NAME

param – C++ language binding for method parameters

SYNOPSIS

```
struct typ; // external or defined in SDL

interface I {
public:
    void op1(
        in          int a,
        inout       int b,
        out         int c
    );

    void op2(
        in          string a,
        in          ref<I> b,
        in          int c,
        in          lref<int>   d,
        in          lref<char>  e,
        in          sequence<char>  f,
        in          set<I>  g
    );
};

// yields the following prototypes
void
I:: op1 ( long a , long &b , long &c );

void
I:: op2 (
    sdl_string a,
    Ref< I  >  b,
    long       c,
    LREF(long) d,
    LREF(char) e,
    Sequence<char> f,
    Set<Ref<I> > g
) const ;

// where

#define LREF(T) T *
```

DESCRIPTION

The function prototype for an SLD operation (method, member function) is generated as follows:

in is passed by value, *regardless of the size of the argument.*

inout is passed by reference.

out is passed by reference.

All methods are virtual.

In addition, the following translations are made:

lref<x>
is translated to `x *`

string
is translated to `sdl_string`

ref<I>
is translated to `Ref<I>`

int is translated to `long`

sequence
is translated to `Sequence`

set<I>
is translated to `set<Ref<I> >`. *This translation is not made for sequences.*

TEMPLATE INSTANTIATIONS

If a template type (sequence, set) is used more than once in argument declarations, when the module is compiled with

```
#define MODULE_CODE
```

to instantiate the templates, you will get a warning about duplicate template instantiations. To avoid that warning, use a `typedef` declaration for the template type, and use that, e.g.,

```
typedef sequence<int> intseq;
void op1(
    // input arguments ...

    out   intseq b,  // replaces out sequence<int> b
    out   intseq c   // replaces out sequence<int> c
);
```

SEE ALSO

intro(oc).