FAST℠规范

版本 1.x.1

2006-12-20

文档状态

本文为FIX社区标准协议的规范，欢迎为作完善的一切讨论及建议。

传播

本文的传播不受任何限制。

版权声明

摘要

本文对FAST协议进行了详细规定。FAST是一种面向消息数据流的、具有高的压缩率和处理效率的编码方法。本文定义了称为“模版”的控制结构的语义，及其二进制的表示格式。同时还定义了一种用于具体模版定义结构的XML（扩展标记语言）语法。

免责声明

致谢

FPL感谢为本规范的制定作出努力的人们。本规范的基本设计思想主要来自于Rolf Andersson, Daniel May，Mike Kreutzjans，以及相关的讨论。David Rosenberg 起草了本规范。 Matt Simpson, Richard Shriver, Jim Northey为规范的后续草案作出大量贡献和反馈。Anders Furuhed, Yuriy Gormakh, Sitaram Guruswamy, Wei Keok, Göran Forsström, 及 Mats Ljungqvist审阅了规范的后续草案。范例部分由Matt Simpson 及 Greg Orsini完成。市场数据优化工作组（MDOWG）所有参与成员的反馈对本规范最终的顺利完成也起到重要的帮助作用。

中文译本

本中文规范由上海证券交易所的徐广斌翻译。

评审

FIX Protocol, Ltd., would like to acknowledge and thank Qingjun Wei of Teraspaces, Inc., an FPL member firm, for reviewing the Chinese translation of the FAST 1.1 Specification contributed by Guangbin Xu.

FPL感谢Teraspaces Inc.的魏庆军先生，对徐广斌先生的FAST 1.1规范的中文翻译所做的评审工作。Teraspaces公司是FPL的会员。

关于本中文译本

本FAST 1.1规范的中译本系英文原本的翻译。如果在本简体中文译本和英文原本之间存在任何差异，应以英文原本的陈述为准。英文原本可以通过以下链接获取：

http://www.fixprotocol.org/documents/3066/FAST%20Specification%201%20x%201.pdf

About This Translation

This translation of the FAST Specification 1.1 is a non-normative translation from the original English version. The original English version has priority and takes precedence should there be any differences between the Simple Chinese version and the English verison. The English version can be found here:

http://www.fixprotocol.org/documents/3066/FAST%20Specification%201%20x%201.pdf

目录

文档历史

版本	日期	作者	说明
1.x.01	2006-05-07	David Rosenborg	在模版定义的基础上扩展了传送编码(TE)，发展为一个统一的规范。传输编码部分增加了EBNF语法。加入了块编码。为减少名称使用冲突，将模版例子中的分组改名为段。对停止位编码实体的概念进行了形式化。
1.x.02	2006-05-09	David Rosenborg	草案的初稿发布
1.x.07	2006-08-17	David Rosenberg	FAST 规范 1.1 候选版本发布，并请评议
1.x.09	2006-12-05	David Rosenborg	少量更正和添加，加入范例
1.x.1	2006-12-20		市场数据优化工作组(MDOWG)批准最终版本。

1 导言

本文定义了FAST的结构和语法。FAST是一种面向消息数据流的二进制编码方法。 FAST是FIX Adapted for Streaming.的缩写（适流FIX）。虽然FAST提出的初衷是为了对FIX消息进行优化，但本文中定义的编码方式已经被推广应用到更广泛的协议集。

FAST编码方法在两个层面上降低数据流的大小。首先，通过“字段操作符”的概念使得可以利用流中数据的相关性，消除冗余数据。其次，在二进制编码对余下数据的串行化中利用了可自描述的字段长度以及指示字段是否存在的位图。

编码依据称为“模版”的控制结构来进行。模版通过规定字段的顺序和结构、字段操作符，及其使用的二进制编码表示方法来控制对流的一部分的编码。

本规范对模版定义(Template Definition, TD)结构的具体语法(concrete syntax)进行了定义。具体语法的引入是为了能提供一个人机可读、正规化、和信息全保真的格式，它被作为默认格式，用来制作、存储和交换FAST模版。

但是，本规范的具体语法不是为了用于在FAST会话的两端间在线交换模版定义结构。对于在线传送，FAST会话控制协议[SCP]提供了对本文所定义的模版结构的FAST串行化方法。

本文使用规划(schema)语言将具体语法正式定义为一个XML的结构。

处理器（processor）（编码器或解码器）并不要求必须使用上述的具体语法。例如，处理器可以使用SCP协议读取编码为FAST消息的模版定义，或者也可将它们硬编码在程序中。

处理器通常需要对一个模版集合进行管理。尽管具体语法提供了在一个XML文档中定义单个或多个模版的方法，但本规范并不对一般情况下如何构建或维护模版集合或模版数据库作出规定。对于特定处理器所使用的特定模版，甚至可以同时使用多个源（譬如XML文档及SCP）来进行定义。

下面的XML片段是采用具体语法格式的一个模版定义的例子。

```
<templates xmlns="http://www.FIXprotocol.org/ns/template-definition"
templateNs="http://www.FIXprotocol.org/ns/templates/sample"
ns="http://www.FIXprotocol.org/ns/FIX">
<template name="MDRefreshSample">
<typeRef name="MarketDataIncrementalRefresh"/>
<string name="BeginString" id="8"> <constant value="FIX4.4"/> </string>
<string name="MessageType" id="35"> <constant value="X"/> </string>
<string name="SenderCompID" id="49"> <copy/> </string>
<uInt32 name="MsgSeqNum" id="34"> <increment/> </uInt32>
<sequence name="MDEntries">
<length name="NoMDEntries" id="268"/>
<uInt32 name="MDUpdateAction" id="279"> <copy/> </uInt32>
<string name="MDEntryType" id="269"> <copy/> </string>
<string name="Symbol" id="55"> <copy/> </string>
<string name="SecurityType" id="167"> <copy/> </string>
<decimal name="MDEntryPx" id="270"> <delta/> </decimal>
<decimal name="MDEntrySize" id="271"> <delta/> </decimal>
<int32 name="NumberOfOrders" id="346"> <delta/> </int32>
<string name="QuoteCondition" id="276"> <copy/> </string>
<string name="TradeCondition" id="277"> <copy/> </string>
</sequence>
</template>
</templates>
```

附录3包含更多的具体语法及其相应字段的编码的范例。

2 术语

术语编码(encode)是指将一个应用类型的实例串行化(serialize)为FAST流的过程。

术语解码(decode)是指将FAST流的一部分的反串行化(deserialize)为一个应用类型实例的过程。鉴于在对编码操进行定义后，解码已较简单，因此，本文并不一定对一个解码操作再作明确地描述。

本文根据处理模型的内容分类来定义FAST流的编解码。但该处理模型被看作是一个抽象的模型，只要能获得与使用本模型一样的结果，可以采用任意的方法来具体实现FAST编解码。

3 标记法

本文使用RELAX NG规划语言[RNC]的紧凑语法来正式定义用于模板定义的XML结构，在规划的片段中穿插了相应的描述性文字。附录1给出了完整的、可扩展的规划。文字中的规划片段并没有包含与可扩展性相关的部分。附录2给出了W3C XML 规划[XSD]的对应版本。

错误由带方框的错误标识符来标记。

参考文献由带方框的、“参考文献”小节中的对应标识符来标记。

3.1 XML命名空间 （namespace）

模板定义结构命名空间(namespace)的URI为 “http://www.FIXprotocol.org/ns/FAST/td/1.1”. 本文中使用前缀td:对该命名空间中的元素进行引用.

```
default Namespace = "http://www.FIXprotocol.org/ns/FAST/td/1.1"
```

4 错误处理

对模板定义进行检验所发现的错误称为静态错误(static error)。编码器和解码器必须对静态错误进行捕获，并将发生该类错误的模板丢弃。

当从XML文档中读取模板定义时，若该文档出现以下情况则为一个静态错误[ERR S1]：

- 不是正确的XML定义格式，
- 不符合XML命名空间 [XMLNS]中的约束，
- 不符合附录1中规定的规划。

对FAST流进行编码或解码时检测到的错误分为两种：一种为动态错误(dynamic error)，另一种为可报告错误(reportable error)。编码器和解码器必须对动态错误进行捕获。虽然对可报告错误的捕获是可选的，但也建议对可报告错误进行捕获。通常来说，不对可报告错误进行捕获的一个原因，是希望能获得更好的性能。但是，为了保证互操作性，建议在开发及对实现的测试阶段对所有的错误都进行捕获。

5 应用类型

应用类型(application type)表示使用FAST协议的应用层的分组或消息的类型。本文并不规定应用类型的结构，并将其看成是抽象的实体，但是认为它们能被映射为以下的模型:

- 分组(group)是包含一个无序的字段的集合的命名类型。
- 字段(field)具有名称和类型。其名称在一个分组内唯一。其类型可以为基本类型、序列类型、

或者分组类型中的一种。

- 序列(sequence)包括长度及一个元素的有序集合。其中，每一元素均为分组类型。本规范并不要求其所有元素的分组类型均相同。这可能会造成在应用层面上异构的序列。但在FAST的一个特定应用中可以就此进行约束，要求所有的元素都具有相同的类型。

- 基本类型(primitive type)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), 十进制数和字节向量，其值域分别与本文中定义的相对应的类型相同。

- 流内最外层的分组也被称为消息(message)。

6 模版

模板(template)规定了如何对应用类型的一个实例（也即字节流的一部分）进行编码。模板用名称来进行标识，其中名称用于对另一模板或外部上下文所定义的模板进行引用。

模板本身不构成类型，但通过引用与应用类型关联。一个应用类型可对应一个或多个模板[1]。也可创建一个可用于一个或多个应用类型的模板。

在具体语法中，模板由“<td:template>”元素定义。一个模板定义的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>元素中，该元素可包含应用于整个封装的模板集合的命名空间参数。

模板包含一个指令(instruction)的序列。其中，指令的顺序很重要，应与数据在流中的位置相对应。指令分为以下两大类：字段指令(field instruction)和模板引用指令(template reference instruction)。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。

```
start = templates | template
templates = element templates { nsAttr?, templateNsAttr?, dictionaryAttr?, template* }
template = element template { templateNsName, nsAttr?, dictionaryAttr?, typeRef?, instruction* }
instruction = field | templateRef
```

6.1 指令上下文

编解码在指令的上下文(Instruction Context)中进行，其中包括：

- 模板的集合
- 当前模板(current template)
- 应用类型的集合
- 当前应用类型(current application type)
- 字典(dictionary)的集合
- 可选的初值(initial value)

当前应用类型在初始时为特殊的类型—“任意(any)”。当处理器碰到包含“<td:typeRef>”的元素时当前类型改变。新类型适用于该元素所包含的指令。“<td:typeRef>”可在“<td:template>”、“<td:group>” 和“<td:sequence>” 元素中出现。

```
typeRef = element typeRef { nameAttr, nsAttr?}
```

当前模板指正被处理的模板，在流中遇到一个模板标识符时当前模版被更新。如“模板引用指令”

[1] 让同一应用类型对应多个模板，使得可针对该类型的不同使用情况进行优化。例如，许多 FIX 消息都包含了大量各类的字段，但只有少数是被组合使用的。与使用一个带有大量可选成分的模板相比，对每一主要的字段的组合选择使用不同的模板是一种更为简炼的方法。

小节中所述的，一个静态模板引用也可改变当前模板。

字典集合和初值在后文的“操作符”小节中进行描述。

6.2 字段指令

每一字段指令(Field Instruction)均具有名称和类型。名称(name)标识了当前应用类型(application type)中对应的字段。类型(type)则规定了字段的基本编码方法。如果指令的类型不能转换为相应的应用字段的类型，或解码时不能由相应的应用字段的类型转换而得到，则产生动态错误[ERR D1] 。

可选的“存在(presence)”参数指示字段是一个必要(mandatory)字段，还是一个可选(optional)字段。如果该参数未被指定，则字段为必要的。

基本字段(primitive field)，也即除了分组或序列之外的字段，可具有一个字段操作符(field operator)，该操作符规定了对字段的某个优化操作。

```
field = integerField | decimalField | asciiStringField | unicodeStringField | byteVectorField | sequence | group
fieldInstrContent = nsName, presenceAttr?, fieldOp?
presenceAttr = attribute presence { "mandatory" | "optional" }
```

6.2.1 整数字段指令

整数(Integer)在传送编码中的大小不受限。但在通常情况下，应用层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元大小。值的编码和解码方法不受整数大小的影响。

前缀“int”表示字段是带符号的，“uInt”则表示字段是无符号的。

```
integerField =
element int32 { fieldInstrContent }
| element uInt32 { fieldInstrContent }
| element int64 { fieldInstrContent }
| element uInt64 { fieldInstrContent }
```

如果流中的整数大于特定类型的最大值，或小于指定类型的最小值，则动态错误[ERR D2]发生。下表列出了各整数类型的最大和最小值。

类型	最小值	最大值
int32	-2147483648	2147483647
uInt32	0	4294967295
int64	-9223372036854775808	9223372036854775807
uInt64	0	18446744073709551615

6.2.2 十进制数字段指令

十进制数字段指令(Decimal Field Instruction)由两部分表示：指数(exponent)部分及尾数(mantissa)部分。该指令可包含用于整个十进制数的单个字段操作符，或两部分各自使用的两个操作符。如果为尾数和/或指数单独指定了操作符，则在结合成十进制数型数之前，操作符被单独应用到每一部分。指数和尾数字段操作符的操作数为带符号整数，分别为int32和int64类型。如果没有为整个十进制数指定操作符，或只指定了单个操作符，则操作数为一个十进制数，在传送编码中由一个带比例数(Scaled Number)表示。

虽然指数被看作int32类型，但其允许的值域范围是[-63,63]。在应用任何操作符后，如果获得的指数超出了该范围，则一个可报告错误[ERR R1]产生。

当单独应用操作符时，为指数和尾数部分生成的名称是该十进制数字段的名称所特有的，它们将被用作相应操作符的关键字的缺省值。

如果十进制数字段是可选存在的，并具有单独的操作符，则尾数存在与否取决于指数是否存在。具体规定请参见“存在图和空值使用”小节。

当使用单独的操作符时，可以对十进制数的范围和精度进行限制。例如，若用于指数的常量操作符采用常量值2，则就不能将为0.01编码到该字段。如果由于使用某个操作符引入了限制，造成值不能被编码到该字段，则动态错误[ERR D3]发生。

为十进制数字段指令指定的初值将被正规化。这将在下文的“初值”小节中进行描述。

```
decimalField = element decimal { nsName, presenceAttr?, ( fieldOp | decFieldOp ) }
decFieldOp = element exponent { fieldOp }?, element mantissa { fieldOp }?
```

6.2.3 字符串字段指令

字符串字段指令(String Field Instruction)具有指示字符串所使用字符集的、可选的一个“字符集（charset）”参数。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，传送编码中字符串可由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，以将基础字节向量的长度前导(length preamble)和一个名称关联。

```
asciiStringField = element string { fieldInstrContent, attribute charset { "ascii" }? }
unicodeStringField = element string { byteVectorLength?, fieldInstrContent, attribute charset { "unicode" } }
```

6.2.4 字节向量字段指令

字节向量字段指令(Byte Vector Field Instruction)表示字段在传送编码中由一个字节向量表示。

在具体语法中，可以通过指定“ <td:length>”元素将字节向量的长度前导和某个名称关联。逻辑上，该字段是uInt32类型的。“ <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句柄。

```
byteVectorField = element byteVector { byteVectorLength?, fieldInstrContent }
byteVectorLength = element length { nsName }
```

6.2.5 序列字段指令

序列字段指令(Sequence Field Instruction)指示应用类型的字段为序列类型，应该对其包括的指令分组重复使用以对其每一元素进行编码。如果该分组中的任一指令需要在存在图中占用一个位元，则在传送编码中每一元素由一个段(segment)表示。

序列具有一个关联的长度字段，其包含一个无符号整数，用以表示编码元素的个数。若流中出现长度字段，则必须紧接在编码元素的前面出现。该长度字段具有名称，类型为uInt32，且可具有字段操作符。其命名方式有以下两种：

- 隐式方式(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。
- 显式方式(explicit)：名称在模板定义中显式地指定。

序列可为必要(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。

在具体语法中，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。该元素规定了长度字段的属性。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，且无字段操作符。

```
sequence = element sequence { nsName, presenceAttr?, dictionaryAttr?, typeRef?, length?, instruction* }
length = element length { nsName?, fieldOp? }
```

6.2.6 分组字段指令

分组字段指令(Group Field Instruction)将一个指令的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图(Presence Map, 或PMAP)中占用一个位元，则在传送编码中该分组由一个段表示。

当前应用类型并不要求具有与分组所对应的概念。也就是说，对分组进行解码所得到的字段可以在应用类型中展开为一层。

分组字段指令的主要用途是使得可以用存在图中的单个位元来指示一整组的字段存在与否。

```
group = element group { nsName, presenceAttr?, dictionaryAttr?, typeRef?, instruction* }
```

6.3 字段操作符

字段操作符(field operator)规定了字段编码的优化方式。不是任何操作符都可任意地应用到不同的字段类型。虽然相应的约束并不都在规划中明确表示，但在对每一操作符的描述文字中都做了明确的阐述。如果为操作符指定了不适用的字段类型，则为一个静态错误[ERR S2]。

```
fieldOp = constant | \default | copy | increment | delta | tail
```

6.3.1 字典与前值

一些操作符依赖于前值(previous value)。前值在带名的字典中维护。字典是一个条目集合。其中，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)，已指定(assigned)。在处理开始时，所有值的状态均为未定义。已指定状态表示前值存在，空状态表示前值不存在。空状态只适用于可选的字段。关于何时设置空状态，请参见“存在图和空值使用”小节的介绍。

操作符在字典中的前值，是与其关键字名称相同的条目的值。操作符的缺省关键字(key)是其字段的名称。显式的关键字可以用“关键字(key)”参数来指定。通过指定显式的关键字，不同名称的字段的操作符可以共享字典中的同一个前值条目，

如果操作符的字段的类型与所访问的条目值的类型不相同，则动态错误[ERR D4]发生。

字典的名称在字段操作符元素的“字典（dictionary）”参数中指定，或者在规划允许的在先的某处元素中指定。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:

- 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中操作符与模板T2中操作符共享相同的字典。
- 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中的操作符与应用类型A2的模板T2中的操作符共享相同的字典。
- 全局(global)字典：字典为全局的。无论模板和应用类型如何，所有操作符共享相同的字典。

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，两个操作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。关于如何通知一个解码器或编码器进行重置，本规范不作规定。但在编码器和解码器中，重置出现的顺序应当与流的内容的顺序相对应。

```
opContext = dictionaryAttr?, nsKey?, initialValueAttr?
dictionaryAttr = attribute dictionary { "template" | "type" | "global" | string }
nsKey = keyAttr, nsAttr?
keyAttr = attribute key { token }
```

6.3.2 初值

初值(Initial Value)为一个Unicode字符集的字符串，由操作符元素的“值(value)”参数指定。该值可通过后文中“字符串转换”小节中定义的方法转换为字段的类型。在对初值进行解译的时候，类型转换过程中可能发生的动态和可执行错误，均被认为是静态错误[ERR S3]。

如果字段为十进制数类型，类型转换得到的值将被正规化。这样做的原因是当对指数和尾数单独应用操作符时，它们必须是可预见的。十进制数值的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，则正规化十进制数的尾数为0，指数也为0.

```
initialValueAttr = attribute value { text }
```

6.3.3 常量操作符

常量操作符(Constant Operator)规定字段的值均相同。字段值即为初值。如果指令上下文无初值则静态错误[ERR S4]发生。

常量字段的值不被传送

常量操作符适用于所有字段类型。

```
constant = element constant { initialValueAttr }
```

6.3.4 缺省操作符

缺省操作符(Default Operator)规定字段的值在流中出现，否则为初值。除非字段是可选存在的，否则若指令上下文无初值的话，则静态错误[ERR S5]发生。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。

缺省操作符适用于所有字段类型。

```
\default = element default { initialValueAttr? }
```

6.3.5 拷贝操作符

拷贝操作符(Copy Operator)规定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：

- 已指定状态：则字段值为前值。
- 未定义状态：则字段值为初值，并成为新的前值。如果指令上下文中无初值的话，除非字段不是可选存在的，否则动态错误[ERR D5]发生。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。
- 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。如果字段是必要的，则动态错误[ERR D6]发生。

拷贝操作符适用于所有的字段类型。

```
copy = element copy { opContext }
```

6.3.6 递增操作符

递增操作符(Increment Operator)规定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：

- 已指定状态：则字段值为前值加1。计算得到的值成为新的前值。
- 未定义状态：则字段值为初值，并且成为新的前值。除非字段是可选存在的，否则如果指令上下文没有初值，则动态错误[ERR D5]发生。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。
- 空状态：则字段值为空。如果字段是可选的，则值被视作不存在。如果字段是必要的，则动态错误[ERR D6]发生。

递增操作符适用于整数字段类型。

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。

```
increment = element increment { opContext }
```

6.3.7 差值操作符

差值操作符(Delta Operator)规定流中存在差值。如果字段是可选存在的，则差值可为NULL。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。

```
delta = element delta { opContext }
```

根据前值的状态，基值按照以下方式来确定：

- 已指定状态：则基值为前值。
- 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。
- 空状态：如果前值为空则动态错误[ERR D6]发生。

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。

6.3.7.1 整数的差值

差值在传送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.

如果结合的值大于特定整数类型的最大值，或小于特定整数类型的最小值，则可报告错误[ERR R4]发生。

备注：差值所需的整数大小可能大于字段类型的特定大小。例如，如果无符号32位整数的基值为4294967295，新值为17，则需要带符号64位整数来表示差值-4294967278。但是，这并不影响差值在流中的表示方式。

6.3.7.2 十进制数的差值

差值在传送编码中以一个带比例数字差值表示。通过将差值和基值的指数和位数部分分别对应相加来计算结合的值。

如果结合后的指数小于-63或大于63，或者结合后的尾数超过64位整数的表示范围，则可报告错误[ERR R1]发生。

十进制数的缺省基值为0，缺省基值的指数为0.

备注：由于十进制数的差值操作符包括单独的指数和尾数差值，因此必须按照以下方式实现十进制数的前值的保存，即

必须保留指数和尾数部分的这种格式安排，或者使之可在处理下一个字段时候被重新构建。

6.3.7.3 ASCII码字符串的差值

差值在传送编码中以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(negative)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。

如果减除的长度大于基值中的字符数量，或者超出了32位带符号整数的范围，则动态错误[ERR D7]发生。

6.3.7.4 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。结合值如果不是一个合法的UTF-8 序列的话，则可报告错误[ERR R2]发生。

6.3.7.5 字节向量的差值

差值在传送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。

如果移除长度大于基值的字符数量，或者超出了int32的表示范围，则动态错误[ERR D7]发生。

6.3.8 接尾操作符

接尾操作符(Tail Operator)规定在流中可选地存在一个尾值(tail value)。

如果字段为可选存在的，则尾值可为NULL。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：

- 已指定状态：则基值为前值。
- 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。
- 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：

- 已指定状态：则字段值为前值。

- 未定义状态：则字段值为前值，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，则动态错误[ERR D6]发生。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。

- 空状态：则字段值为空。如果字段是可选地，则值被视为不存在，如果字段是必要的，则动态错误[ERR D7]发生。

在具体语法中，接尾操作符以“<td:tail>”元素表示。

```
tail = element tail { opContext }
```

以下的小节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。接尾操作符只能应用于这些类型上。

6.3.8.1 ASCII字符串的接尾

尾值在传送编码中以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(empty string)。

6.3.8.2 Unicode字符串的接尾

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，则可报告错误[ERR R2]发生。

6.3.8.3 字节向量的接尾

尾值在传送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。

6.4 模版引用指令

模版引用指令(Template Reference Instruction)规定该模版的一部分由另一模版来定义。模版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版作为当前模版来继续处理。静态引用并不需要流中一定有存在图或模版ID。如果指定名称的模版不存在，则动态错误[ERR D8]发生。

动态引用规定了流中存在一个存在图和模版标识符(template identifier)，应该以该标识符所指示的模版作为当前模版继续进行处理，并在传送编码中以一个段来进行表示。如果与流中出现的模版标识符所关联的模版不存在，则动态错误[ERR D9]发生。

当处理进行到引用的静态或动态模版的终止处时，当前模版被恢复，并从引用指令后的执行点继续进行处理。

```
templateRef = element templateRef { ( nameAttr, templateNsAttr? )? }
```

7 名称

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

应用类型、字段和操作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。

模版的命名空间URI由“templateNs”参数指定，其继承方式与ns参数相同。为模版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，但是模版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。

```
nsName = nameAttr, nsAttr?, idAttr?
templateNsName = nameAttr, templateNsAttr?, idAttr?
nameAttr = attribute name { token }
nsAttr = attribute ns { text }
templateNsAttr = attribute templateNs { text }
idAttr = attribute id { token }
```

7.1 辅助标识符

任何一个可以带名称，且不是一个引用的构件，还可以带有一个辅助的标识符(auxiliary identifier)，该标识符由“标识符(id)”参数指定。本规范不对辅助标识符的使用语义做任何规定。也不对标识符的作用范围做规定。但是，对于一个特定的适用于FAST的通信协议，可以选择对辅助标识符的使用作出限制。

例如，当使用Fast传送FIX消息时，辅助标识符的一个典型应用是用来指定每一个字段的FIX标签号码。辅助标识符另一个可能的应用，是在双方通信不支持动态模版交换和标识符指定的时候，指定静态的模版标识符。

备注：本规范提供了带内指定辅助标识符的方法，但并不表示不能采用其他的带内（如通过外来元素或参数，参见“可扩展性”小节）或者带外的模式将辅助标识符映射为构件的名称。

8 类型转换

如果模版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。字节向量与其余类型的转换均为一个动态错误[ERR D10]。

8.1 字符串转换为其他类型

在以下的小节中，“空白切除(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).

如果一个字符串与下列小节中所规定的语法不匹配，则动态错误[ERR D11]发生。

备注：虽然带符号整数和十进制数的语法允许形如-0和-0.0的负零值，但此类值将被正规化为正数形式。负零值在FAST流中无法表示。

8.1.1 转换为整数

字符串被作为'0'—'9'数字的序列解译。如果为带符号类型，则允许以起始的减号来表示负数。文字已经过空白切除。如果结果数字超出了特定整数的大小，则可报告错误[ERR R4]发生。例如，如果类型为int8的话，则字符串"4711"将导致错误。

8.1.2 转换为十进制数

字符串具有一个整数部分和一个小数部分。允许对两者进行指定，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。文字已经过空白切除。如果转换结果的指数小于-63或者大于63，或者尾数超过int64的表示范围，则可报告错误[ERR R1]发生。

8.1.3 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。首先通过剥离所有的空白，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。

8.1.4 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，否则将是一个可报告错误[ERR R3]。

8.2 整数转换为其他类型

8.2.1 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。如果目标类型无法表示值的话，则可报告错误[ERR R4]发生。例如，一个负值不能转换为一个无符号类型。

8.2.2 转换为十进制数

如果可以由一个指数范围为[-63 , 63] 、尾数为int64的带比例数表示，则一个整数可以被转换为十进制数，否则可报告错误[ERR R1]发生。

8.2.3 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('-')。

8.3 十进制数转换为其他类型

8.3.1 转换为整数

当前仅当无小数部分时，一个十进制数可以转换为一个整数。也就是说，其值实际上为一个整数。如果其值实际上不为一个整数，则可报告错误 [ERR R5] 发生。

8.3.2 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.

8.4 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 可扩展性

使用“外来(foreign)”参数和元素，可以将特定于应用层的数据，添加到xml格式的模版定义结构中。规划中的任一元素可以带有外来参数和外来子元素。一个外来参数是一个带名参数，其名称的命名空间URI即非空字符串，也非模版的命名空间URI。一个外来元素是一个带名元素，其名称的命名空间URI与模版的命名空间URI不同。外来子元素可以相对其他子元素任意放置。外来参数和元素的内容不受限制。

在规划中使用“other”样式来实现可扩展性的部分，“other”样式可利用交错式的操作被放置到规划里的相关位置。

```
 other = foreignAttr*, foreignElm*
foreignElm = element * - td:* { any }
foreignAttr = attribute * - (local:* | td:*) { text }
any = attribute * { text }*, ( text | element * { any } )*
```

10 传送编码

下面的扩展巴科斯范式（EBNF）语法规定了FAST流的总体结构。其中，末端符号为斜体，起始符号为stream。

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

一个FAST流(stream)为一个消息的序列(sequence)，或为一个块的序列。本规范不对特定的流使用何种方式作出建议。因此，在编码数据的产生方和使用方之间需要就此达成协议。

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)。如果块的大小为0，则动态错误[ERR D12]发生。块大小由无符号整数表示，可能会“过长(overlong)”。“过长”属性在下文的“整型数”小节中定义。

每个消息(message)由一个段 (即消息段)来表示。

段(segment)带有一个头部，由一个存在图和紧跟在后的一个可选的模版标识符构成。如果一个段

为消息段，或者段作为动态模版引用指令的结果出现，则该段带一个模版标识符。模版标识符的编码与指定了拷贝操作符的情况相同。该拷贝操作符使用全局字典，且具有所有模版标识符字段公用的一个内部关键字。也就是说，带有模版标识符的段并不总是物理地包含模版的标识符。但是，段存在图的第1个位元由其拷贝操作符占用。

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

10.1 字节和位元顺序

所有的整数字段使用大尾端(big-endian)方式来表示，其中位元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。

10.2 停止位编码实体

Fast传送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个位元为有效数据位(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。实体值的位元数量总是7的倍数，其最小的长度为7位。

10.3 模版标识符

模版标识符在流中以一个无符号整数来表示。如果过长，则为一个可报告错误[ERR R6]。

如果一个解码器不能找到与流中出现的模版标识符所关联的模版，则动态错误[ERR D9]发生。

本规范不对如何将一个标识符映射为模版名称作出规定。一个特定的实现可以选择对模版标识符进行静态的分配。若是这种情况，可以利用具体语法中的辅助标识符来进行这种映射。而在其他实现中，则也可以选择使用例如会话控制协议[SCP]等方法来动态地分配模版标识符。

10.4 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NULL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NULL的表示方式。所有的可空类型按照以下方式构建，即NULL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，NULL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。

10.5 存在图

存在图(Presence Map PMAP)为位元的一个序列。存在图代表的段中的字段根据当前模版的规定来使用位元。

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

如果一个存在图超过7位，且以7位或7位以上的0结尾，则其过长。如果一个存在图过长，则一个可报告错误[ERR R7]发生。如果存在图包含的位元数大于指令使用所需的位元数，则可报告错误[ERR R8]发生。

10.5.1 存在图与空值的使用

存在图按照字段的条目的顺序分配位元。也就是说，在模版中先出现的指令比后出现的指令所分配的位元的顺序要高。位元在当前段的存在图中进行分配。

备注： 如6.2.5小节所规定的，序列的长度(length)字段的类型为uInt32。也就是说，适用于无符号整数字段的所有编码规则对序列的长度字段也适用。

如果一个字段为必要的、且具有常量操作符，则其在存在图中不占位。

具有常量操作符的可选字段占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。

如果字段无字段操作符且为必要的，则其在存在图中不占位，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，且无字段操作符，则其采用可空的表示方法进行编码，并且用NULL表示值不存在，其在存在图中不占位。

缺省、拷贝和递增操作符对存在图和空值的使用如下：

- 必要的整数、十进制数、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。
- 可选的整数、十进制数、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。NULL表示值不存在，前值的状态也将被设置为空，但是在使用缺省操作符时，前值的状态则不作更改.

差值操作符对存在图的使用如下：

- 必要的整数、十进制数、字符串和字节向量字段：不占位。
- 可选的整数、十进制数、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。NULL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。

接尾操作符对存在图的使用如下：

- 必要的字符串和字节向量字段：占1位、
- 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。NULL表示值不存在，且前值被设置为空。

具有单独操作符的十进制数字段的使用情况如下：

- 如果十进制数是必须存在的，则指数字段和尾数字段将作为两个独立的、必要的整数字段，按照前述的相关方式进行处理。
- 如果十进制数是可选存在的，则指数字段被视作可选的整数字段，尾数字段被视作必要的整数字段。尾数字段及其在存在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。如果尾数的操作符需要在存在图中占1位，则当且仅当指数值被视作存在，该位才可存在。

10.6 字段

10.6.1 整数

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)。如果在流中出现一个过长的整型数，除非该整数被用作块大小，否则可报告错误[ERR R6]发生。

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NULL由所有位为0的7位实体值表示。

备注： 可空整数的编码可能需要比该整数类型所规定的最大位元数量更多的位元。例如，最大的32位无符号整数4294967295的可空由4294967296来表示，其编码为0x10 0x00 0x00 0x00 0x80，在实体值中需要33个有效位。

10.6.1.1 有符号整数

实体的值为一个补码[TWOC]。实体值的最高有效位为符号位。

备注： 由于实体值的最高有效位是符号位，因此在某些情况下实体值的7位最高有效位必须都为0。例如，如果要被编码的值为64，其二进制表示为01000000，则其停止位编码为0x00 0xC0。如果我们不用另外7位0位元作为起始，则最高有效位（同时也是符号位）将为1，则该编码将错误地表示为-64。

10.6.1.2 无符号整数

实体值即为整数的二进制表示形式。

10.6.2 带比例数

带比例数(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，但为了支持十进制数的确切表示形式，带比例数采用了以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。

带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。

如果一个带比例数可空，则其指数可空，但尾数不可空。NULL值的带比例数由一个NULL的指数来表示。当且仅当指数不为NULL时，尾数在流中出现。

10.6.3 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。

位元序列若以7个0位元作为开始，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元组成。如果一个字符串在移除零前导后有剩余位元，并且剩余位元的前7个位元不全为0，则称该字符串过长(overlong)。如果流中出现一个过长的字符串，则发生一个可报告错误[ERR R9]。

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元，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附加零前导。跟随的位元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元，则其表示一个NULL串。

下列表格对零前导的使用进行了总结：

实体值	是否可空	说明
0x00		空字符串
0x00 0x00		“\0”
0x00 0x41		“A”,过长
0x00	是	NULL 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

10.6.4 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。

10.6.5 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，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。NULL字节向量由一个NULL长度前导来表示。

10.7 差值

10.7.1 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。

10.7.2 带比例数差值

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NULL差值由一个NULL的指数差值来表示。当且仅当指数差值不为NULL的时，尾数差值在流中出现。

10.7.3 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NULL差值用一个NULL减除长度来表示。当且仅当减除长度不为NULL时，流中存在字符串部分。

10.7.4 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NULL时，流中存在字节向量部分。

附录1 RELAX NG 规划

```
default Namespace td = "http://www.FIXprotocol.org/ns/FAST/td/1.1"
Namespace local = ""    start = templates | template

templates = element templates { ( nsAttr?, templateNsAttr?, dictionaryAttr?, template* ) & other }

template = element template { ( templateNsName, nsAttr?, dictionaryAttr?, typeRef?, instruction* ) & other }
typeRef = element typeRef { nameAttr, nsAttr?, other }

instruction = field | templateRef

fieldInstrContent = ( nsName, presenceAttr?, fieldOp? ) & other

field = integerField | decimalField | asciiStringField | unicodeStringField | byteVectorField | sequence | group

integerField =
element int32 { fieldInstrContent }
| element uInt32 { fieldInstrContent }
| element int64 { fieldInstrContent }
| element uInt64 { fieldInstrContent }

decimalField = element decimal { ( nsName, presenceAttr?, ( fieldOp | decFieldOp ) ) & other }

decFieldOp = element exponent { fieldOp & other }?, element mantissa { fieldOp & other }?

asciiStringField = element string { fieldInstrContent, attribute charset { "ascii" }? }
unicodeStringField = element string { byteVectorLength?, fieldInstrContent, attribute charset { "unicode" } }

byteVectorField = element byteVector { byteVectorLength?, fieldInstrContent }
byteVectorLength = element length { nsName }

sequence = element sequence { ( nsName, presenceAttr?, dictionaryAttr?, typeRef?, length?, instruction* ) &
other }

length = element length { ( nsName?, fieldOp? ) & other }

group = element group { ( nsName, presenceAttr?, dictionaryAttr?, typeRef?, instruction* ) & other }

fieldOp = constant | \default | copy | increment | delta | tail

constant = element constant { initialValueAttr & other }

\default = element default { initialValueAttr? & other }

copy = element copy { opContext }

increment = element increment { opContext }

delta = element delta { opContext }

tail = element tail { opContext }

initialValueAttr = attribute value { text }

opContext = ( dictionaryAttr?, nsKey?, initialValueAttr? ) & other

dictionaryAttr = attribute dictionary { "template" | "type" | "global" | string }

nsKey = keyAttr, nsAttr?

keyAttr = attribute key { token }

templateRef = element templateRef { ( nameAttr, templateNsAttr? )?, other }

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 }

other = foreignAttr*, foreignElm*

foreignElm = element * - td:* { any }

foreignAttr = attribute * - (local:* | td:*) { text }

any = attribute * { text }*, ( text | element * { any } )*
```

附录2　W3C XML规划（非规范化）

本规划由规范化的RELAX NG规划自动翻译而来，与正本只是近似。因为RELAX NG比XML规划具有更强的表示性，所以该规划比正本的容许性更强。

```
<xs:schema xmlns:td="http://www.fixprotocol.org/ns/fast/td/1.1"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" targetNamespace="http://www.fixprotocol.org/ns/fast/td/1.1">

<xs:element name="templates">
<xs:complexType>
<xs:choice minOccurs="0" maxOccurs="unbounded">
<xs:element ref="td:template"/>
<xs:group ref="td:other"/>
</xs:choice>
<xs:attribute name="ns"/>
<xs:attribute name="templateNs"/>
<xs:attribute name="dictionary">
<xs:simpleType>
<xs:union memberTypes="xs:string">
<xs:simpleType>
<xs:restriction base="xs:token">
<xs:enumeration value="template"/>
<xs:enumeration value="type"/>
<xs:enumeration value="global"/>
</xs:restriction>
</xs:simpleType>
</xs:union>
</xs:simpleType>
</xs:attribute>
<xs:attributeGroup ref="td:other"/>
</xs:complexType>
</xs:element>

<xs:element name="template">
<xs:complexType>
<xs:choice minOccurs="0" maxOccurs="unbounded">
<xs:choice>
<xs:element ref="td:typeRef"/>
<xs:group ref="td:instruction"/>
</xs:choice>
<xs:group ref="td:other"/>
</xs:choice>
<xs:attributeGroup ref="td:templateNsName"/>
<xs:attribute name="ns"/>
<xs:attribute name="dictionary">
<xs:simpleType>
<xs:union memberTypes="xs:string">
<xs:simpleType>
<xs:restriction base="xs:token">
<xs:enumeration value="template"/>
<xs:enumeration value="type"/>
<xs:enumeration value="global"/>
</xs:restriction>
</xs:simpleType>
</xs:union>
</xs:simpleType>
</xs:attribute>
<xs:attributeGroup ref="td:other"/>
</xs:complexType>
</xs:element>

<xs:element name="typeRef">
<xs:complexType>
<xs:group ref="td:other"/>
<xs:attributeGroup ref="td:nameAttr"/>
<xs:attribute name="ns"/>
<xs:attributeGroup ref="td:other"/>
</xs:complexType>
</xs:element>

<xs:group name="instruction">
<xs:choice>
<xs:group ref="td:field"/>
<xs:element ref="td:templateRef"/>
```

```
</xs:choice>
</xs:group>

<xs:group name="fieldInstrContent">
<xs:sequence>
<xs:choice minOccurs="0" maxOccurs="unbounded">
<xs:group ref="td:fieldOp"/>
<xs:group ref="td:other"/>
</xs:choice>
</xs:sequence>
</xs:group>

<xs:attributeGroup name="fieldInstrContent">
<xs:attributeGroup ref="td:nsName"/>
<xs:attribute name="presence">
<xs:simpleType>
<xs:restriction base="xs:token">
<xs:enumeration value="mandatory"/>
<xs:enumeration value="optional"/>
</xs:restriction>
</xs:simpleType>
</xs:attribute>
<xs:attributeGroup ref="td:other"/>
</xs:attributeGroup>

<xs:group name="field">
<xs:choice>
<xs:group ref="td:integerField"/>
<xs:element ref="td:decimal"/>
<xs:group ref="td:stringField"/>
<xs:element ref="td:byteVector"/>
<xs:element ref="td:sequence"/>
<xs:element ref="td:group"/>
</xs:choice>
</xs:group>

<xs:complexType name="integerField">
<xs:group ref="td:fieldInstrContent"/>
<xs:attributeGroup ref="td:fieldInstrContent"/>
</xs:complexType>

<xs:group name="integerField">
<xs:choice>
<xs:element ref="td:int32"/>
<xs:element ref="td:uInt32"/>
<xs:element ref="td:int64"/>
<xs:element ref="td:uInt64"/>
</xs:choice>
</xs:group>

<xs:element name="int32" type="td:integerField"/>
<xs:element name="uInt32" type="td:integerField"/>
<xs:element name="int64" type="td:integerField"/>
<xs:element name="uInt64" type="td:integerField"/>

<xs:element name="decimal">
<xs:complexType>
<xs:choice minOccurs="0" maxOccurs="unbounded">
<xs:choice>
<xs:group ref="td:fieldOp"/>
<xs:choice>
<xs:element ref="td:exponent"/>
<xs:element ref="td:mantissa"/>
</xs:choice>
</xs:choice>
<xs:group ref="td:other"/>
</xs:choice>
<xs:attributeGroup ref="td:nsName"/>
<xs:attribute name="presence">
<xs:simpleType>
<xs:restriction base="xs:token">
<xs:enumeration value="mandatory"/>
<xs:enumeration value="optional"/>
</xs:restriction>
</xs:simpleType>
</xs:attribute>
<xs:attributeGroup ref="td:other"/>
</xs:complexType>
</xs:element>
```

```
<xs:element name="exponent">
<xs:complexType>
<xs:choice minOccurs="0" maxOccurs="unbounded">
<xs:group ref="td:fieldOp"/>
<xs:group ref="td:other"/>
</xs:choice>
<xs:attributeGroup ref="td:other"/>
</xs:complexType>
</xs:element>

<xs:element name="mantissa">
<xs:complexType>
<xs:choice minOccurs="0" maxOccurs="unbounded">
<xs:group ref="td:fieldOp"/>
<xs:group ref="td:other"/>
</xs:choice>
<xs:attributeGroup ref="td:other"/>
</xs:complexType>
</xs:element>

<xs:group name="stringField">
<xs:sequence>
<xs:element name="string">
<xs:complexType>
<xs:sequence>
<xs:group minOccurs="0" ref="td:byteVectorLength"/>
<xs:group ref="td:fieldInstrContent"/>
</xs:sequence>
<xs:attributeGroup ref="td:fieldInstrContent"/>
<xs:attribute name="charset">
<xs:simpleType>
<xs:restriction base="xs:token">
<xs:enumeration value="ascii"/>
<xs:enumeration value="unicode"/>
</xs:restriction>
</xs:simpleType>
</xs:attribute>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:group>

<xs:element name="byteVector">
<xs:complexType>
<xs:sequence>
<xs:group minOccurs="0" ref="td:byteVectorLength"/>
<xs:group ref="td:fieldInstrContent"/>
</xs:sequence>
<xs:attributeGroup ref="td:fieldInstrContent"/>
</xs:complexType>
</xs:element>

<xs:group name="byteVectorLength">
<xs:sequence>
<xs:element name="length">
<xs:complexType> <xs:attributeGroup ref="td:nsName"/> </xs:complexType>
</xs:element>
</xs:sequence>
</xs:group>

<xs:element name="sequence">
<xs:complexType>
<xs:choice minOccurs="0" maxOccurs="unbounded">
<xs:choice>
<xs:element ref="td:typeRef"/>
<xs:group ref="td:length"/>
<xs:group ref="td:instruction"/>
</xs:choice>
<xs:group ref="td:other"/>
</xs:choice>
<xs:attributeGroup ref="td:nsName"/>
<xs:attribute name="presence">
<xs:simpleType>
<xs:restriction base="xs:token">
<xs:enumeration value="mandatory"/>
<xs:enumeration value="optional"/>
</xs:restriction>
</xs:simpleType>
```

```
</xs:attribute>
<xs:attribute name="dictionary">
<xs:simpleType>
<xs:union memberTypes="xs:string">
<xs:simpleType>
<xs:restriction base="xs:token">
<xs:enumeration value="template"/>
<xs:enumeration value="type"/>
<xs:enumeration value="global"/>
</xs:restriction>
</xs:simpleType>
</xs:union>
</xs:simpleType>
</xs:attribute>
<xs:attributeGroup ref="td:other"/>
</xs:complexType>
</xs:element>
<xs:group name="length">
<xs:sequence>

<xs:element name="length">
<xs:complexType>
<xs:choice minOccurs="0" maxOccurs="unbounded">
<xs:group ref="td:fieldOp"/>
<xs:group ref="td:other"/>
</xs:choice>
<xs:attribute name="name" type="xs:token"/>
<xs:attribute name="ns"/>
<xs:attribute name="id" type="xs:token"/>
<xs:attributeGroup ref="td:other"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:group>

<xs:element name="group">
<xs:complexType>
<xs:choice minOccurs="0" maxOccurs="unbounded">
<xs:choice>
<xs:element ref="td:typeRef"/>
<xs:group ref="td:instruction"/>
</xs:choice>
<xs:group ref="td:other"/>
</xs:choice>
<xs:attributeGroup ref="td:nsName"/>
<xs:attribute name="presence">
<xs:simpleType>
<xs:restriction base="xs:token">
<xs:enumeration value="mandatory"/>
<xs:enumeration value="optional"/>
</xs:restriction>
</xs:simpleType>
</xs:attribute>
<xs:attribute name="dictionary">
<xs:simpleType>
<xs:union memberTypes="xs:string">
<xs:simpleType>
<xs:restriction base="xs:token">
<xs:enumeration value="template"/>
<xs:enumeration value="type"/>
<xs:enumeration value="global"/>
</xs:restriction>
</xs:simpleType>
</xs:union>
</xs:simpleType>
</xs:attribute>
<xs:attributeGroup ref="td:other"/>
</xs:complexType>
</xs:element>

<xs:group name="fieldOp">
<xs:choice>
<xs:element ref="td:constant"/>
<xs:element ref="td:default"/>
<xs:element ref="td:copy"/>
<xs:element ref="td:increment"/>
<xs:element ref="td:delta"/>
```

```
<xs:element ref="td:tail"/>
</xs:choice>
</xs:group>

<xs:element name="constant">
<xs:complexType>
<xs:group ref="td:other"/>
<xs:attributeGroup ref="td:initialValueAttr"/>
<xs:attributeGroup ref="td:other"/>
</xs:complexType>
</xs:element>

<xs:element name="default">
<xs:complexType>
<xs:group ref="td:other"/>
<xs:attribute name="value"/>
<xs:attributeGroup ref="td:other"/>
</xs:complexType>
</xs:element>

<xs:element name="copy" type="td:opContext"/>
<xs:element name="increment" type="td:opContext"/>
<xs:element name="delta" type="td:opContext"/>
<xs:element name="tail" type="td:opContext"/>
<xs:attributeGroup name="initialValueAttr"> <xs:attribute use="required" name="value"/>
</xs:attributeGroup>
<xs:complexType name="opContext">
<xs:group ref="td:other"/>
<xs:attribute name="dictionary">
<xs:simpleType>
<xs:union memberTypes="xs:string">
<xs:simpleType>
<xs:restriction base="xs:token">
<xs:enumeration value="template"/>
<xs:enumeration value="type"/>
<xs:enumeration value="global"/>
</xs:restriction>
</xs:simpleType>
</xs:union>
</xs:simpleType>
</xs:attribute>
<xs:attribute name="key" type="xs:token"/>
<xs:attribute name="ns"/>
<xs:attribute name="value"/>
<xs:attributeGroup ref="td:other"/>
</xs:complexType>

<xs:element name="templateRef">
<xs:complexType>
<xs:group ref="td:other"/>
<xs:attribute name="name" type="xs:token"/>
<xs:attribute name="templateNs"/>
<xs:attributeGroup ref="td:other"/>
</xs:complexType>
</xs:element>

<xs:attributeGroup name="nsName">
<xs:attributeGroup ref="td:nameAttr"/>
<xs:attribute name="ns"/>
<xs:attribute name="id" type="xs:token"/>
</xs:attributeGroup>

<xs:attributeGroup name="templateNsName">
<xs:attributeGroup ref="td:nameAttr"/>
<xs:attribute name="templateNs"/>
<xs:attribute name="id" type="xs:token"/>
</xs:attributeGroup>
<xs:attributeGroup name="nameAttr"> <xs:attribute use="required" name="name" type="xs:token"/>
</xs:attributeGroup>

<xs:group name="other">
<xs:sequence> <xs:group minOccurs="0" ref="td:foreignElm" maxOccurs="unbounded"/> </xs:sequence>
</xs:group>

<xs:attributeGroup name="other"> <xs:attributeGroup ref="td:foreignAttr"/> </xs:attributeGroup>

<xs:group name="foreignElm">
<xs:choice>
<xs:any namespace="##other" processContents="skip"/>
<xs:any namespace="##local" processContents="skip"/>
```

```
</xs:choice>
</xs:group>

<xs:attributeGroup name="foreignAttr"> <xs:anyAttribute namespace="##other" processContents="skip"/>
</xs:attributeGroup>

</xs:schema>
```

附录3 范例(非规范化)

附录3.1 数据类型范例

附录3.1.1 FAST v1.1带符号整数范例

1. Int32 范例–可选的正数

<int32 id="1" presence="optional" name="Value"/>

输入值	本地十六进制/二进制	FAST 十六进制/二进制
942755	0x0e 0x62 0xa3 00001110 01100010 10100011	0x39 0x45 0xa4 **0**0111001 **0**1000101 **1**0100100
		停止位由黑体表示。 符号位由下划线表示。 字段可选且值非负，所以自加1.

2. Int32范例 – 必要的正数

<int32 id="1" presence="mandatory" name="Value"/>

输入值	本地十六进制/二进制	FAST 十六进制/二进制
942755	**0x0e 0x62 0xa3** **00001110 01100010 10100011**	**0x39 0x45 0xa3** **0**0111001 **0**1000101 **1**0100011
		停止位由黑体表示。 符号位由下划线表示。 字段为必要的，因此不自加1.

3. Int32范例 – 可选的负数

<int32 id="1" presence="optional" name="Value"/>

输入值	本地十六进制/二进制	FAST 十六进制/二进制
-942755	0xf1 0x9d 0x5d 11110001 10011101 01011101	0x46 0x3a 0xdd **0**1000110 **0**0111010 **1**1011101
	最左端字节的高值被抛掉。	停止位由黑体表示。 符号位由下划线表示。 值为负数，因此不用自加1.

4. Int32范例 –必要的负数

<int32 id="1" presence="mandatory" name="Value"/>

输入值	本地十六进制/二进制	FAST 十六进制/二进制
-7942755	0xff 0x86 0xcd 0x9d 11111111 10000110 11001101 10011101	0x7c 0x1b 0x1b 0x9d **0**1111100 **0**0011011 **0**0011011 **1**0011101

	最左端位的高值被抛掉	停止位由黑体表示。 符号位由下划线表示。 必要的字段，因此不自加1.

5. Int32范例 –带符号位扩展的必要的整数

<int32 id="1" presence="mandatory" name="Value"/>

输入值	本地十六进制/二进制	FAST 十六进制/二进制
8193	0x20 0x01 00100000 00000001	0x00 0x40 0x81 ***0***0000000 **0**1000000 **1**0000001
		停止位由黑体表示。 符号位由下划线表示。 表示符号所必需的符号位扩展（斜体）。 字段为必要的，因此不自加1.

6. Int32范例–带符号位扩展的必要的负数

<int32 id="1" presence="mandatory" name="Value"/>

输入值	本地十六进制/二进制	FAST 十六进制/二进制
-8193	0xff 0xdf 0xff 11111111 11011111 11111111	0x7f 0x3f 0xff ***0***1111111 **0**0111111 **1**1111111
		停止位由黑体表示。 符号位由下划线表示。 符号位扩展（斜体）是表示符号所必需的 字段为必要的，因此不自加1.

附录 3.1.2 FAST v1.1无符号整数范例

1. uInt32范例–可选的数

<uInt32 id="1" presence="optional" name="Value"/>

输入值	本地十六进制/二进制	FAST 十六进制/二进制
null	n/a	0X80 **1**0000000
0	0x00 0	0x81 **1**0000001
1	0x01 1	0x82 **1**0000010
942755	0x0e 0x62 0xa3 1110 01100010 10100011	0x39 0x45 0xa4 **0**0111001 **0**1000101 **1**0100100
		停止位由黑体表示 字段可选，因此自加1

2. uInt32范例 –必要的数

<uInt32 id="1" presence="mandatory" name="Value"/>

输入值	本地十六进制/二进制	FAST 十六进制/二进制

0	0x00 0	0x80 **1**0000000
1	0x01 1	0x81 **1**0000001
942755	0x0e 0x62 0xa3 1110 01100010 10100011	0x39 0x45 0xa3 **0**0111001 **0**1000101 **1**0100011
		停止位由黑体表示 字段必要，因此不自加1

附录 3.1.3 FAST v1.1字符串范例

1. 美制ASCII码字符串范例–可选的字符串

```
<string id="1" presence="optional" name="Value"/>
```

输入值	本地十六进制/二进制	FAST 十六进制/二进制
Null	n/a	0x80 **1**0000000
ABC	0x41 0x42 0x43 01000001 01000010 01000011	0x41 0x42 0xc3 **0**1000001 **0**1000010 **1**1000011
“” – 零长度字符串	n/a	0x00 0x80 **0**0000000 **1**0000000
		停止位由黑体表示

2.美制ASCII码字符串范例–必要的字符串

```
<string id="1" presence="mandatory" name="Value"/>
```

输入值	本地十六进制/二进制	FAST 十六进制/二进制
ABC	0x41 0x42 0x43 01000001 01000010 01000011	0x41 0x42 0xc3 **0**1000001 **0**1000010 **1**1000011
“” – 零长度字符串	n/a	0x80 **1**0000000
		停止位由黑体表示

附录 3.1.4 FAST v1.1 字节向量范例

1.字节向量范例 –可选的字节向量

```
<byteVector id="1" presence="optional" name="Value"/>
```

输入值	本地十六进制/二进制	FAST 十六进制/二进制	
		长度	值
Null	n/a	0x80 **1**0000000	n/a
ABC	0x41 0x42 0x43 01000001 01000010 01000011	0x84 **1**0000011	0x41 0x42 0x43 01000001 01000010 01000011
零长度值	n/a	0x81 **1**0000001	n/a

		字段为可选，因此零长度值自加1 停止位由黑体表示

2.字节向量范例–必要的字节向量

```
<byteVector id="1" presence="mandatory" name="Value"/>
```

输入值	本地十六进制/二进制	FAST 十六进制/二进制	
		长度	值
ABC	0x41 0x42 0x43 01000001 01000010 01000011	0x83 **1**0000011	0x41 0x42 0x43 01000001 01000010 01000011
零长度值	n/a	0x80 **1**0000000	n/a
		停止位由黑体表示	

附录 3.1.5 FAST v1.1十进制数范例

1.十进制数范例–必要的正十进制数

```
<decimal id="1" presence="mandatory" name="Value"/>
```

输入值	分解的输入值		FAST 十六进制/二进制	
Ascii码	指数	尾数	指数	尾数
94275500	2	942755	0x82 **1**0000010	0x39 0x45 0xa3 **0**0111001 **0**1000101 **1**0100011
			停止位由黑体表示。 符号位由下划线表示。	

2.十进制数范例 –尾数按比例缩放的必要的正十进制数

```
<decimal id="1" presence="mandatory" name="Value"/>
```

输入值	分解的输入值		FAST 十六进制/二进制	
Ascii码	指数	尾数	指数	尾数
94275500	1	9427550	0x81 **1**0000001	0x04 0x3f 0x34 0xde **0**0000100 **0**0111111 **0**0110100 **1**1011110
			停止位由黑体表示。 符号位由下划线表示。 指数设为1，以对尾数缩放	

3.十进制数范例 –可选的正十进制数

```
<decimal id="1" presence="optional" name="Value"/>
```

输入值	分解的输入值		FAST 十六进制/二进制	
Ascii码	指数	尾数	指数	尾数

94275500	2	942755	0x83 **1**0000011	0x39 0x45 0xa3 **0**0111001 **0**1000101 **1**0100011
			停止位由黑体表示。 符号位由下划线表示。 字段可选且值非负，则指数部分自加1.	

4. 十进制数范例–必要的正十进制数

```
<decimal id="1" presence="mandatory" name="Value"/>
```

输入值	分解的输入值		FAST 十六进制/二进制	
Ascii码	**指数**	**尾数**	**指数**	**尾数**
9427.55	-2	942755	0xfe **1**1111110	0x39 0x45 0xa3 **0**0111001 **0**1000101 **1**0100011
			停止位由黑体表示。 符号位由下划线表示。	

5. 十进制数范例–可选的负十进制数

```
<decimal id="1" presence="optional" name="Value"/>
```

输入值	分解的输入值		FAST 十六进制/二进制	
Ascii码	**指数**	**尾数**	**指数**	**尾数**
-9427.55	-2	-942755	0xfe **1**1111110	0x46 0x3a 0xdd **0**1000110 **0**0111010 **1**1011101
			停止位由黑体表示。 符号位由下划线表示。	

6.十进制数范例 –采用单个字段操作符的可选的正十进制数

```
<decimal id="1" presence="optional" name="Value"> <copy/> </decimal>
```

输入值	分解的输入值		FAST 十六进制/二进制	
Ascii码	**指数**	**尾数**	**指数**	**尾数**
9427.55	-2	942755	0xfe **1**1111110	0x39 0x45 0xa3 **0**0111001 **0**1000101 **1**0100011
			停止位由黑体表示。 符号位由下划线表示。 字段可选，且在字段一级指定了操作符，则PMAP中只占用单个槽位。	

7. 十进制数范例–采用单独字段操作符的可选的正十进制数

```
<decimal id="1" presence="optional" name="Value">
<exponent> <copy/> </exponent>
<mantissa> <delta/> </mantissa>
```

</decimal>

输入值	分解的输入值		FAST 十六进制/二进制	
Ascii码	指数	尾数	指数	尾数
9427.55	-2	942755	0xfe **1**1111110	0x39 0x45 0xa3 **0**0111001 **0**1000101 **1**0100011
			停止位由黑体表示。 符号位由下划线表示。 指数在pmap中占1位，但尾数在pmap中不占位	

8.十进制数范例–带符号位扩展的可选的负十进制数

<decimal id="1" presence="optional" name="Value"/>

输入值	分解的输入值		FAST 十六进制/二进制	
Ascii码	指数	尾数	指数	尾数
-8.193	-3	-8193	0xfd **1**1111101	0x7f 0x3f 0xff ***01111111*** **0**0111111 **1**1111111
			停止位由黑体表示。 符号位由下划线表示。 符号位扩展（斜体）是指定符号所必需的。	

附录 3.2字段操作符范例

附录 3.2.1 FAST v1.1 常量操作符范例

1.常量操作符范例–必要的无符号整数

<uInt32 id="1" presence="mandatory" name="Flag"> <constant value="0"/> </uInt32>

输入值	前值	编码值	Pmap位	FAST 十六进制/二进制
0	N/A	无	不需要	无
99	N/A	错误	错误	错误
无	N/A	错误	错误	错误

2.常量操作符范例–可选的无符号整数

<uInt32 id="1" presence="optional" name="Flag"> <constant value="0"/> </uInt32>

输入值	前值	编码值	Pmap位	FAST 十六进制/二进制
0	N/A	无	1*	无
无	N/A	无	0*	无

* 使用常量操作符的可选的字段在 PMAP 中占 1 位。如果输入值等于指令上下文的初始值，则该位设置为 1。如果没有输入值，则该位设置为 0。

附录 3.2.2 FAST v1.1缺省操作符范例

1. 缺省操作符范例–必要的无符号整数

<uInt32 id="1" presence="mandatory" name="Flag"> <default value="0"/> </uInt32>

输入值	前值	编码值	Pmap位	FAST 十六进制/二进制
0	N/A	无	0	无
1	N/A	1	1	0x81 **1**0000001

2.缺省操作符范例–可选的无符号整数

<uInt32 id="1" presence="optional" name="Flag"> <default/> </uInt32>

输入值	前值	编码值	Pmap位	FAST 十六进制/二进制
无	N/A	无	0	无

附录 3.2.3 FAST v1.1拷贝操作符范例

1.拷贝操作符范例–必要的字符串

<string id="1" presence="mandatory" name="Flag"> <copy/> </string>

输入值	前值	编码值	Pmap位	FAST 十六进制/二进制
CME	无	CME	1	0x43 0x4d 0xc5 **0**1000011 **0**1001101 **1**1000101
CME	CME	无	0	无
ISE	CME	ISE	1	0x49 0x53 0xc5 **0**1001001 **0**1010011 **1**1000101

2. 拷贝操作符范例–可选的字符串的NULL值使用

<string id="1" presence="optional" name="Flag"> <copy/> </string>

输入值	前值	编码值	Pmap位	FAST 十六进制/二进制
无	无	Null	1	0x80 **1**0000000
无	Null	无	0	无
CME	Null	CME	1	0x43 0x4d 0xc5 **0**1000011 **0**1001101 **1**1000101

附录 3.2.4 FAST v1.1递增操作符范例

1. 递增操作符范例–必要的无符号整数

<uInt32 id="1" presence="mandatory" name="Flag"> <increment value="1"/> </uInt32>

输入值	前值	编码值	Pmap位	FAST 十六进制/二进制
1	1	无	0	无
2	1	无	0	无
4	2	4	1	0x84 **1**0000100

5	4	无		

附录 3.2.5 FAST v1.1差值操作符范例

1.差值操作符范例 –必要的带符号整数

<int32 id="1" presence="mandatory" name="Price"> <delta/> </int32>

输入值	前值	编码值	Pmap位	FAST 十六进制/二进制
942755	0	942755	N/A	0x39 0x45 0xa3 **0**0111001 **0**1000101 **1**0100011
942750	942755	-5	N/A	0xfb **1**1111011
942745	942750	-5	N/A	0xfb **1**1111011
942745	942745	0	N/A	0x80 **1**0000000

上面例1中的初始前值为0。缺省值可在模版中进行指定。

2.差值操作符范例 –必要的十进制数

<decimal id="1" presence="mandatory" name="Price"> <delta/> </decimal>

输入值	前值		编码值		Pmap位	FAST 十六进制/二进制	
	指数	尾数	指数	尾数		指数	尾数
9427.55	0	0	-2	942755	N/A	0xfe **1**1111110	0x39 0x45 0xa3 **0**0111001 **0**1000101 **1**0100011
9427.51	-2	942755	0	-4	N/A	0x80 **1**0000000	0xfc **1**1111100
9427.46	-2	942751	0	-5	N/A	0x80 **1**0000000	0xfb **1**1111011

3. 差值操作符范例–具有初值的必要的十进制数

<decimal id="1" presence="mandatory" name="Price"> <delta value="12000"/> </decimal>

Initial/输入值	前值 (mantissa尾数)		编码值		Pmap位	FAST 十六进制/二进制	
	指数	尾数	指数	尾数		指数	尾数
12000	0	0	N/A	N/A	N/A	N/A	N/A
12100	3	12	-2	1198	N/A	0xfe **1**1111110	0x09 0xae **0**0001001 **1**0101110
12150	1	1210	0	5	N/A	0x80 **1**0000000	0x85 **1**0000101

12200	1	1215	0	5	N/A	0x80 **1**0000000	x85 0**1**0000101

4. 差值操作符范例 –必要的字符串

```
<string id="1" presence="mandatory" name="Security"> <delta/> </string>
```

输入值	前值	编码值	减除长度	Pmap位	FAST 十六进制/二进制	
					长度	编码字符串
GEH6	空字符串	GEH6	0	N/A	0x80 **1**0000000	0x47 0x45 0x48 0xb6 **0**1000111 **0**1000101 **0**1001000 **1**0110110
GEM6	GEH6	M6	2	N/A	0x82 **1**0000010	0x4d 0xb6 **0**1001101 **1**0110110
ESM6	GEM6	ES	-2	N/A	0xfd **1**1111101	0x45 0xd3 **0**1000101 **1**1010011
RSESM6	ESM6	RS	-0	N/A	0xff **1**1111111	0x52 0xd3 **0**1010010 **1**1010011

负的减除长度用于从字符串的前端移除值。减除长度为负0用于在字符串的前端附加值。

如果在模版中没有指定缺省的字符串，则初始的前值为空字符串。

附录 3.2.6 FAST v1.1扩展范例

3.多PMAP槽位范例–带单独字段操作符的可选的正十进制数。

```
<decimal id="1" presence="optional" name="Value">
<exponent> <copy/> </exponent>
<mantissa> <copy/> </mantissa>
</decimal>
```

输入值	分解的 输入值				Pmap位	FAST 十六进制/二进制	
Ascii码	指数		尾数			指数	尾数
	值	编码	值	编码			
9427.55	-2 (无前值)	-2	942755 (无前值)	942755	11	0xfe **1**1111110	0x39 0x45 0xa3 **0**0111001 **0**1000101 **1**0100011
9427.60	-2	无	942760	942760	01	无	0x39 0x45 0xa8 **0**0111001 **0**1000101 **1**0101000
无	NULL	NULL	无	无	1	0x80 **1**0000000	无

	停止位由黑体表示。 符号位由下划线表示。 由于使用单独的字段操作符，指数和尾数各自在PMAP占单独的槽位。

附录3.3 存在图

附录 3.3.1 FAST v1.1存在图范例

不同的字段操作符使用存在位的规则总结如下表：

	是否需要占用PMAP的位元	
操作符	**必要**	**可选**
无	否	否
常量 <constant/>	否	是*
拷贝 <copy/>	是	是
缺省 <default/>	是	是
差值 <delta/>	否	否
递增 <increment/>	是	是
接尾 <tail/>	是	是

*使用常量操作符的可选的字段占用1位。如果输入值等于指令上下文中的初值的话，该位被设置为1。 如果输入值不存在，则该位被设置为0.

附录4 错误代码汇总

静态错误

ERR S1

如果按照具体XMl语法进行编码的模版的实际的格式不正确，或者不符合XML命名空间的规则，或者不符合附录1中的规划，则为一个静态错误。

ERR S2

如果为操作符指定了该操作符不适用的字段类型，则为一个静态错误。

ERR S3

如果在具体语法中由“值”参数所指定的初值不能转换为字段的类型的值，则为一个静态错误。

ERR S4

如果常量操作符没有指定初值，则为一个静态错误。

ERR S5

如果一个必要的字段的缺省操作符没有指定初值，则为一个静态错误。

动态错误

ERR D1

如果模版中的字段的类型不能转换为相应的应用层的字段的类型, 或不能由相应的应用层的字段的类型转换而来，则为一个动态错误。

ERR D2

如果流中的整数的范围超出了指定类型的范围，则为一个动态错误。

ERR D3

如果由于指数和尾数使用单独的操作符而引入了某些限制，使得一个十进制数值不能被编码，则为一个动态错误。

ERR D4

如果前值的类型与当前操作符的字段的类型不相同，则为一个动态错误。

ERR D5

如果一个必要的字段在流中不存在，且前值为未定义，在指令上下文中也无初值，则为一个动态错误。

ERR D6

如果一个必要的字段在流中没有存在，且前值为空，则为一个动态错误。

ERR D7

如果减除长度大于基值的长度，或者超出了int32的表示范围，则为一个动态错误。

ERR D8

如果对于编码器或解码器来说，静态模版引用的名称所指向的模版不可知，则为一个动态错误。

ERR D9

如果解码器无法找到与流中出现的模版标识符所关联的模版，则为一个动态错误。

ERR D10

除字符串外，字节向量与其他类型的相互转换均为一个动态错误。

ERR D11

如果字符串的语法不符合类型转换的结果类型的规则，则为一个动态错误。

ERR D12

如果块的大小前导为0，则为一个动态错误。

可报告错误

ERR R1

如果一个十进制数不能以[-63, 63]范围的指数表示，或者其尾数超过了int64的范围，则为一个可报告错误。

ERR R2

如果对一个Unicode字符串应用一个接尾或差值操作符，得到的结合值不是一个合法的UTF-8序列，则为一个可报告错误。

ERR R3

在将一个Unicode字符串转换为ASCII码字符串时，如果其包含ASCII字符集以外的字符，则为一个可报告错误。

ERR R4

如果在转换时，一个整数类型的值无法由目标整数类型来表示，则为一个可报告错误。

ERR R5

在将一个十进制数转换为整数时，如果指数为负，或者结果整数不能由目标整数类型来表示，则为一个可报告错误。

ERR R6

如果出现对一个整数的过长编码，则为一个可报告错误。

ERR R7

如果存在图过长，则为一个可报告错误。

ERR R8

如果存在图包含的位元大于所需要的位元，则为一个可报告错误。

ERR R9

如果出现对一个字符串的过长编码，则为一个可报告错误。

参考文献

SCP

Rolf Andersson. FAST Session Control Protocol Specification. FPL, 2005.

RNC

James Clark, editor. RELAX NG Compact Syntax . OASIS, 2002.

XSD

Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn, editors. XML Schema Part 1 . W3C (World Wide Web Consortium), 2001.

XML

Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, editors. Extensible Markup Language. W3C (World Wide Web Consortium), 2004.

XMLNS

Tim Bray, Dave Hollander, Andrew Layman, editors. Namespaces in XML . W3C (World Wide Web Consortium), 1999.

UNICODE

The Unicode Standard, Version 3.2. The Unicode Consortium, 2000.

TWOC

A description of two's complement at Wikipedia: http://en.wikipedia.org/wiki/Two's_complement

关键术语中英文对照表

Application Type	应用类型
Assigned	已指定
Auxiliary Identifier	辅助标识符
Basic Value	基值
Big-Endian	大尾端
Block	块
Blocksize	块大小
Byte Vector Field Instruction	字节向量字段指令
Concrete Syntax	具体语法
Constant Operator	常量操作符
Copy Operator	拷贝操作符
Decimal Field Instruction	十进制数字段指令
Default Operator	缺省操作符
Delta Operator	差值操作符
Delta Value	差值
Dictionary	字典
Dynamic Error	动态错误
Empty	空
Empty String	空字符串
Excess-1	额外减一
Explicit	显式
Exponent	指数
Field	字段
Field Instruction	字段指令
Field Operator	字段操作符
Foreign	外来
Global	全局
Group	分组
Group Field Instruction	分组字段指令
ID	标识符
Implicit	隐式
Increment Operator	递增操作符
Initial Value	初值
Instruction	指令
Instruction Context	指令上下文
Integer Field Instruction	整数字段指令
Key	关键字
Length Preamble	长度前导
Mandatory	必要
Mantissa	尾数
Message	消息
Name	名称
Namespace	命名空间
Non-Nullable	不可空
Nullable	可空
Optional	可选
Overlong	过长
Presence	存在
Presence Map, PMAP	存在图
Previous Value	前值
Primitive Field	基本字段
Primitive Type	基本类型
Reportable Error	可报告错误
Reset	重置
Scaled Number	带比例数
Schema	规划
Segment	段
Sequence	序列
Sequence Field Instruction	序列字段指令
Significant Data Bits	有效数据位
Static Error	静态错误
Stop Bit Encoded Entity	停止位编码实体
Stream	流
String Field Instruction	字符串字段指令
Subtraction Length	减除长度
Tail Operator	接尾操作符
Tail Value	尾值
Transfer Encoding, TE	传输编码
Template	模板
Template Definition, TD	模版定义
Template Identifier	模版标识符
Template Reference Instruction	模版引用指令
Type	类型
Undefined	未定义
User Defined	用户自定义
Value	值
Whitespace Trimming	空白切除
Zero-Preamble	零前导

译者[①]注：已作少量勘误，为避免可能的错误，建议参考英文原版阅读。

[①] 徐广斌译，上海证券交易所，gbxu@sse.com.cn， 欢迎交流。