Appendix C — Symbolic Debugger Stabstrings

C. Symbolic Debugger Stabstrings

The debug section contains the symbolic debugger stabstrings (symbol table strings). It is generated by the compilers and assemblers. It provides symbol attribute information for use by the symbolic debugger.

C.1. Stabstring Terminal Symbols

In the stabstring grammar, there are five types of terminal symbols, which are written in all capital letters. These symbols are described by the regular expressions in the following list:

Note: The [ ] (brackets) denote one instance, [ ]* (brackets asterisk) denote zero or more instances, [ ]+ (brackets plus sign) denote one or more instances, ( ) (parentheses) denote zero or one instance, .* (dot asterisk) denotes a sequence of zero or more bytes, and | (pipe) denotes alternatives.

Symbol	Regular Expression
NAME	[^ **; :** **’ ”**] (A name consists of any non-empty set of characters, excluding **; : ’** or **”**.)
STRING	**’.*’** | **”.*”**, where \”, \’, or \\ can be used inside the string

Within a string, the \ (backslash character) may have a special meaning. If the character following the \ is another \, one of the backslashes is ignored. If the next character is the quote character used for the current string, the string is interpreted as containing an embedded quote. Otherwise, the \ is interpreted literally. However, if the closing quote is the last character in the stabstring, and a \ occurs immediately before the quote, the \ is interpreted literally. This use is not recommended.

The \ must be quoted only in the following instances:

- The \ is the last character in the string (to avoid having the closing quote escaped).
- The \ is followed by the current quote character.
- The \ is followed by another \.

An escaped quote is required only when a single string contains both a single quote and a double quote. Otherwise, the string should be quoted with the quote character not contained in the strings.

A string can contain embedded null characters, so utilities that process stabstrings must use the length field to determine the length of a stabstring.

INTEGER	**(−)[0−9]+**
HEXINTEGER	**[**0−9A−F**]+**
	The hexadecimal digits A−F must be uppercase.
REAL	**[+−][0−9]+(.)[0−9]*([**eEqQ**](+−)[0−9]+)** | **(+−)**INF | QNAN | SNAN
	Real numbers are the same strings recognized by the **scanf** function when using the ”%lf” pattern. Therefore, white space may occur before a real number.

C.2. Stabstring Grammar

REALs may be preceded by white space, and STRINGs may contain any characters, including null and blank characters. Otherwise, there are no null or blank characters in a stabstring.

Long stabstrings can be split across multiple symbol table entries for easier handling. In the stabstring grammar, an * (asterisk) indicates a point at which a stabstring may be continued. A continuation is indicated by using the ? (question mark) or \ as the last character in the string.

The next part of the stabstring is in the name of the next symbol table entry. If an alternative for a production is empty, the grammar shows the keyword /*EMPTY*/.

The following list contains the stabstring grammar:

Stabstring Basic structure of stabstring:

NAME **:** *Class*	Name of object followed by object classification
: *Class*	Unnamed object classification.

Class Object classifications:

c = *Constant*	Constant object
NamedType	User-defined types and tags
Parameter	Argument to subprogram
Procedure	Subprogram declaration
Variable	Variable in program
Label	Label object.

Constant Constant declarations:

b *OrdValue*	Boolean constant
c *OrdValue*	Character constant
e *TypeID* **,** *OrdValue*	Enumeration constant
i INTEGER	Integer constant
r REAL	Floating point constant
s STRING	String constant
C REAL**,** REAL	Complex constant
S *TypeID* **,** *NumElements* **,** *NumBits* **,** *BitPattern*	Set constant.

OrdValue Associated numeric value:

INTEGER

NumElements: Number of elements in the set:

INTEGER

NumBits: Number of bits in item:

INTEGER

NumBytes: Number of bytes in item:

INTEGER

BitPattern: Hexadecimal representation, up to 32 bytes:

HEXINTEGER

NamedType: User-defined types and tags:

t *TypeID*	User-defined type (TYPE or typedef), excluding those that are valid for **T** *TypeID*
T *TypeID*	Struct, union, class, or enumeration tag.

Parameter: Argument to procedure or function:

a *TypeID*	Passed by reference in general register
p *TypeID*	Passed by value on stack
v *TypeID*	Passed by reference on stack
C *TypeID*	Constant passed by value on stack

D *TypeID*	Passed by value in floating point register
R *TypeID*	Passed by value in general register.

Procedure: Procedure or function declaration:

Proc — Procedure at current scoping level

Proc **,** NAME **:** NAME
: Procedure named 1st NAME, local to 2nd NAME, where 2nd NAME is different from the current scope.

Variable: Variable in program:

TypeID	Local (automatic) variable of type *TypeID*
d *TypeID*	Floating register variable of type *TypeID*
r *TypeID*	Register variable of type *TypeID*
G *TypeID*	Global (external) variable of type *TypeID*
S *TypeID*	Module variable of type *TypeID* (C static global)
V *TypeID*	Own variable of type *TypeID* (C static local)
Y	FORTRAN pointer variable
Z *TypeID* : NAME	FORTRAN pointee variable.

Label: Label:

L — Label name.

Proc: Different types of functions and procedures:

f *TypeID*	Private function of type TypeID
g *TypeID*	Generic function (FORTRAN)
m *TypeID*	Module (Modula−2, ext. Pascal)
J *TypeID*	Internal function of type *TypeID*
F *TypeID*	External function of type *TypeID*
I	(capital i) Internal procedure
P	External procedure
Q	Private procedure.

TypeID: Type declarations and identifiers:

INTEGER	Type number of previously defined type
INTEGER **=** *TypeDef*	New type number described by *TypeDef*
INTEGER **=** *TypeAttrs TypeDef*	New type with special type attributes.

TypeAttrs: @ *TypeAttrList* **;**

Note: Type attributes (*TypeAttrs)* are extra information associated with a type, such as alignment constraints or pointer-checking semantics.

TypeAttrList: List of special type attributes:

TypeAttrList **;** @ *TypeAttr*
TypeAttr

TypeAttr: Special type attributes:

a INTEGER	Align boundary
s INTEGER	Size in bits
p INTEGER	Pointer class (for checking)
P	Packed type
Other	Anything not covered is skipped entirely.

TypeDef:	Basic descriptions of objects:	
	INTEGER	Type number of a previously defined type
	b *TypeID* **; *** *NumBytes*	Pascal space type
	c *TypeID* **; *** *NumBits*	Complex type *TypeID*
	d *TypeID*	File of type *TypeID*
	e *EnumSpec* **;**	Enumerated type (default size, 32 bits)
	g *TypeID* **; *** *NumBits*	Floating-point type of size *NumBits*.

For **i** types, *ModuleName* refers to the Modulo-2 module from which it is imported.

i NAME **:** NAME **;**	Imported type *ModuleName***:***Name*
i NAME **:** NAME **,** *TypeID* **;**	Imported type ModuleName:Name of type *TypeID*
k *TypeID*	C++ constant type
l ; *	Usage-is-index; specific to COBOL
m *OptVBaseSpec OptMultiBaseSpec TypeID* **:** *TypeID* **:** *TypeID* **;**	C++ pointer to member type; the first *TypeID* is the member type; the second is the type of the class.
n *TypeID* **; *** *NumBytes*	String type, with maximum string length indicated by *NumBytes*
o NAME **;**	Opaque type
o NAME **,** *TypeID*	Opaque type with definition of *TypeID*
w *TypeID*	Wide character
z *TypeID* **; *** *NumBytes*	Pascal gstring type
C *Usage*	COBOL Picture
I *NumBytes* **; *** *PicSize*	(uppercase i) Index is type; specific to COBOL
K *CobolFileDesc***;**	COBOL File Descriptor
M *TypeID* **; *** Bound	Multiple instance type of *TypeID* with length indicated by INTEGER
N	Pascal *Stringptr*
S *TypeID*	Set of type *TypeID*
* *TypeID*	Pointer of type *TypeID.* The * in this alternative is a literal *, not a possible continuation point.
& *TypeID*	C++ reference type
V *TypeID*	C++ volatile type
Z	C++ ellipses parameter type
Array	
Subrange	
ProcedureType	For function types rather than declarations
Record	Record, structure, union, or group types.

EnumSpec:	List of enumerated scalars:	
	EnumList	Enumerated type (C and other languages)

TypeID : *EnumList*
C++ enumerated type with repeating integer type

EnumList:

Enum
EnumList Enum

Enum: Enumerated scalar description:

NAME **:** *OrdValue* **, ***

Array: Array descriptions:

a *TypeID* **; *** *TypeID*
Array; *TypeID* is the index type
A *TypeID* Open array of *TypeID*
D INTEGER **,** *TypeID*
N-dimensional dynamic array of *TypeID*
E INTEGER **,** *TypeID*
N-dimensional dynamic subarray of *TypeID*
O INTEGER **,** *TypeID*
New open array
P *TypeID* **; *** *TypeID*
Packed array.

Subrange: Subrange descriptions:

r *TypeID* **;*** *Bound* **;*** *Bound*
Subrange type (for example, char, int, ...), lower and upper bounds

Bound: Upper and lower bound descriptions:

INTEGER Constant bound.
Boundtype INTEGER
Variable or dynamic bound;
value is address of or offset to bound.
J Bound is indeterminable (no bounds).

Boundtype: Adjustable subrange descriptions:

A Bound passed by reference on stack
S Bound passed by value in static storage
T Bound passed by value on stack
a Bound passed by reference in register
t Bound passed by value in register.

ProcedureType:
Function variables (1st type C only; others Modulo-2 & Pascal)

f *TypeID* **;** Function returning type *TypeID*
f *TypeID* **,** *NumParams* **;** *TParamList* **;**
Function of N parameters returning type *TypeID*
p *NumParams* **;** *TParamList* **;**
Procedure of N parameters
R *NumParams* **;** *NamedTParamList*
Pascal function parameter
F *TypeID* **,** *NumParams* **;** *NamedTParamList* **;**
Pascal function parameter.

NumParams: Number of parameters in routine:

INTEGER.

TParamList: Types of parameters in Modulo-2 function variable:

TParam Type of parameter and passing method
TParamList TParam

TParam: Type and passing method

TypeID **,** *PassBy* **; ***

NamedTParamList:
Types of parameters in Pascal-routine variable:

/*EMPTY*/
NamedTPList

NamedTPList:

NamedTParam
NamedTPList NamedTParam

NamedTParam:
Named type and passing method:

Name **:** *TypeID* **,** *PassBy InitBody* **;***
: *TypeID* **,** *PassBy InitBody* **;***
Unnamed parameter

Record: Types of structure declarations:

s *NumBytes * FieldList* **;**
Structure or record definition
u *NumBytes * FieldList* **;**
Union
v *NumBytes * FieldList VariantPart* **;**
Variant Record
Y *NumBytes* ClassKey OptPBV OptBaseSpecList* **(** *ExtendedFieldList OptNameResolutionList* **;**
C++ class
G *Redefinition* **, n** *NumBits * FieldList* **;**
COBOL group without conditionals
Gn *NumBits FieldList* **;**
G *Redefinition* **, c** *NumBits * CondFieldList* **;**
COBOL group with conditionals
Gc *NumBits CondFieldList* **;**

OptVBaseSpec:

v ptr-to-mem class has virtual bases.
/*EMPTY*/ Class has no virtual bases.

OptMultiBaseSpec:

m Class is multi-based.
/*EMPTY*/ Class is not multi-based.

OptPBV:

V Class is always passed by value.
/*EMPTY*/ Class is never passed by value.

ClassKey:

s struct

	u	union
	c	class
OptBaseSpecList:		
	/*EMPTY*/	
	BaseSpecList	
BaseSpecList:		
	BaseSpec	
	BaseSpecList **,** *BaseSpec*	
BaseSpec:		
	VirtualAccessSpec BaseClassOffset **:** *ClassTypeID*	
BaseClassOffset:		
	INTEGER	Base record offset in bytes
ClassTypeID:		
	TypeID	Base class type identifier
VirtualAccessSpec:		
	v *AccessSpec*	Virtual
	AccessSpec	
GenSpec:		
	c	Compiler-generated
	/*EMPTY*/	
AccessSpec:		
	i	Private*
	o	Protected*
	u	Public*
AnonSpec:		
	a	Anonymous union member
	/*EMPTY*/	
VirtualSpec:		
	v p	Pure virtual
	v	Virtual
	/*EMPTY*/	
ExtendedFieldList:		
	ExtendedFieldList ExtendedField	
	/*EMPTY*/	
ExtendedField:		
	GenSpec AccessSpec AnonSpec DataMember	
	GenSpec VirtualSpec AccessSpec OptVirtualFuncIndex MemberFunction	
	AccessSpec AnonSpec NestedClass	
	AnonSpec FriendClass	
	AnonSpec FriendFunction	
DataMember:		

MemberAttrs **:** *Field*

MemberAttrs:

IsStatic IsVtblPtr IsVBasePtr

IsStatic:

/*EMPTY*/
s Member is static.

IsVtblPtr:

/*EMPTY*/
p INTEGER NAME
Member is vtbl pointer; NAME is the external name of v-table.

IsVBasePtr:

/*EMPTY*/
b Member is vbase pointer.
r Member is vbase self-pointer.

MemberFunction:

[*FuncType MemberFuncAttrs* **:** NAME **:** *TypeID* **;***

MemberFuncAttrs:

MemberFuncAttr
MemberFuncAttrs MemberFuncAttr

MemberFuncAttr:

IsStatic
IsInline
IsConst
IsVolatile

IsInline:

/*EMPTY*/
i Inline function

IsConst:

/*EMPTY*/
k const member function

IsVolatile:

/*EMPTY*/
V Volatile member function

NestedClass:

N *TypeID* **;***

FriendClass:

(*TypeID* **;***

FriendFunction:

] NAME **:** *TypeID* **;***

OptVirtualFuncIndex:

/*EMPTY*/
INTEGER

FuncType:

f	Member function
c	Constructor
d	Destructor

InitBody:

STRING
/*EMPTY*/

OptNameResolutionList:

/*EMPTY*/
) *NameResolutionList*

NameResolutionList:

NameResolution
NameResolution **,** *NameResolutionList*

NameResolution:

MemberName **:** *ClassTypeID*
Name is resolved by compiler.
MemberName **:** Name is ambiguous.

MemberName:

NAME

FieldList: Structure content descriptions:

/*EMPTY*/
FieldList Field

Field: Structure-member type description:

NAME **:** *TypeID* **,** *BitOffset* **,** *NumBits* **; ***

VariantPart: Variant portion of variant record:

[*Vtag VFieldList*]
Variant description.

VTag: Variant record tag:

(*Field*	Member of variant record
(NAME **: ; ***	Variant key name .

VFieldList: Variant record content descriptions:

VList Member of variant record.
VFieldList VList

VList: Variant record fields:

VField Member of variant record.
VField VariantPart

VField: Variant record member type description:

(*VRangeList* **:** *FieldList*
Variant with field list.

VRangeList: List of variant field labels:

VRange	Member of variant record
VRangeList **,** *VRange*	

VRange: Variant field descriptions:

b *OrdValue*	Boolean variant
c *OrdValue*	Character variant
e *TypeID* **,** *OrdValue*	Enumeration variant
i INTEGER	Integer variant
r *TypeID* **;*** *Bound* **;*** *Bound*	Subrange variant.

CondFieldList:

*Conditions *FieldList*
**FieldList*

BitOffset: Offset in bits from beginning of structure:

INTEGER.

Usage: Cobol usage description:

PICStorageType NumBits **,** *EditDescription* **,** *PicSize* **;**

Redefinition **,** *PICStorageType NumBits* **,** *EditDescription* **,** *PicSize* **;**

PICStorageType NumBits **,** *EditDescription* **,** *PicSize* **, *** *Conditions* **;**

Redefinition **,** *PICStorageType NumBits* **,** *EditDescription* **,** *PicSize* **, *** *Conditions* **;**

Redefinition: Cobol redefinition:

r NAME

PICStorageType:
Cobol PICTURE types:

a	Alphabetic
b	Alphabetic, edited
c	Alphanumeric
d	Alphanumeric, edited
e	Numeric, signed, trailing, included
f	Numeric, signed, trailing, separate
g	Numeric, signed, leading, included
h	Numeric, signed, leading, separate
i	Numeric, signed, default, comp
j	Numeric, unsigned, default, comp
k	Numeric, packed, decimal, signed
l	Numeric, packed, decimal, unsigned
m	Numeric, unsigned, comp-x
n	Numeric, unsigned, comp-5
o	Numeric, signed, comp-5
p	Numeric, edited
q	Numeric, unsigned
s	Indexed item
t	Pointer.

EditDescription:
Cobol edit description:

STRING	Edit characters in an alpha PIC
INTEGER	Decimal point position in a numeric PIC.

PicSize: Cobol description length:

INTEGER	Number of repeated '9's in numeric clause, or length of edit format for edited numeric.

Condition: Conditional variable descriptions:

NAME **:** INTEGER **= q** *ConditionType* **,** *ValueList* **; ***

Conditions: Conditional variable descriptions:

Condition
Conditions Condition

ConditionType: Conditions descriptions:

ConditionPrimitive **,** *KanjiChar*

ConditionPrimitive:
Primitive type of Condition:

n *Sign DecimalSite*	Numeric conditional
a	Alphanumeric conditional
f	Figurative conditional.

Sign: For types with explicit sign:

+	Positive
−	Negative
Any other character	(No explicit sign)

DecimalSite: Number of places from left for implied decimal point:

INTEGER.

KanjiChar: 0 only if Kanji character in value:

INTEGER.

ValueList Values associated with condition names

Value
ValueList Value.

Value Values associated with condition names:

INTEGER **:** *ArbitraryCharacters*
Integer indicates length of string.

CobolFileDesc: COBOL file description:

Organization AccessMethod NumBytes

Organization: COBOL file-description organization:

i	Indexed
l	Line Sequential
r	Relative
s	Sequential.

AccessMethod: COBOL file description access method:

d	Dynamic
o	Sort
r	Random
s	Sequential.

PassBy: Parameter passing method:

INTEGER	0 = passed-by reference; 1 = passed-by value.