25 Random Number Generator Module (rand.hhf)

The rand.hhf header file contains definitions for HLA's random number generators. These functions provide a variety of pseudo-random number generators and support routines.

25.1 The Random Module

To use the random number generator functions in your application, you will need to include one of the following statements at the beginning of your HLA application:

#include("rand.hhf")

or

#include("stdlib.hhf")

25.2 The Random Number Generators

`procedure rand.randomize;`

This function "randomizes" the seed used by the random number generators. If you call *rand.randomize*, the random number generators should begin generating a sequence starting at a random point in the normal sequence put out by the random number generator.

The randomization function is based on the number of CPU clock cycles that have occurred since the CPU was last powered up. This function uses the Pentium's RDTSC instruction, hence you should only call this function on machines that have this instruction available (Intel Pentium and later as well as other manufacturer's CPUs that have this instruction).

Because of the nature of the RDTSC instruction, you should not call *rand.randomize* frequently or you will compromise the quality of the random numbers (indeed, it's generally not a good idea to "randomize" a random number generator more than once per program invocation). Similarly, you should avoid calling this function from a fixed script after power-on since that may also degrade the quality of the randomization. (These two suggestions are only important to those who are extremely concerned about the quality of the randomness of the generated numbers).

```
HLA high-level calling sequence example:

  rand.randomize();


HLA low-level calling sequence example:

  call rand.randomize;
```

`procedure rand.uniform; @returns( "eax" );`

This function generates a new random number on each call. This function returns a new 32-bit value in the EAX register on each call (bit 31 is randomly set, you may choose to interpret this value as a signed or unsigned integer). This function generates uniformly-distributed random numbers.

This functions uses an algorithm from Knuth's *The Art of Computer Programming* for details (and limitations) on this type of random number generator.

```
HLA high-level calling sequence example:

  rand.uniform();
  mov( eax, randomValue );


HLA low-level calling sequence example:
    call rand.uniform;
    mov( eax, randomValue );
```

procedure rand.urange(startRange:int32; endRange:int32); @returns("eax");

This function generates a uniformly distributed random number in the range *"startRange..endRange"* (inclusive). This function generates its random numbers using the *rand.uniform* function. This function returns the value in the EAX register. This uses the same random-number generator algorithm that *rand.uniform* uses.

```
  HLA high-level calling sequence example:

    rand.urange( minValue, maxValue );
    mov( eax, randomValue );


  HLA low-level calling sequence example:

    push( minValue );
    push( maxValue );
    call rand.urange;
    mov( eax, randomValue );
```

procedure rand.random; @returns("eax");

This function generates a uniformly distributed random number using a linear congruential random number generator. This function returns a new 32-bit value in the EAX register on each call (bit 31 is randomly set, you may choose to interpret this value as a signed or unsigned integer). This function generates uniformly-distributed random numbers.

See Knuth's *The Art of Computer Programming* for details (and limitations) on linear congruential random number generators.

```
  HLA high-level calling sequence example:

    rand.random();
    mov( eax, randomValue );


  HLA low-level calling sequence example:

    call rand.random;
    mov( eax, randomValue );
```

procedure rand.range(startRange:int32; endRange:int32); @returns("eax");

This function generates a uniformly distributed random number in the range *"startRange..endRange"* (inclusive) using a linear congruential random number generator. This function returns the value in the EAX register. This uses the same random-number generator algorithm that *rand.random* uses.

```
  HLA high-level calling sequence example:

    rand.range( minValue, maxValue );
    mov( eax, randomValue );


  HLA low-level calling sequence example:
```

```
push( minValue );
push( maxValue );
call rand.range;
mov( eax, randomValue );
```

iterator rand.deal(count:uns32);

The *rand.deal* iterator returns a sequence of *count* unique randomly arranged values in the range *0..count-1*. Therefore, it returns all values in the range *0..count-1*, but in a random order.

Since *rand.deal* is an iterator, you must only use it within a FOREACH loop, e.g.,

```
foreach deal( 52 ) do

    << EAX contains a value in the range 0..51 here>>

endfor;
```

This function uses the *rand.uniform* function to randomly arrange the values.