15. PERFORMANCE ISSUES

Three main areas in LOOPS can affect performance:

- Garbage collection
- Instance variable access
- Method lookup

This chapter describes the impact of these areas on LOOPS. Also included is a section on cache clearing.

15.1 Garbage Collection

The Interlisp-D garbage collector maintains reference counts of each piece of data in the system. (Refer to the *Lisp Release Notes* and the *Interlisp-D Reference Manual* for information on reference counts.) There is potential for noticeable performance degradation if many items have reference counts greater than one. Object-oriented systems in general, and LOOPS in particular, can easily create objects that have multiple references.

The LOOPS system uses a number of methods to avoid creating items with large reference counts. Classes, for example, can easily have large reference counts since each instance of the class points to the class. Because of this, LOOPS does not maintain reference counts of classes. Performance is enhanced, but classes in LOOPS are not garbage collected. This should not present a problem as classes are not often destroyed.

Unique Identifiers (UIDs) also have multiple references: from the instance they name and from the table used by the LOOPS system to associate UIDs with instances. LOOPS avoids this problem by storing copies of the instance UID in the instance. This complicates testing for equality of UIDs, which is a rare event, but removes a potential garbage collection problem.

These and other implementation details substantially reduce the impact of LOOPS on the Interlisp-D garbage collector. In a typical running system, LOOPS objects accounted for less than 16% of the data items with reference count greater than one.

15.2 INSTANCE VARIABLE ACCESS

15.2 Instance Variable Access

LOOPS uses macros to speed the instance variable access from compiled code. Instance variable property access is compiled differently from instance variable value access, and various caching schemes are used to speed up repeated access to a given slot.

LOOPS uses two layers of caching to speed up instance variable access:

- Local cache.

  Instance variable access from compiled code uses a local cache. This cache remembers the class of *self* and the instance variable index the last time this piece of code was executed. If the class of *self* on the next pass through the code matches the stored value, then the stored instance variable index is used. In this case, instance variable access is very fast.

- Global cache.

  A global cache is used by the instance variable access functions when the local cache fails. This global cache is a fixed size table of instance variable pairs. Looking in this cache for a given class is typically faster than computing the instance variable index.

You should be aware that instance variable access is optimized to be faster than accessing the properties of instance variables. Also, be aware that when instances are first created, the data for an instance variable may need to be found by performing a lookup through the class hierarchy. If the lookup goes through several classes, this can be slow. By guaranteeing that the instance variable data is stored in the instance, this lookup delay can be avoided.

The following macros are used to access instance variables. They are mentioned here to point out that calls to **GetValue** and **PutValue** could result in the compilation of any one of several different functions.

(**GetValue** *self varName* &OPTIONAL *propName*) [Macro]

Purpose/Behavior: Compiles to a call to one of the functions **Cached-GetIVValue**, **Cached-GetIVProp**, **GetIVValue**, or **GetIVProp**. The particular function depends on details of the arguments to **GetValue**.

Arguments:

self	A class or an instance.
varName	Instance or class variable name.
propName	Property name.

Returns: Used for side effect only.

(**PutValue** *self varName value* &OPTIONAL *propName*) [Macro]

Purpose/Behavior: Compiles to a call to one of the functions **Cached-PutIVValue**, **Cached-PutIVProp**, **PutIVValue**, or **PutIVProp**. The particular function depends on details of the arguments to **PutValue**.

Arguments:

self	A class or an instance.
varName	Instance or class variable name.
value	The new value for *varName* or *propName*.
propName	Property name.

Returns: Used for side effect only.

15.3 METHOD LOOKUP

15.3 Method Lookup

LOOPS uses two layers of caching to speed the method lookup:

- Local cache

  Method lookup from compiled code uses a local cache when the selector can be determined at compile time. This cache remembers the class of *self* and the computed method the last time this message was sent. If the class of *self* on the next pass through the code matches the stored value, then the method is used. In this case, method lookup is very fast.

- Global cache

  A global cache is used by the method lookup functions when the local cache fails. This global cache is a fixed size table of class / selector / method triples. Looking in this cache for a given class and selector is typically faster than searching the class hierarchy for the appropriate method.

15.4 CACHE CLEARING

15.4 CACHE CLEARING

15.4 Cache Clearing

Code that directly manipulates the structure of LOOPS objects sometimes needs to invalidate the caches used for instance variable access and message sending.

The following functions can be used to clear these caches if you suspect that they might be invalid.

(**ClearAllCaches**) [Function]

Purpose/Behavior: Clears all LOOPS and Interlisp-D runtime caches. This includes local and global instance variable access caches, local and global method lookup caches, and the system CLISP translations hash array.

Returns: NIL

[This page intentionally left blank]