iML开发手册

# 前言

iML是为嵌入式系统测试开发和仿真开发而设计的专用程序语言，iML中类似自然语言的语法极其简洁，iML从SQL、Protocol Buffer中借鉴了诸多特性，针对嵌入式系统特点进行专门设计，是开发嵌入式系统测试用例和仿真模型的最佳语言。

iML语言分为MD（Message Definition）和IC（Interface Control）两个子集，即消息定义和接口控制。MD用于定义接口间传输数据时使用的消息格式，MD中很多关键词来源于Protocol Buffer，可读性非常高；IC则用来进行接口调度与模型开发，语法上与SQL类似。

# iML语言入门

## Hello World之于iML

## iML基本规则

iML是大小写敏感的语言，iML中的关键词及内置函数名全部由小写字母或数字组成，本手册示例代码中，大写字母单词代表开发者输入的标识符。

### 空白符

在iML中任意多个连续空格或TAB符的组合构成一个空白符。

### 自定义标识

自定义标识是开发人员自定义的消息名称、字段名称、变量名称、函数名称等标识的统称，自定义标识符的命名规则是：以下划线或字母打头，后跟任意多个字母或数字。

### 常量标识

**nil**

iML中空值常量标识是nil，类似于C语言中的null。

**true**

true表示判断结果为真

**false**

false表示判断结果为假

### 整数常量

### 浮点数常量

### 字符串常量

### 计算式

### 判断式

# MD详解

## MD简介

MD（Interface Message Definition）用于定义接口间传输数据时使用的消息格式，MD的内容为一系列消息定义，有了消息定义测试用例与仿真模型才知道如何解读从输入接口收到的数据，以及在输出数据前如何进行数据打包。通常在使用IC进行开发前，首先会创建MD文档。创建MD文档可以通过图形化的工具自动生成，也可以直接书写MD代码；本章主要介绍MD代码的使用，如读者仅使用前者可略过本章内容。

## 消息定义

一个典型的消息定义格式如下：

1. message MSGNAME
2. {
3. **int** SEG1 bitcount=8;
4. **int** SEG2 bitcount=32;
5. }

一个消息的定义由消息声明和消息体两部分组成。

消息声明部分由关键字message开始，紧接着为[空白符](#_空白符_1)后跟消息名称，消息名称为开发者[自定义标识](#_标识)。

消息体定义在一组 **{ }** 内，内容为一个或多个[字段定义](#_字段定义)，每个字段定义由 **;** 结尾。

## 字段定义

字段定义由字段声明、字段属性、字段结束符三部分组成。字段声明以字段类型开始，紧接着为[空白符](#_空白符_1)后跟字段名称，字段名称为开发者定义的[标识](#_标识)；字段属性紧跟在字段名后，多个字段属性以空白符分割；字段结束符为一个 **;** 。

1. **int** SEG1 bitcount=8 repeated=100;

上面的MD代码定义了一个长度为100的8位整数数组，其中**int** SEG1是字段声明，bitcount=8和repeated=100是字段属性。

## 字段类型

MD只有四种字段类型：int、real、string、block；分别代表整型、浮点数、字符串和内存块。虽然只有四种字段类型，但通过与字段属性的灵活组合，几乎可以应对所有复杂的消息格式。

## 字段属性

字段属性是MD中最核心的内容，MD的强大之处便在于通过简单的字段属性就可以定义出各种复杂的消息格式。大部分字段属性专属于某一种字段类型，少数的如repeated属性可以用于全部四种字段类型，byteorder、encoded则适用于int和real两种字段类型。当某个字段属性未在代码中进行明确定义，则iML编译器会自动使用该属性的默认值。

字段属性的表示方式为：属性名称 = 属性值；属性名称为MD预设的名称标识，属性值为属性对应的取值（各属性的取值范围详见下文）， = 前后可以存在[空白符](#_空白符_1)。同一字段的多个属性使用空白符进行分割。同一字段定义中相同属性不能重复赋值。

### int字段属性

signed = false || true

**signed**属性标识整数是否有符号，signed=true有符号，signed=false无符号，默认值为false。

bitcount = (1-64)

**bitcount**属性标识字段按位计算的长度；取值范围为1至64的整数，默认值为8；例如：bitcount=32 表示32位的整数。

byteorder = big || small

**byteorder**属性标识字节排列顺序；byteorder=small表示低字节在前，byteorder=big表示高字节在前，默认值为big。该属性只有在bitcount=16 || 32 || 64时才有效，否则该属性被自动忽略。

encoded = primitive || inversion || complement

**encoded**属性标识字段的编码方式，encoded=primitive表示原码，encoded=inversion表示反码，encoded= complement表示补码，默认值为primitive。

default = CONST\_NUM\_VALUE

**default**属性用来给

### real字段属性

precision = double || float

**precision**属性标识浮点数的精度，precision=double表示双精度浮点数，precision= float表示单精度浮点数，默认值为double。

byteorder = big || small

**byteorder**属性标识字节排列顺序；byteorder=small表示低字节在前，byteorder=big表示高字节在前，默认值为big。

encoded = primitive || inversion || complement

**encoded**属性标识字段的编码方式，encoded=primitive表示原码，encoded=inversion表示反码，encoded= complement表示补码，默认值为primitive。

### string字段属性

tail = “TAIL\_CHAR”

**tail**属性标识字符串的结尾标志，取值范围为字符串常量，默认值为:”\0”，例如：tail=”\0”表示将0字符作为字符串的结尾标识，这也是c语言中字符串的默认格式。

aligned = 8 || 16 || 32 || 64 ||…8\*n

**aligned**属性标识字符串进行整字节对齐时的位长度，取值范围为大于0 的8的整数倍数值，默认值为8。例如aligned=32，表示字符串进行32位对齐，即字符串的按字节计算的长度为4的整数倍。

filltail = “FILL\_CHAR” || nil

**filltail**属性标识对字符串进行整字节对齐后，对产生的空字节进行赋值的填充符，取值范围为字符串常量或nil，默认值为nil。例如：filltail=nil 表示不使用filltail的属性值进行空字节填充，此时MD会使用tail属性的值作为空字节填充符。

提示：进行整字节对齐时，结合使用tail和filltail可以产生不同的空字节填充效果。当filltail的值设置为除nil以外的字符串常量时，tail属性将被忽略，整字节对齐后如果没有产生空字节，则MD不进行任何填充。当filltail的值设置为nil时，无论是否产生空字节，MD都会使用tail属性值进行结尾填充。

### block字段属性

type = MSG\_NAME

|| { MSG\_DEFINITION }

|| oneof (ENUM\_SEG\_NAME) { CASE\_DEFINITION }

**type**属性标识block字段所对应的消息类型。type属性默认值为nil，通常需手动赋值，取值范围为以下四种之一：

* blob

type = blob 表示原始二进制块，同时需要设置bytecount属性。

* [消息名称](#消息声明)

赋值给type属性的消息名称，所对应的消息必须已经进行完整定义

赋值给type属性的消息名称一定不能为当前消息的名称

* [消息体定义](#消息体定义)

将消息体定义赋值给type属性时，相当于使用一个匿名消息

此处的消息体与完整消息定义中的消息体规则完全一致

此处的消息体定义中，不能引用当前消息名称，也不能引用当前消息的字段名称

* [oneof](#oneof)执行符（详见下节）

oneof执行符的定义中，规则标识对应目标字段上执行toenum后的结果，oneof的执行结果取值规则与type属性的取值规则相同，oneof可以嵌套使用。

bytecount = (>0) || auto

**bytecount**属性标识当前字段的总长度（以字节计算），取值范围为正整数，默认值为1，表示自动计算长度。

对bytesize属性进行赋值时，可以将一个消息字段名称赋值给repeated属性，但需要注意：使用消息字段名称给bytesize属性赋值时，对应的消息字段类型必须为int且出现在当前字段之前。例如：bytesize=SEG表示将字段SEG的值作为bytesize属性的值。

另外，bytesize属性还支持[计算式](#_计算式)赋值，即将一个运行时计算结果赋值给repeated属性，计算式可以使用整数常量和int型的已定义字段名称，允许出现的运算符包括：+ - \* / （） 。例如：bytesize=SIZE\_SEG-8\*4。

## 执行符

执行符所起的作用是动态控制MD的运行时行为，当MD执行消息的拆包、打包或检查时会根据执行符所设置的参数与规则执行相应的动作。执行符只能在字段定义中使用，执行符默认执行对象为当前字段，对其它字段的引用以括号内参数的形式出现。

### repeated

repeated= (>=0) || auto

**repeated**属性标识当前字段类型连续重复的次数，取值范围为正整数或auto，默认值为1。例如：repeated=20 表示一个长度为20的数组。repeated = auto，表示自动计算重复次数。此处要特别注意，只有在以下情况中repeated = auto才有效：将当前消息赋值给一个block类型字段的type属性，且block字段的bytesize属性值不为auto。

对repeated属性进行赋值时，可以将一个消息字段名称赋值给repeated属性，但需要注意：使用消息字段名称给repeated属性赋值时，对应的消息字段类型必须为int且出现在当前字段之前。例如：repeated=SEG表示将字段SEG的值作为repeated属性的值，从而定义了一个运行时创建的可变长度数组。

另外，repeated属性还支持[计算式](#_计算式)赋值，即将一个运行时计算结果赋值给repeated属性，计算式可以使用整数常量和int型的字段名称，允许出现的运算符包括：+ - \* / ( )。例如：repeated=SIZE\_SEG-8\*4。

### autovalue与lengthof

### valuecheck

crc

max

min

validator

### toenum和oneof

toenum执行符的作用是定义枚举转换规则，当MD需要把整数字段转换为一个标识符时，MD会自动按照该字段上toenum中定义的转换规则进行转换。

用法：

toenum { 整数常量=执行结果; other=执行结果; … }

说明：

执行结果是一个合法的[自定义标识](#_自定义标识)，不同的转换规则可以返回相同的执行结果

转换规则中必须包含一条other规则，用于无法匹配其它全部规则时的默认规则

不同的转换规则中不能使用相同的整数常量

oneof执行符的作用是，先取目标字段的toenum结果，然后根据oneof中定义的规则选取对应的消息定义作为执行结果。

用法：

oneof (目标字段名称) { 规则标识=执行结果; …}

说明：

目标字段名称是同一消息定义中当前字段之前的一个整数字段的名称

规则标识与toenum的所有可能执行结果一一对应

执行结果的取值规则与block字段的[type属性](#type属性)取值规则相同

# IC详解

## IC数值类型

## IC模型设计

## IC接口操作

## IC模型操作

## IC函数

# 开发实操

# iML索引表