Threading Issues

A Natural process on Windows, UNIX and OpenVMS always contains only one thread that executes Natural code. Thus in an interactively started Natural session, it can never occur that several threads try to execute Natural code in parallel. The situation is different when a client program that runs several threads in parallel uses the Natural Native Interface.

The Natural Native Interface can be used by multithreaded applications. The interface functions are thread safe. As long as a given thread T is executing one of the interface functions, other threads of the same process that call one of the interface functions are blocked until T has left the interface function. Effectively the parallel executing threads of the process are serialized as far as the usage of the interface functions is concerned. It is not necessary to serialize interface access among the threads of different processes, because each different process that uses the NNI runs its own Natural session.

The calling application can also control the multithreaded access to the NNI explicitly. This can make sense if a thread wants to execute a series of NNI calls without being interrupted by another thread. To achieve this, the thread calls `nni_enter`, which lets the thread wait until all other threads have left the NNI. Then the thread does its work and calls NNI functions at will. After having finished its work, the thread calls `nni_leave` to allow other threads to access the NNI.

A multithreaded application that uses the NNI must follow these rules:

- The functions `nni_initialize` and `nni_uninitialize` must be called at least once per process.

- The function `nni_uninitialize` must be called on the same thread as the corresponding call to `nni_initialize`.

- The function `nni_uninitialize` must not be called before the last thread that uses the NNI has terminated.