|  |  |  |
| --- | --- | --- |
| Syntax of SASL Symbol Name Mangling | September 26  2010 | |
| This script describes the rules of symbol name mangling. | | V1.2 |

Syntax of SASL Symbol Name Mangling

# History

|  |  |  |
| --- | --- | --- |
| 1.2 | Sep 29, 2010 | Add build-in function name mangling rules. |
| 1.1 | Sep 27, 2010 | Move scope qualifier.  Add type of member function mangling support.  Fixed type qualifier ambiguous.  Function mangling was edited. |
| 1.0.1 | Sep 26, 2010 | Fixed typo & grammar error. |
| 1.0 | Sep 26, 2010 | The document created. |

# Grammars

**Name mangling** can be applied to name of function mainly, for distinguishing the overloaded functions

MangledName is the result of Name Mangling. It starts with ‘M’ character and ends with ‘Z’ character.

MangledName ::= ‘M’ QualifiedName ‘@@’ ParameterTypes CallingConvention

QualifiedName is a scope name qualified name. A scope name could be a structure name, class name or namespace name.

ParameterTypes describes the types of parameter list.

‘@@’ is the splitter which separates qualified name and parameter type.

QualifiedName ::= ScopeQualifier BaseName

QualifiedName is composited by scope name and base name. ‘@’ is used to split the scope segments.

ScopeQualifier ::= ( ScopeName ‘@’ )\*

ScopeName ::=

‘S’ StructTypeName

‘N’ NamespaceName

There are two sorts of ScopeName , structure name and namespace name. We distinguish them by a lead character. If the name starts with ‘S’, it means the name represents structure, and ‘N’ means following name represents a namespace.

StructTypeName ::= <name of structure>

*NamespaceName ::= <name of namespace>*

*BaseName ::= <name of function>*

BaseName is the literal name of the function or a type.

ParameterTypes is a list of parameter types. Because SASL do not support variance parameters, so we need not to considerate that. Syntax of ParameterTypes as following:

ParameterTypes ::= ( ParameterType ‘@@’)\*

ParameterType is encoded by type qualifiers and type name.

***NOTE THAT***, The term “type name” appearing in this paper means the original definition name but not alias (such as generated by “typedef”).

ParameterType ::= QualifiedTypeName

QualifiedTypeName ::=TypeQualifier ScopeQualifier UnqualifiedTypeName

UnqualifiedTypeName::=

‘B’ BuildinScalarTypeName

‘V’ BuildinVectorTypeName

‘M’ BuildinMatrixTypeName

‘F’ FunctionTypeName

‘S’ StructTypeName

‘A’ ArraySize ArrayComponentTypeName

***QualifiedTypeName*** is the type name qualified by scopes and type qualifiers.

TypeQualifier ::= (‘C’)?(‘H’)?(‘S’)?(‘U’)?Q

TypeQualifier describes the qualifier of declaration. In SASL, following qualifiers are supported: constant qualifier (‘C’), uniform qualifier (‘U’) , shared qualifier (‘H’) and static qualifier (‘S’). Each qualifier is represented as a character, and qualifier sequence is ended by a character ‘Q’.

BuildinScalarTypeNames encode all build-in types into one or two characters.

StructTypeName::= < name of structure>

BuildinScalarTypeName::=

‘U1’ | ‘U2’ | ‘U4’ | ‘U8’ |‘I1’ | ‘I2’ | ‘I4’ | ‘I8’ | ‘B’ | ’F’ | ‘D’ | ‘O’

BuildinVectorName ::= VectorLength BuildinScalarTypeName

BuildinMatrixName ::= RowCountOfMatrix ColumnCountOfMatrix BuildinScalarTypeName

The selectable items of BuildinScalarTypeName are corresponded following building types: ***uint8\_t***, ***uint16\_t***, ***uint32\_t***, ***uint64\_t***, ***int8\_t***, ***int16\_t***, ***int32\_t***, ***int64\_t***, ***boolean***, ***float***, ***double*** and ***omit*** (***void***).

BuildinVectorTypeName shows the name of building vector. It makes of length component and scalar type component.

VectorLength Could be 2, 3 or 4 and represents the length of vector.

BuildinMatrixTypeName is as same composition as build-in vector type name.

RowCountOfMatrix, ColumnCountOfMatrix means the row count and column count of matrix. It could be 1, 2, 3 or 4.

ArrayComponentTypeName ::= QualfiedTypeName

ArraySize ::=

<length of array >

<epsilon>

ArraySize is an integer demonstrating the size of fixed-length array or ignore it. If no array size, it means it is a variant size array.

ArrayComponentTypeName is the description of type of component of array. For example, array component type of “const int32\_t[]” is “const int32\_t”. So the mangled name of this array type is “QACQI4”.

Note that, since array type and build in type has no scope information, they do not have any scope qualification. For example, following array are mangled as “QAQNS0@SAStruct”, not “QNS0@ASAStruct”

NS0::AStruct[] a;

CallingConvention is a single character which shows the calling convention of this function. ‘S’ means “\_\_stdcall”, ‘C’ means “\_\_cdecl” and ‘F’ ,means “\_\_fastcall” and ‘T’ means ‘\_\_thiscall’

CallingConvention ::=

‘S’

‘C’

‘F’

‘T’

FunctionTypeName reuses non-terminators and terminators mentioned above, and end with the symbol ‘@’. Note that, it looks like that a single ‘@’ character is ambiguous with scope name separator. But in fact, we know that the function is the end of a type name always, the result is ‘@’ can be combined with follow characters “@@” to a new splitter “@@@”. It not only means it the end of function, but told us it is the end of the whole composited type also, if function is the inner type of a composited type.

FunctionTypeName ::= ReturnType ‘@@’ PrarameterTypes CallConvetion ‘@’

ReturnType ::= ParameterType

NOTE THAT: If calling convention of function is **this call**, the first type of parameter types is the type of this object.

# Build-in Functions/Operators Name Mangling

Build-in operators and functions mangling is as same as user-defined functions. Particularly, since build-in function could not have a common name, so their names are pre-defined necessarily. In SASL, we use ‘0’ lead character to decorate build-in operator name for distinguishing from user-defined identifiers.

Following are operator base names:

|  |  |  |
| --- | --- | --- |
| **Operators** | **Symbols** | **Base Name** |
| Arithmetic Operators | + | 0add |
| - | 0sub |
| \* | 0mul |
| / | 0div |
| % | 0mod |
| Index Operators | [i] | 0index |
| Arithmetic Assignment Operators | = | 0assign |
| += | 0add\_assign |
| -= | 0sub\_assign |
| \*= | 0mul\_assign |
| /= | 0div\_assign |
| %= | 0mod\_assign |
| Cast Operators | (type)expr | 0cast |
| Bitwise Operators | ~ | 0bit\_not |
| << | 0shift\_left |
| >> | 0shift\_right |
| & | 0bit\_and |
| | | 0bit\_or |
| ,^ | 0bit\_xor |
| Bitwise Assignment Operators | <<= | 0lshift\_assign |
| >>= | 0rshift\_assign |
| &= | 0band\_assign |
| |=, | 0bor\_assign |
| ^= | 0bxor\_assign |
| ~= | 0bnot\_assign |
| Boolean Operators | && | 0logic\_and |
| || | 0logic\_or |
| ! | 0logic\_not |
| Comparison Operators | >= | 0greater\_equal |
| > | 0greater |
| = | 0equal |
| <= | 0less\_equal |
| < | 0less |
| != | 0not\_equal |
| Prefix/Postfix Operators | ++expr | 0prefix\_incr |
| expr++ | 0postfix\_incr |
| --expr | 0prefix\_decr |
| expr-- | 0postfix\_decr |
| Unary Operator | - | 0negative |
| + | 0positive |