为int64的比例数表示，则一个整数可以被转换为浮点数。

### 8.3.3 转换为字符串

数字由‘0’ —‘9’的一个数字序列表示。数字不能以0作为起始。如果类型是带符号的，且数字为负，则数字序列的前面接一个减号(‘-‘)。

## 8.4 浮点数转换为其他类型

### 8.4.1 转换为整数

当前仅当无小数部分时，一个浮点数可以转换为一个整数。也就是说，其值实际上为一个整数。

#### 8.4.2 转换为字符串

如果数字实际上是一个整数，则其当作整数类型进行转换。否则数字由以小数点分隔的一个整数部分和一个小数部分表示。每一部分均为‘0’ – ‘9’的数字序列。小数点的两边都必须至少有1个数字。如果数字为负的，则前面接减号(‘-‘)。整数部分不能以0开始。

### 8.5 字节向量转换为字符串

字节向量由偶数个[0-9 a-f]的十六进制数字的序列表示。其中每一对是表示向量中一个字节的十六进制数。

## 9 传输编码

### 9.1 基本要求

以下扩展巴科斯范式（EBNF）语法规定了DEEP流的总体结构。

stream ::= message* | block*
block ::= *BlockSize* message+
message ::= segment
segment ::= *PresenceMap TemplateIdentifier*? (field | segment)*
field ::= integer | string | delta | *ScaledNumber* | *ByteVector* | *BitGroupField* | BinInteger
integer ::= *UnsignedInteger* | *SignedInteger*
string ::= *ASCIIString* | *UnicodeString*
BinInteger::= *UnsignedBinInteger* | *SignedBinInteger*
delta ::= *IntegerDelta* | *ScaledNumberDelta* | *ASCIIStringDelta* | *ByteVectorDelta*

一个DEEP流(stream)为一个消息的序列(sequence)，或为一个块的序列。本标准不对特定的流使用何种方式作出规定。

块(block)是包含单个或多个消息的一个序列。块带有一个前导的、用来表示块包含的消息所占字节数量的块大小(BlockSize)。

每个消息(message)由一个段 (即消息段)来表示。

段(segment)带有一个头部，由一个存在位图和紧跟在后的一个可选的模板标识符构成。如果一个段为消息段，或者段作为动态模板引用指令的结果出现，则该段带一个模板标识符。模板标识符的编码与指定了拷贝运算符的情况相同。该拷贝运算符使用全局字典，且具有所有模板标识符字段公用的一个内部关键字。也就是说，带有模板标识符的段并不总是物理地包含模板的标识符。但是，段存在位图的第1个位固定由其拷贝运算符占用。

段主体是字段及可能的子段构成的一个序列。段的跨度范围由模板定义，并且依赖于存在位图的设置。

布尔、枚举、集合字段如果单独编码，则形式与整数字段相同。如果包含在一个位组中，则该位组对应一个段，并按填充的方式被编码为一个停止位编码实体，为段中所含的唯一的字段。

附录A给出了RELAX NG模式的XML模板描述。

### 9.2 字节和位顺序

所有的整数字段使用大尾端(big-endian)方式来表示,其中位和字节均采用网络字节顺序，即高位在低位前，高字节在低字节前。

### 9.3 停止位编码实体

传输编码的一个重要特性是采用了停止位编码实体(stop bit encoded entity)。一个停止位编码实体是一个字节序列，其中每个字节的最高有效位(the most significant bit)指示下一字节是否是实体的一部分。如果该位未被设置，则下一个字节属于该实体，否则其为最后一个字节。跟在停止位后的7个位为有效数据位(significant data bits)。通过链接每一字节的有效数据位来表示停止位编码实体的实体值。实体值的位数量总是7的倍数，其最小的长度为7位。

### 9.4 模板标识符

模板标识符在流中以一个无符号整数来表示。

如果一个解码器不能找到与流中出现的模板标识符所关联的模板，则错误。

本标准不对如何将一个标识符映射为模板名称作出规定,可选地可对模板标识符进行静态的分配。

### 9.5 空值属性

每个字段的类型具有一个空值(nullability)的属性。如果类型为可空(nullable)，则空值采用一种特殊表示方式。如果类型不可空(non-nullable),则不保留任何空的表示方式。所有的可空类型按照以下方式构建，即空由所有位均为0的7位实体值来表示。在使用停止位编码时，空由0x80来表示。

除非显示地指定,否则缺省使用不可空的表示方式。

### 9.6 存在位图

#### 9.6.1 基本要求

存在位图(Presence Map，或PMAP)为位的一个序列。存在位图代表的段中的字段根据当前模板的规定来使用位。

存在位图用停止位编码实体来表示。逻辑上，存在位图具有无限的0后缀。这使得可对以一串0结尾的存在位图进行截取，剩余部分长度则必须为7的倍数。

如果一个存在位图超过7位，且以7位或7位以上的0结尾，则其过长。

#### 9.6.2 存在位图与空值的使用

存在位图按照字段的条目的顺序分配位。也就是说，在模板中先出现的指令比后出现的指令所分配的位的顺序要高。位在当前段的存在位图中进行分配。

如果一个字段为必选的、且具有常量运算符，则其在存在位图中不占位。

具有常量运算符的可选字段占1位。如果该位被设置，则值为指令上下文的初值。如果该位未被设置，则值被视作不存在。

如果字段无字段运算符且为必选的,则其在存在位图中不占位,且其值必须在流中存在。

如果一个分组字段为可选的，则其在存在位图中占1位。当且仅当该位被设置时，该分组的内容的才可能在流中出现。如果该位未被设置,则分组中的指令将不被处理。也就是说，该分组的字段的前值不会由于一个分组不存在而受影响。如果消息在应用层表示没有分组的概念，则若一个分组在流中不存在，则其包含的每一个字段也被视作不存在。

如果字段是可选的，且无字段运算符，则其采用可空的表示方法进行编码，并且用空表示值不存在，其在存在位图中不占位。

缺省、拷贝和递增运算符对存在位图和空值的使用如下：

a) 必选的整数、浮点数、字符串和字节向量字段：占 1 位。如果被设置，则值在流中出现。

b) 可选的整数、浮点数、字符串和字节向量字段：占 1 位。如果被设置，则值在流中以可空的表示方式出现。空表示值不存在，前值的状态也将被设置为空，但是在使用缺省运算符时，前值的状态则不作更改。

布尔、枚举、集合，以及二进制整数字段的情况与整数相同。

差值运算符对存在位图的使用如下：

a) 必选的整数、浮点数、字符串和字节向量字段：不占位。

b) 可选的整数、浮点数、字符串和字节向量字段：不占位。差值在流中以可空的表示方式出现。空表示差值不存在。需要注意的是，如果值不存在，则前值不被设为空，而是保持不变。

尾运算符对存在位图的使用如下：

a) 必选的字符串和字节向量字段：占 1 位、

b) 可选的字符串和字节向量字段：占 1 位。尾值在流中以可空的表示方式出现。空表示值不存在，且前值被设置为空。

具有单独运算符的浮点数字段的使用情况如下：

a) 如果浮点数是必须存在的，则指数字段和尾数字段将作为两个独立的、必选的整数字段，按照前述的相关方式进行处理。

b) 如果浮点数是可选存在的，则指数字段被视作可选的整数字段，尾数字段被视作必选的整数字段。尾数字段及其在存在位图中相关位的存在与否取决于指数是否存在。当且仅当指数值被视作存在的，尾数字段才在流中出现。如果尾数的运算符需要在存在位图中占 1 位，则当且仅当指数值被视作存在，该位才可存在。

## 9.7 字段

### 9.7.1 整数

#### 9.7.1.1 基本要求

整数由停止位编码实体表示。如果一个整数在移除7位或7位以上的最高有效位后，实体值还是表示相同的整数，则该整数过长(overlong)。

如果一个整数可空，则所有的非负整数在被编码前被加1。可空整数的空由所有位为0的7位实体值表示。

#### 9.7.1.2 有符号整数

实体值为一个补码，最高有效位为符号位。

#### 9.7.1.3 无符号整数

实体值即为整数的二进制表示形式。

### 9.7.2 比例数

比例数(Scaled Number)与浮点数类似，由一个尾数和一个指数表示。

为了计算效率，浮点数使用以2为底的指数，但为了支持浮点数的确切表示形式，比例数采用了以10为底的指数。

$$数值=尾数*10^{指数}$$

其中，数值由尾数和10的指数次方相乘来得到。

比例数由一个带符号整数指数及其之后的一个带符号整数尾数来表示。

如果一个比例数可空，则其指数可空，但尾数不可空。空值的比例数由一个空的指数来表示。当且仅当指数不为空时，尾数在流中出现。

### 9.7.3 ASCII 码字符串

ASCII码字符串由一个停止位编码实体来表示。其实体值被作为7位的ASCII码字符的序列进行解译。

位序列若以7个0位作为开始，则被称为具有一个零前导。以零前导开始的字符串由移除该前导后的余下位组成。如果一个字符串在移除零前导后有剩余位，并且剩余位的前7个位不全为0，则称该字符串过长(overlong)。如果流中出现一个过长的字符串，则发生告错误。

如果一个字符串具有零前导(zero-preamble)，并且在移除零前导后无剩余位，则其表示一个空字符串(empty string)。

如果一个ASCII码字符串可空，则在字符串起始处允许增加一个附加零前导。跟随的位被当作不可空字符串进行解译，包括可能的零前导。如果在移除零前导后无剩余位，则其表示一个空串。

表1列举了零前导的应用规则：

**表 1 字符串零前导应用规则一览**

| 实体值 | 是否可空 | 说明 |
|---|---|---|
| 0x00 | | 空字符串 |
| 0x00 0x00 | | “\0” |
| 0x00 0x41 | | “A”,过长 |
| 0x00 | 是 | 空 值 |
| 0x00 0x00 | 是 | 空字符串 |
| 0x00 0x41 | 是 | “A”,过长 |
| 0x00 0x00 0x00 | 是 | “\0” |
| 0x00 0x00 0x41 | 是 | “A”,过长 |

### 9.7.4 Unicode 字符串

一个Unicode字符串由一个包含UTF-8编码格式的字符串的字节向量来表示。其中,UTF-8在Unicode3.2 [UNICODE]标准中定义。如果一个Unicode字符串可空，则用一个可空的字节向量来表示。

### 9.7.5 字节向量

字节向量字段由一个无符号整数类型的长度前导及其之后的特定数量的原始字节来表示。数据部分的每一字节的有效数据位为8位。因此实际上其数据部分没有采用停止位编码。

可空的字节向量长度前导也为可空。空字节向量由一个空长度前导来表示。

### 9.7.6 二进制整数

二进制整数字段由一个无符号整数类型的长度前导及其之后的特定数量的字节来表示。其数据部分没有采用停止位编码，数据部分的每一字节的有效数据位为8位。

当二进制整数值的有效数据位少于等于19位时，有符号二进制整数的编码值为其补码，最高有效位为符号位，无符号二进制整数的编码值即为整数的二进制表示形式，长度前导的值等于实际所需的最小字节数。

当二进制整数值的有效数据位超过19位时，在从最高有效位开始将每19位编码到8个字节中。其中，只有包含最高有效位的部分可能为有符号数。如果含最低有效位的最后部分不足19位时，则其占用实际所需的最小字节数，长度前导的值则等于所有编码生成的字节数。。

可空的二进制整数长度前导也为可空。空二进制整数由一个空长度前导来表示。

布尔、枚举、集合，以及位组的编码如前文所述。

### 9.8 差值

#### 9.8.1 整数差值

差值由一个带符号整数来表示。可空的整数差值由一个可空的带符号整数来表示。

#### 9.8.2 比例数差值

差值由两个带符号整数来表示。第一个整数为指数的差值，第二个整数为尾数的差值。如果差值是可空的，则其具有一个可空的指数差值和一个不可空的尾数差值。一个空差值由一个空的指数差值来表示。当且仅当指数差值不为空的时，尾数差值在流中出现。

#### 9.8.3 ASCII 码字符串差值

差值由一个带符号的整数类型的减除长度，及其之后的ASCII字符串来表示。如果差值可空，则减除长度为可空。一个空差值用一个空减除长度来表示。当且仅当减除长度不为空时，流中存在字符串部分。

#### 9.8.4 字节向量差值

差值由一个带符号整数类型的减除长度及其之后的字节向量来表示。如果差值可空，则减除长度可空。当且仅当减除长度不为空时，流中存在字节向量部分。

附录C给出了本规范的编/解码规则表。

## 10 会话机制

### 10.1 基本要求

会话机制工作在传输层之上，使用Hello、Reset和Alert三个会话消息用来对编码消息的传输进行控制。会话发起方通过发送Hello消息来发起会话。Hello消息中包含了发送者的名称和协议的版本。对于每个不可分割的消息序列的传输过程，通常会以一个Reset消息，或任一个具有重置属性的模板的消息作为开始。在收到Reset消息的时候，解码器字典的前值将会被重置。

会话可以以显式或者异常的两种方式结束。通常情况下，通过发送一个Alert消息来显式地结束一次会话，以其他方式结束的情况的均为异常方式的结束。

### 10.2 Hello 消息

Hello消息用于在发起会话的时候交换双方的名称和协议的版本。Hello消息的发送必须先于其他任何消息，并且只能被发送一次。Hello消息可被并行地发送，但会话接收方在进行响应前，需要等待会话发起方的hello消息。

Hello消息使用具有reset属性为“有效”的编码模板，解码器的字典的前值在收到该消息后将会被重置。Hello消息的格式如表2所示：

**表 2 Hello 消息格式**

| Hello消息 | | | | |
|---|---|---|---|---|
| **字段名(Field)** | **必选** | **应用类型** | **编码类型** | **说明** |
| SenderName | 是 | String(16) | String | 发送方名称 |
| VendorId | 否 | Length(128) | String | 软件开发商的标识符，应为一个URL |

## 10.3 Alert 消息

Alert消息的格式如表3所示，用于在结束一个会话前向对方发送指示终止的理由。

**表 3 Alert 消息格式**

| Alert消息 | | | | |
|---|---|---|---|---|
| **字段名(Field)** | **必要** | **应用类型** | **编码类型** | **说明** |
| Severity | Y | enum | uInt32 | 警示消息的严重程度，字段值可以为 Fatal, Error, Warn, Info 中的一个。用于诊断，可归类后发送给应用层处理。 |
| Code | Y | enum | uInt32 | 警示代码。字段值可为 Close, Unauthorized, UnknownTemplateId, UnknownTemplateName, Other 中的一个，具体说明参见下表。 |
| Value | N | uInt(4) | uInt32 | 该字段用于传递一个数值，具体的使用根据 Code 的取值而不同，具体参见下表的相关说明。 |
| Description | N | String(256) | String | 相关信息的文字性说明 |

表4描述了警示代码的具体含义。如果表示即将终止会话的代码，则带有该代码的警示消息后将不会有其他消息被发送。

**表 4 Alert 消息警示代码说明**

| 警示代码 | **说明** | **是否结束会话** |
|---|---|---|
| Close | 正常结束 | 是 |

| | | |
|---|---|---|
| Unauthorized | 未经授权 | 是 |
| UnknownTemplateId | 模板 ID 无法找到匹配的模板名称。未知的模板 ID 由 Value 字段包含。 | 是 |
| UnknownTemplateName | 解码器无法找到能与一个模板名称对应的模板定义。如果可以的话，模板 ID 应该在 Value 字段中提供。未知模板的名称由 Description 字段包含。 | 是 |
| Other | | |

### 10.4 Reset 消息

Reset消息采用空消息体，不含任何字段。Reset消息由一个带有开启的reset属性的模板来编码，在收到该消息的时候，解码的字典的前值将会被重置。

对于编码后的一个二进制会话消息的传输结构如图2所示。其中PMAP和模板ID部分必需存在，但如前所述，Reset消息无消息体部分。


**图 2 编码后的会话消息格式**

附录B给出了会话消息的模板。

附录E给出了本规范的协议层次参考模型。

# 附 录 A
（资料性附录）
## RELAX NG 模板 XML 模式描述

```
default Namespace td = "http://www.csisc.cn/ns/DEEP/td/1.1"
Namespace local = ""    start = templates | template
templates = element templates { nsAttr?, templateNsAttr?, dictionaryAttr?, template* }
template = element template { templateNsName, nsAttr?, dictionaryAttr?, typeRef?, instruction* }
typeRef = element typeRef { nameAttr, nsAttr? }
instruction = field | templateRef
fieldInstrContent =  nsName, presenceAttr?, fieldOp?
field = integerField | decimalField | asciiStringField | unicodeStringField | byteVectorField |
booleanField | enumField | setField | sequence | group | bitGroup | binIntegerField
integerField =
element int32 { fieldInstrContent }
| element uInt32 { fieldInstrContent }
| element int64 { fieldInstrContent }
| element uInt64 { fieldInstrContent }
decimalField = element decimal { nsName, presenceAttr?, ( fieldOp | decFieldOp ) }
decFieldOp = element exponent fieldOp?, element mantissa fieldOp?
asciiStringField = element string { fieldInstrContent, attribute charset { "ascii" }? }
unicodeStringField = element string { byteVectorLength?, fieldInstrContent, attribute charset
{ "unicode" } }
byteVectorField = element byteVector { byteVectorLength?, fieldInstrContent }
byteVectorLength = element length { nsName }
booleanField = element boolean { fieldInstrContent }
enumField = element enum { fieldInstrContent }
setField = element set { fieldInstrContent }
binIntegerField =
element binInt { binIntLength?, fieldInstrContent }
| element uBinInt { binIntLength?, fieldInstrContent }
binIntLength = element length { nsName }
sequence = element sequence { nsName, presenceAttr?, dictionaryAttr?, typeRef?, length?,
instruction* }
length = element length {nsName?, fieldOp? }
group = element group { nsName, presenceAttr?, dictionaryAttr?, typeRef?, instruction* }
bitGroup = element bitGroup { nsName, presenceAttr?, dictionaryAttr?, typeRef?, instruction* }
fieldOp = constant | \default | copy | increment | delta | tail
constant = element constant { initialValueAttr }
\default = element default { initialValueAttr? }
copy = element copy { opContext }
increment = element increment { opContext }
delta = element delta { opContext }
tail = element tail { opContext }
initialValueAttr = attribute value { text }
opContext = dictionaryAttr?, nsKey?, initialValueAttr?
dictionaryAttr = attribute dictionary { "template" | "type" | "global" | string }
```

```
nsKey = keyAttr, nsAttr?
keyAttr = attribute key { token }
templateRef = element templateRef { ( nameAttr, templateNsAttr? )? }
presenceAttr = attribute presence { "mandatory" | "optional" }
nsName = nameAttr, nsAttr?, idAttr?
templateNsName = nameAttr, templateNsAttr?, idAttr?
nameAttr = attribute name { token }
nsAttr = attribute ns { text }
templateNsAttr = attribute templateNs { text }
idAttr = attribute id { token }
```

# 附 录 B
（资料性附录）
会话消息模板

```
<template name="Hello" scp:reset="yes" id="16002">
<typeRef name="Hello"/>
<string name="SenderName"/>
<string name="VendorId" presence="optional"/>
</template>
<template name="Reset" scp:reset="yes" id="120">
<typeRef name="Reset"/>
</template>
<template name="Alert" id="16003">
   <typeRef name="Alert"/>
   <uInt32 name="Severity"/>
   <uInt32 name="Code"/>
   <uInt32 name="Value" presence="optional"/>
   <string name="Description" presence="optional"/>
 </template>
```

# 附 录 C
（资料性附录）
协议编/解码规则表

| 运算符 | 存在属性 | PMAP 槽值 | 流中传输 | 空值 | 应用值 | 前值变化 | 下一状态 |
|---|---|---|---|---|---|---|---|
| 无 | 必要 | 无 | 值 | 不可 | 值 | 无 | 无 |
| | 可选 | 无 | 可空的值 | 空值 | 无 | 无 | 无 |
| | | | | 值 | 值 | 无 | 无 |
| 常量 | 必要 | 无 | 无 | 不可 | 常量 | 无 | 无 |
| | 可选 | 0 | 无 | 不可 | 无 | 无 | 无 |
| | | 1 | 无 | 不可 | 常量 | 无 | 无 |
| 缺省 | 必要 | 0 | 无 | 不可 | 缺省 | 不变 | 无 |
| | | 1 | 值 | 不可 | 值 | 不变 | 无 |
| | 可选 | 0 | 无 | 不可 | 缺省 | 不变 | 无 |
| | | 1 | 可空的值 | 空值 | 无 | 不变 | 无 |
| | | | | 值 | 值 | 不变 | 无 |
| 拷贝 | 必要 | 0 | 无 | 不可 | 前值/初值 | 不变/初值 | 已定义 |
| | | 1 | 值 | 不可 | 值 | 值 | 已定义 |
| | 可选 | 0 | 无 | 不可 | 前值/初值 | 不变/初值 | 已定义 |
| | | 1 | 可空的值 | 空值 | 无 | 空 | 空 |
| | | | | 值 | 值 | 值 | 已定义 |
| 差值 | 必要 | 无 | 差值 | 不可 | 前值/初值/缺省+差值 | 前值/初值/缺省+差值 | 已定义 |
| | 可选 | 无 | 可空的值 | 空值 | 无 | 不变 | 不变且不空 |
| | | | | 差值 | 前值/初值/缺省基值+差值 | 前值/初值/缺省基值+差值 | 已定义 |
| 递增 | 必要 | 0 | 无 | 不可 | 字典+1/初值 | 字典+1/初值 | 已定义 |
| | | 1 | 值 | 不可 | 值 | 值 | 已定义 |
| | 可选 | 0 | 无 | 不可 | 字典+1/初值 | 字典+1/初值/无 | 已定义 |
| | | 1 | 可空的值 | 空值 | 无 | 空 | 空 |
| | | | | 值 | 值 | 值 | 已定义 |
| 尾运算 | 必要 | 0 | 无 | 不可 | 前值/初值 | 前值/初值 | 已定义 |
| | | 1 | 尾值 | 不可 | 前值/初值/缺省基值*尾值 | 前值/初值/缺省基值*尾值 | 已定义 |
| | 可选 | 0 | 无 | 不可 | 前值/初值 | 前值/初值 | 已定义 |
| | | 1 | 可空的尾值 | 空 | 无 | 空 | 空 |
| | | | | 尾值 | （前值/初值/缺省基值）*尾值 | （前值/初值/缺省基值）*尾值 | 已定义 |

# 附 录 D
（资料性附录）
模板定义实例

```
<?xml version="1.0" encoding="utf-8"?>
<templates xmlns="http://www.csisc.cn/ns/DEEP/td/1.1"
templateNs="http://www.sse.com.cn/ns/templates/NGTS"
ns="http://www.sse.com.cn/ns/STEP1.0.0">
  <!--Incremental-->
  <template id="4003" name="MDIncrementalRefresh_4003">
    <!--incremental-->
    <typeRef name="MDIncrementalRefresh" />
    <string id="8" name="BeginString">
      <constant value="STEP.1.0.0" />
    </string>
    <uInt32 id="9" name="BodyLength">
      <delta />
    </uInt32>
    <string id="35" name="MsgType">
      <constant value="X" />
    </string>
    <string id="49" name="SenderCompID">
      <constant value="XSHG" />
    </string>
    <string id="56" name="TargetCompID">
      <constant value="EZView" />
    </string>
    <uInt64 id="34" name="MsgSeqNum" >
      <delta />
    </uInt64>
    <bitGroup>
    <boolean id="43" name="PossDupFlag" presence="optional">
      <default value="false" />
    </boolean>
    <boolean id="97" name="PossResend " presence="optional">
      <default value="false" />
    </boolean>
    <uInt3 id="339" name="TradSesMode">
      <copy />
    </uInt3>
    </bitGroup>
    <string id="52" name="SendingTime">
      <tail/>
    </string>
    <string id="347" name="MessageEncoding">
      <default value="GBK"/>
    </string>
```

```
<string id="1180" name="ApplID" presence="optional">
  <default />
 </string>
 <uInt64 id="1181" name="ApplSeqNum" presence="optional">
  <default />
 </uInt64>
 <sequence name="MDIncGrp">
  <length id="268" name="NoMDEntries">
   <delta />
  </length>
  <bitGroup>
   <enum id="279"   name=" MDUpdateAction ">
              <element name="New"/>
              <element name="Modify"/>
              <element name="Delete"/>
              <copy/>
              </enum>
              <enum id="285" presence="optional" name=" DeleteReason ">
              <element name="Cancel"/>
              <element name="Error"/>
              <copy/>
              </enum>
  </bitGroup>
   <string id="75" name="TradeDate" presence="optional">
  <default/>
  </string>
        <string id="55" name="Symbol" charset="unicode">
   <copy />
  </string>
  <string id="48" name="SecurityID">
   <tail />
  </string>
  <string id="22" name="SecurityIDSource">
   <copy value="101" />
  </string>
  <string id="167" name="SecurityType" presence="optional">
   <copy value="CS" />
  </string>
  <string id="762" name="SecuritySubType" presence="optional">
   <copy value="A"/>
  </string>
  <string id="461" name="CFICode" presence="optional">
   <copy value="ES"/>
  </string>
  <uInt64 id="8503" name="NumTrades" presence="optional">
   <delta/>
  </uInt64>
  <uInt64 id="1020" name="TradeVolume" presence="optional">
   <delta/>
  </uInt64>
  <decimal id="8504" name="TotalValueTraded" presence="optional">
```

```
            <delta/>
          </decimal>
          <decimal id="140" name="PreClosePx" presence="optional">
            <delta/>
          </decimal>
          <string id="269" name="MDEntryType" presence="optional">
            <copy />
          </string>
        <decimal id="270" name="MDEntryPx" presence="optional">
            <delta />
          </decimal>
          <uInt64 id="271" name="MDEntrySize" presence="optional">
            <delta />
          </uInt64>
          <string id="272" name="MDEntryDate" presence="optional">
            <default />
          </string>
          <string id="273" name="MDEntryTime" presence="optional">
            <default />
          </string>
          <uInt64 id="346" name="NumberOfOrders" presence="optional">
            <delta />
          </uInt64>
          <uInt32 id="290" name="MDEntryPositionNo" presence="optional">
            <increment value="1" />
          </uInt32>
                <string id="275" name="MDMkt" presence="optional">
            <copy value="XSHG"/>
          </string>
                <decimal id="8505" name="LastPriceChange" presence="optional">
            <delta/>
          </decimal>
          <decimal id="8506" name="TotalLongPosition" presence="optional">
            <default/>
          </decimal>
          <decimal id="8524" name="PERatio1" presence="optional">
            <default/>
          </decimal>
          <string id="8538" name="TradingPhaseCode" presence="optional">
            <copy value="TRADE"/>
          </string>
        </sequence>
        <uInt32 id="10" name="CheckSum"></uInt32>
      </template>
    </templates>
```

# 附 录 E
（资料性附录）
## 协议层次参考模型