8. ACTUAL PARAMETERS

For <type> quantities, the call by value causes an assignment to be made to a local variable. In the same way, a call by default (text only) will cause a denotes operation to be performed to a local variable.

For classes, the parameter checking and type conversion are done by the compiler which will also generate the necessary in-line coding to store the values into the class instance.

For procedures, with the exception of external, formal and virtual ones, the value parameters are checked, type converted and stored within the procedure instance by compiler generated in-line coding.

Name parameters have been partly checked by the compiler in that all errors that do not depend on actual values at runtime are detected.

For external, formal and virtual procedures, the parameter correspondence cannot be checked at compile time. Thus the checking must be done by the appropriate runtime system routines.

For external, formal and virtual procedures, all parameters have a static parameter descriptor (spd).

The contents of an spd are as follows:

All spd's contain type, kind and spd-type information.

Depending on the spd-type information, the rest of the spd contains:

1. A constant address.
   Actual parameter is a <type> constant (real, integer, Boolean, character, text or ref).

2. A block level and a relative data address.
   Actual parameter is a simple variable (except label) or a formal parameter.

3. A block level and a switch (or label) address.
   Actual parameter is a switch (or label) name.

4. A block level and a prototype pointer.
   Actual parameter is a procedure name.

5. A thunk address.
   Actual parameter is a subscripted variable, remote variable or an expression.

Any reference parameter spd must contain or give access to the qualification of the actual parameter, i.e. the identification (prototype pointer) of the qualifying class and its apparent block level.

The appropriate runtime subroutine will, for name parameters, convert an spd to a dynamic parameter descriptor (dpd).

For value and default parameters, a dpd is not formed, but the parameter is evaluated and the result stored within the procedure instance. The exceptions to this rule are switches, labels and procedures which get a dpd and are handled (from the compiler's point of view) as name parameters.

The contents of the different possible dpd are given below using the following notation:

tha	thunk address
dp	driver pointer
ra	relative address
oa	object address
swa	switch address
la	label address
pp	prototype pointer
ca	constant address

Possible dpds are:

<type> constant:	ca
simple <type> variable:	ra,oa or pp,dp
<type> subscripted variable or <type> remote variable or <type> expression:	tha,dp
switch name or default	swa,dp
designational expression:	tha,dp
label name, value label:	la,dp
procedure name or default	pp,dp
array default	no dpd. A copy of array descriptor.

Any reference dpd must contain or give access to the identification (prototype pointer) of the class qualifying the actual parameter.