Interface Functions

For detailed information on the individual functions, use the selection box on the right.

---

nni_get_interface

Syntax

```
int nni_get_interface( int iVersion, void** ppnni_func );
```

The function returns an instance of the Natural Native Interface.

An application calls this function after having retrieved and loaded the interface library with platform depending system calls. The function returns a pointer to a structure that contains function pointers to the individual interface functions. The functions returned in the structure may differ between interface versions.

Instead of a specific interface version, the caller can also specify the constant `NNI_VERSION_CURR`, which always refers to the most recent interface version. The interface version number belonging to a given Natural version is defined in the header file *natni.h* that is delivered with that version. In Natural Version *n.n*, the interface version number is defined as `NNI_VERSION_`*nn*. `NNI_VERSION_CURR` is also defined as `NNI_VERSION_`*nn*. If the Natural version against which the function is called does not support the requested interface version, the error code `NNI_RC VERSION_ERROR` is returned. Otherwise the return code is `NNI_RC_OK`.

The pointer returned by the function represents one instance of the interface. In order to use this interface instance, the application holds on to that pointer and passes it to subsequent interface calls.

Usually the application will subsequently initialize a Natural session by calling `nni_initialize` on the given instance. After the application has finished using that Natural session, it calls `nni_uninitialize` on that instance. After that it can initialize a different Natural session on the same interface instance. After the application has finished using the interface instance entirely, it calls `nni_free_interface` on that instance.

Parameters

Parameter	Meaning
`iVersion`	Interface version number. (`NNI_VERSION_`*nn* or `NNI_VERSION_CURR`).
`ppnni_func`	Points to an NNI interface instance on return.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
NNI_RC_OK	
NNI_RC_PARM_ERROR	
NNI_RC_VERSION_ERROR	

nni_free_interface

Syntax

```
int nni_free_interface(void* pnni_func);
```

An application calls this function after it has finished using the interface instance and has uninitialized the Natural session it hosts. The function frees the resources occupied by that interface instance.

Parameters

Parameter	Meaning
pnni_func	Pointer to an NNI interface instance.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
NNI_RC_OK	

nni_initialize

Syntax

```
int nni_initialize(void* pnni_func, const char* szCmdLine, void*, void*);
```

The function initializes a Natural session with a given command line. The syntax and semantics of the command line is the same as when Natural is started interactively. If a Natural session has already been initialized on the given interface instance, that session is implicitly uninitialized before the new session is initialized.

The command line must be specified in the way that the Natural initialization can be completed without user interaction. This means especially that if a program is passed on the stack or a startup program is specified, that program must not perform an `INPUT` statement that is not satisfied from the stack. Otherwise the subsequent behavior of the Natural session is undetermined.

The Natural session is initialized as batch session and in server mode. This means that the usage of certain statements and commands in the executed Natural modules is restricted. These restrictions and error conditions are the same as documented in the section *Using Statements and Commands in a NaturalX Server Environment* of the *Operations* documentation.

When initalizing a Natural session under Natural Security, the command line must contain a `LOGON` command to a freely chosen default library under which the session will be started, and an appropriate user ID and password.

Example:

```
int iRes =
pnni_func->nni_initialize( pnni_func, "STACK=(LOGON,MYLIB,MYUSER,MYPASS)", 0, 0);
```

If the application later calls `nni_logon` to a different library with a different user ID and afterwards calls `nni_logoff`, the Natural session will be reset to the library and user ID that was passed during `nni_initialize`.

Parameters

Parameter	Meaning
`pnni_func`	Pointer to an NNI interface instance.
`szCmdLine`	Natural command line. May be a null pointer.
`void*`	For future use. Must be a null pointer.
`void*`	For future use. Must be a null pointer.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
`NNI_RC_OK`	
`NNI_RC_PARM_ERROR`	
rc, where *rc* < `NNI_RC_SERR_OFFSET`	Natural startup error. The real Natural startup error number as documented in *Natural Startup Errors* (which is part of the *Operations* documentation) can be determined by the following calculation: *startup-error-nr* = - (*rc* - `NNI_RC_SERR_OFFSET`) Warnings that occur during session initialization are ignored.
> 0	Natural error number.

nni_is_initialized

Syntax

```
int nni_is_initialized( void* pnni_func, int* piIsInit );
```

The function checks if the interface instance contains an initialized Natural session.

Parameters

Parameter	Meaning
pnni_func	Pointer to an NNI interface instance.
piIsInit	Returns 0, if no Natural session is initialized, a non-zero value otherwise.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
NNI_RC_OK	
NNI_RC_PARM_ERROR	

nni_uninitialize

Syntax

```
int nni_uninitialize(void* pnni_func);
```

The function uninitializes the Natural session hosted by the given interface instance.

Parameters

Parameter	Meaning
pnni_func	Pointer to an NNI interface instance.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
NNI_RC_OK	

nni_enter

Syntax

```
int nni_enter(void* pnni_func);
```

The function lets the current thread wait for exclusive access to the interface instance and the Natural session it hosts. A thread calls this function if it wants to issue a series of interface calls that may not be interrupted by other threads. The thread releases the eclusive access to the interface instance by calling nni_leave.

Parameters

Parameter	Meaning
pnni_func	Pointer to an NNI interface instance.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
NNI_RC_OK	

nni_try_enter

Syntax

```
int nni_try_enter(void* pnni_func);
```

The function behaves like nni_enter except that it does not block the thread and instead always returns immediately. If a different thread already has exclusive access to the interface instance, the function returns NNI_RC_LOCKED.

Parameters

Parameter	Meaning
pnni_func	Pointer to an NNI interface instance.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
NNI_RC_OK	
NNI_RC_LOCKED	

nni_leave

Syntax

```
int nni_leave(void* pnni_func);
```

The function releases exclusive access to the interface instance and allows other threads to access that instance and the Natural session it hosts.

Parameters

Parameter	Meaning
pnni_func	Pointer to an NNI interface instance.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
NNI_RC_OK	

nni_logon

Syntax

```
int nni_logon(void* pnni_func, const char* szLibrary, const char* szUser, const char* szPassword);
```

The function performs a LOGON to the specified Natural library.

Parameters

Parameter	Meaning
pnni_func	Pointer to an NNI interface instance.
szLibrary	Name of the Natural library.
szUser	Name of the Natural user. May be a null pointer, if the Natural session is not running under Natural Security or if AUTO=ON was used during initialization.
szPassword	Password of that user. May be a null pointer, if the Natural session is not running under Natural Security or if AUTO=ON was used during initialization..

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
NNI_RC_OK	
NNI_RC_NOT_INIT	
NNI_RC_PARM_ERROR	
> 0	Natural error number.

nni_logoff

Syntax

```
int nni_logoff(void* pnni_func);
```

The function performs a `LOGOFF` from the current Natural library. This corresponds to a `LOGON` to the previously active library and user ID.

Parameters

Parameter	Meaning
`pnni_func`	Pointer to an NNI interface instance.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
`NNI_RC_OK`	
`NNI_RC_NOT_INIT`	
`NNI_RC_PARM_ERROR`	
> 0	Natural error number.

nni_callnat

Syntax

```
int nni_callnat(void* pnni_func, const char* szName, int iParm, struct parameter_description* rgDesc, struct natural_exception* pExcep);
```

The function calls a Natural subprogram.

The function receives its parameters as an array of `parameter_description` structures. The caller creates these structures using NNI functions in the following way:

- Use one the functions `create_parm` or `create_module_parm` to create an appropriate parameter set for the subprogram.
- If you have used `create_parm`, use the functions `init_parm_*` to initialize each parameter to the appropriate Natural data format. If you have used `create_module_parm`, the parameters are already initialized to the appropriate Natural data format.
- Assign a value to each parameter, using one the functions `nni_put_parm` or `nni_put_parm_array`.
- Call `nni_get_parm` on each parameter in the set. This fills the `parameter_description` structures.

- Pass the array of `parameter_description` structures to the function `nni_callnat`.
- After the call has been executed, extract the modified parameter values from the parameter set using the function `nni_get_parm` or `nni_get_parm_array`.

Parameters

Parameter	Meaning
`pnni_func`	Pointer to an NNI interface instance.
`szName`	Name of the Natural subprogram.
`iParm`	Number of parameters. Indicates the number of occurrences of the array `rgDesc`.
`rgDesc`	An array of `parm_description` structures containing the parameters for the subprogram. If the subprogram does not expect parameters, the caller passes a null pointer.
`pExcep`	Pointer to a `natural_exception` structure. If a Natural error occurs during execution of the subprogram,this structure is filled with Natural error information. The caller may specify a null pointer. In this case no extended error information is returned.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
`NNI_RC_OK`	
`NNI_RC_NOT_INIT`	
`NNI_RC_PARM_ERROR`	
`NNI_RC_NO_MEMORY`	
> 0	Natural error number.

nni_function

Syntax

```
int nni_callnat(void* pnni_func, const char* szName, int iParm, struct parameter_description* rgDesc, struct natural_exception* pExcep);
```

The function calls a Natural subprogram.

The function receives its parameters as an array of `parameter_description` structures. The caller creates these structures using NNI functions in the following way:

- Use one the functions `create_parm` or `create_module_parm` to create an appropriate parameter set for the subprogram.
- If you have used `create_parm`, use the functions `init_parm_*` to initialize each parameter to the appropriate Natural data format. If you have used `create_module_parm`, the parameters are already initialized to the appropriate Natural data format.

- Assign a value to each parameter, using one the functions `nni_put_parm` or `nni_put_parm_array`.
- Call `nni_get_parm` on each parameter in the set. This fills the `parameter_description` structures.
- Pass the array of `parameter_description` structures to the function `nni_callnat`.
- After the call has been executed, extract the modified parameter values from the parameter set using the function `nni_get_parm` or `nni_get_parm_array`.

Parameters

Parameter	Meaning
`pnni_func`	Pointer to an NNI interface instance.
`szName`	Name of the Natural subprogram.
`iParm`	Number of parameters. Indicates the number of occurrences of the array `rgDesc`.
`rgDesc`	An array of `parm_description` structures containing the parameters for the subprogram. If the subprogram does not expect parameters, the caller passes a null pointer.
`pExcep`	Pointer to a `natural_exception` structure. If a Natural error occurs during execution of the subprogram,this structure is filled with Natural error information. The caller may specify a null pointer. In this case no extended error information is returned.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
`NNI_RC_OK`	
`NNI_RC_NOT_INIT`	
`NNI_RC_PARM_ERROR`	
`NNI_RC_NO_MEMORY`	
> 0	Natural error number.

nni_create_object

Syntax

```
int nni_create_object(void* pnni_func, const char* szName, int iParm, struct parameter_description* rgDesc, struct natural_exception* pExcep);
```

Creates a Natural object (an instance of a Natural class).

The function receives its parameters as a one-element array of `parameter_description` structures. The caller creates the structures using NNI functions in the following way:

- Use the function `nni_create_parm` to create parameter set with one element.
- Use the function `nni_init_parm_s` to initialize the parameter with the type `HANDLE OF OBJECT`.
- Call `nni_get_parm_info` on this parameter. This fills the `parameter_description` structure.
- Pass the `parameter_description` structure to the function `nni_create_object`.
- After the call has been executed, extract the modified parameter value from the parameter set using one the function `nni_get_parm`.

The parameters passed in `rgDesc` have the following meaning:

- The first (and only) parameter must be initialized with the data type `HANDLE OF OBJECT` and contains on return the Natural object handle of the newly created object.

Parameters

Parameter	Meaning
`pnni_func`	Pointer to an NNI interface instance.
`szName`	Name of the class.
`iParm`	Number of parameters. Indicates the number of occurrences of the array `rgDesc`.
`rgDesc`	An array of `parm_description` structures containing the parameters for the object creation. The caller always passes one parameter, which will contain the object handle on return.
`pExcep`	Pointer to a `natural_exception` structure. If a Natural error occurs during object creation, this structure is filled with Natural error information. The caller may specify a null pointer. In this case no extended exception information is returned.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
`NNI_RC_OK`	
`NNI_RC_NOT_INIT`	
`NNI_RC_PARM_ERROR`	
`NNI_RC_NO_MEMORY`	
> 0	Natural error number.

nni_send_method

Syntax

```
int nni_send_method(void* pnni_func, const char* szName, int iParm, struct parameter_description* rgDesc, struct natural_exception* pExcep);
```

Sends a method call to a Natural object (an instance of a Natural class).

The function receives its parameters as an array of `parameter_description` structures. The caller creates these structures using NNI functions in the following way:

- Use the function `nni_create_parm` or `nni_create_method_parm` to create a matching parameter set.
- If you have used `create_parm`, use the functions `init_parm_*` to initialize each parameter to the appropriate Natural data format. If you have used `nni_create_method_parm`, the parameters are already initialized to the appropriate Natural data format.
- Assign a value to each parameter using one the functions `nni_put_parm` or `nni_put_parm_array`.
- Call `nni_get_parm_info` on each parameter in the set. This fills the `parameter_description` structures.
- Pass the array of `parameter_description` structures to the function `nni_send_method`.
- After the call has been executed, extract the modified parameter values from the parameter set using one of the `nni_get_parm` functions.

The parameters passed in `rgDesc` have the following meaning:

- The first parameter contains the object handle.
- The second parameter must be initialized to the data type of the method return value. If the method does not have a return value, the second parameter remains not initialized. On return from the method call, this parameter contains the return value of the method.
- The remaining parameters are the method parameters.

Parameters

Parameter	Meaning
pnni_func	Pointer to an NNI interface instance.
szName	Name of the method.
iParm	Number of parameters. Indicates the number of occurrences of the array rgDesc. This is always 2 + the number of method parameters.
rgDesc	An array of parm_description structures containing the parameters for the method. If the method does not expect parameters, the caller still passes two parameters, the first for the object handle and the second for the return value.
pExcep	Pointer to a natural_exception structure. If a Natural error occurs during execution of the method, this structure is filled with Natural error information. The caller may specify a null pointer. In this case no extended exception information is returned.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
NNI_RC_OK	
NNI_RC_NOT_INIT	
NNI_RC_PARM_ERROR	
NNI_RC_NO_MEMORY	
> 0	Natural error number.

nni_get_property

Syntax

```
int nni_get_property(void* pnni_func, const char* szName, int iParm, struct parameter_description* rgDesc, struct natural_exception* pExcep);
```

Retrieves a property value of a Natural object (an instance of a Natural class).

The function receives its parameters as an array of parameter_description structures. The caller creates these structures using NNI functions in the following way:

- Use the function nni_create_parm or nni_create_method_parm to create a matching parameter set.
- If you have used create_parm, use the functions init_parm_* to initialize each parameter to the appropriate Natural data format. If you have used create_method_parm, the parameters are already initialized to the appropriate Natural data format.
- Assign a value to each parameter using one the functions nni_put_parm or nni_put_parm_array.

- Call `nni_get_parm_info` on each parameter in the set. This fills the `parameter_description` structures.
- Pass the array of `parameter_description` structures to the function `nni_send_method`.
- After the call has been executed, extract the modified parameter values from the parameter set using one of the `nni_get_parm` functions.

The parameters passed in `rgDesc` have the following meaning:

- The first parameter contains the object handle.
- The second parameter is initialized to the data type of the property. On return from the property access, this parameter contains the property value.

Parameters

Parameter	Meaning
`pnni_func`	Pointer to an NNI interface instance.
`szName`	Name of the property.
`iParm`	Number of parameters. Indicates the number of occurrences of the array `rgDesc`. This is always 2.
`rgDesc`	An array of `parm_description` structures containing the parameters for the property access. The caller always passes two parameters, the first for the object handle and the second for the returned property value.
`pExcep`	Pointer to a `natural_exception` structure. If a Natural error occurs during property access, this structure is filled with Natural error information. The caller may specify a null pointer. In this case no extended exception information is returned.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
`NNI_RC_OK`	
`NNI_RC_NOT_INIT`	
`NNI_RC_PARM_ERROR`	
`NNI_RC_NO_MEMORY`	
> 0	Natural error number.

nni_set_property

Syntax

```
int nni_set_property(void* pnni_func, const char* szName, int iParm, struct parameter_description* rgDesc, struct natural_exception* pExcep);
```

Assigns a property value to a Natural object (an instance of a Natural class).

The function receives its parameters as an array of `parameter_description` structures. The caller creates these structures using NNI functions in the following way:

- Use the function `nni_create_parm` or `nni_create_prop_parm` to create a matching parameter set.
- If you have used `create_parm`, use the functions `init_parm_*` to initialize each parameter to the appropriate Natural data format. If you have used `create_prop_parm`, the parameters are already initialized to the appropriate Natural data format. Assign a value to each parameter using one of the `nni_put_parm` functions.
- Assign a value to each parameter using one the functions `nni_put_parm` or `nni_put_parm_array`.
- Call `nni_get_parm_info` on each parameter in the set. This fills the `parameter_description` structures.
- Pass the array of `parameter_description` structures to the function `nni_set_property`.

The parameters passed in `rgDesc` have the following meaning:

- The first parameter contains the object handle.
- The second parameter contains the property value.

Parameters

Parameter	Meaning
`pnni_func`	Pointer to an NNI interface instance.
`szName`	Name of the property.
`iParm`	Number of parameters. Indicates the number of occurrences of the array `rgDesc`. This is always 2.
`rgDesc`	An array of `parm_description` structures containing the parameters for the property access. The caller always passes two parameters, the first for the object handle and the second for the property value.
`pExcep`	Pointer to a `natural_exception` structure. If a Natural error occurs during property access, this structure is filled with Natural error information. The caller may specify a null pointer. In this case no extended exception information is returned.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
NNI_RC_OK	
NNI_RC_NOT_INIT	
NNI_RC_PARM_ERROR	
NNI_RC_NO_MEMORY	
> 0	Natural error number.

nni_delete_object

Syntax

```
int nni_delete_object(void* pnni_func, int iParm, struct parameter_description* rgDesc, struct natural_exception* pExcep);
```

Deletes a Natural object (an instance of a Natural class) created with `nni_create_object`.

The function receives its parameters as a one-element array of `parameter_description` structures. The caller creates the structures using NNI functions in the following way:

- Use the function `nni_create_parm` to create parameter set with one element.
- Use the function `nni_init_parm_s` to initialize the parameter with the type `HANDLE OF OBJECT`.
- Assign a value to the parameter using one the functions `nni_put_parm`.
- Call `nni_get_parm_info` on this parameter. This fills the `parameter_description` structure.
- Pass the `parameter_description` structure to the function `nni_delete_object`.

The parameters passed in `rgDesc` have the following meaning:

- The first (and only) parameter must be initialized with the data type `HANDLE OF OBJECT` and contains the Natural object handle of the object to be deleted.

Parameters

Parameter	Meaning
pnni_func	Pointer to an NNI interface instance.
szName	Name of the class.
iParm	Number of parameters. Indicates the number of occurrences of the array rgDesc. This is always 1.
rgDesc	An array of parm_description structures containing the parameters for the object creation. The caller always passes one parameter, which contains the object handle.
pExcep	Pointer to a natural_exception structure. If a Natural error occurs during object creation, this structure is filled with Natural error information. The caller may specify a null pointer. In this case no extended exception information is returned.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
NNI_RC_OK	
NNI_RC_NOT_INIT	
NNI_RC_PARM_ERROR	
NNI_RC_NO_MEMORY	
> 0	Natural error number.

nni_create_parm

Syntax

```
int nni_create_parm(void* pnni_func, int iParm, void** pparmhandle);
```

Creates a set of parameters that can be passed to a Natural module.

The parameters contained in the set are not yet initialized to specific Natural data types. Before using the parameter set in a call to nni_callnat, nni_create_object, nni_send_method, nni_set_property or nni_get_property:

- Initialize each parameter to the required Natural data type using one of the functions nni_init_parm_s, nni_init_parm_sa, nni_init_parm_d or nni_init_parm_da.
- Assign a value to each parameter using one of the functions nni_put_parm or nni_put_parm_array.
- Turn each parameter into a parm_description structure using the function nni_get_parm_info.

Parameters

Parameter	Meaning
pnni_func	Pointer to an NNI interface instance.
iParm	Requested number of parameters. The maxinum number of parameters is 32767.
pparmhandle	Points a to a pointer to a parameter set on return.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
NNI_RC_OK	
NNI_RC_NOT_INIT	
NNI_RC_PARM_ERROR	
NNI_RC_ILL_PNUM	
> 0	Natural error number.

nni_create_module_parm

Syntax

```
int nni_create_module_parm(void* pnni_func, char chType, const char* szName, void** pparmhandle);
```

Creates a set of parameters that can be used in a call to nni_callnat. The function enables an application to dynamically explore the signature of a callable Natural module.

The parameters contained in the returned set are already initialized to Natural data types according to the parameter data area of the specified module. Before using the parameter set in a call to nni_callnat:

- Assign a value to each parameter using one of the functions nni_put_parm or nni_put_parm_array.
- Turn each parameter into a parm_description structure using the function nni_get_parm_info.

Parameters

Parameter	Meaning
pnni_func	Pointer to an NNI interface instance.
chType	Type of the Natural module. Always "N" (for subprogram).
szName	Name of the Natural module.
pparmhandle	Points a to a pointer to a parameter set on return.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
NNI_RC_OK	
NNI_RC_NOT_INIT	
NNI_RC_PARM_ERROR	
> 0	Natural error number.

nni_create_method_parm

Syntax

```
int nni_create_method_parm( void* pnni_func, const char* szClass, const char* szMethod, void** pparmhandle );
```

Creates a set of parameters that can be used in a call to `nni_send_method`. The function enables an application to dynamically explore the signature of a method of a Natural class.

The returned parameter set contains not only the method parameters, but also the other parameters required by `nni_send_method`. This means: If the method has *n* parameters, the parameter set contains *n* + 2 parameters.

- The first parameter in the set is initialized to the data type `HANDLE OF OBJECT`.
- The second parameter in the set is initialized to the data type of the method return value. If the method does not have a return value, the second parameter is not initialized.
- The remaining parameters in the set are initialized to the data types of the method parameters.

Before using the parameter set in a call to `nni_send_method`:

- Assign a value to each parameter using one of the functions `nni_put_parm` or `nni_put_parm_array`.
- Turn each parameter into a `parm_description` structure using the function `nni_get_parm_info`.

Parameters

Parameter	Meaning
pnni_func	Pointer to an NNI interface instance.
szClass	Name of the Natural class.
szMethod	Name of the Natural method.
pparmhandle	Points a to a pointer to a parameter set on return.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
NNI_RC_OK	
NNI_RC_NOT_INIT	
NNI_RC_PARM_ERROR	
> 0	Natural error number.

nni_create_prop_parm

Syntax

```
int nni_create_prop_parm(void* pnni_func, const char* szClass, const char* szProp,void** pparmhandle);
```

Creates a set of parameters that can be used in a call to `nni_get_property` or `nni_set_property`. The returned parameter set contains all parameters required by `nni_get_property` or `nni_set_property`. The function enables an application to determine the data type of a property of a Natural class.

- The first parameter in the set is initialized to the data type `HANDLE OF OBJECT`.
- The second parameter in the set is initialized to the data type of the property.

Before using the parameter set in a call to `nni_get_property` or `nni_set_property`:

- Assign a value to each parameter using one of the functions `nni_put_parm` or `nni_put_parm_array`.
- Turn each parameter into a `parm_description` structure using the function `nni_get_parm_info`.

Parameters

Parameter	Meaning
pnni_func	Pointer to an NNI interface instance.
szClass	Name of the Natural class.
szProp	Name of the Natural property.
pparmhandle	Points a to a pointer to a parameter set on return.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
NNI_RC_OK	
NNI_RC_NOT_INIT	
NNI_RC_PARM_ERROR	
> 0	Natural error number.

nni_parm_count

Syntax

```
int nni_parm_count( void* pnni_func, void* parmhandle, int* piParm )
```

The function retrieves the number of parameters in a parameter set.

Parameters

Parameter	Meaning
pnni_func	Pointer to an NNI interface instance.
parmhandle	Pointer to a parameter set.
piParm	Returns the number of parameters in the parameter set.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
NNI_RC_OK	
NNI_RC_NOT_INIT	
NNI_RC_PARM_ERROR	

nni_init_parm_s

Syntax

```
int nni_init_parm_s(void* pnni_func, int iParm, void* parmhandle, char chFormat, int iLength, int iPrecision, int iFlags);
```

Initializes a parameter in a parameter set to a static data type.

Parameters

Parameter	Meaning
pnni_func	Pointer to an NNI interface instance.
iParm	Index of the parameter. The first parameter in the set has the index 0.
parmhandle	Pointer to a parameter set.
chFormat	Natural data type of the parameter.
iLength	Natural length of the parameter.
iPrecision	Number of decimal places (NNI_TYPE_NUM and NNI_TYPE_PACK only).
iFlags	Parameter flags. The following flags can be used: NNI_FLG_PROTECTED

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
NNI_RC_OK	
NNI_RC_NOT_INIT	
NNI_RC_PARM_ERROR	
NNI_RC_ILL_PNUM	
NNI_RC_NO_MEMORY	
NNI_RC_BAD_FORMAT	
NNI_RC_BAD_LENGTH	

nni_init_parm_sa

Syntax

```
int nni_init_parm_sa (void* pnni_func, int iParm, void* parmhandle, char chFormat, int iLength, int iPrecision, int iDim, int* rgiOcc, int iFlags);
```

Initializes a parameter in a parameter set to an array of a static data type.

Parameters

Parameter	Meaning
pnni_func	Pointer to an NNI interface instance.
iParm	Index of the parameter. The first parameter in the set has the index 0.
parmhandle	Pointer to a parameter set.
chFormat	Natural data type of the parameter.
iLength	Natural length of the parameter.
iPrecision	Number of decimal places (NNI_TYPE_NUM and NNI_TYPE_PACK only).
iDim	Array dimension of the parameter.
rgiOcc	Three dimensional array of int values, indicating the occurrence count for each dimension. The occurrence count for unused dimensions must be specified as 0.
iFlags	Parameter flags. The following flags can be used: NNI_FLG_PROTECTED NNI_FLG_LBVAR_0 NNI_FLG_UBVAR_0 NNI_FLG_LBVAR_1 NNI_FLG_UBVAR_1 NNI_FLG_LBVAR_2 NNI_FLG_UBVAR_2 If one of the NNI_FLG_*VAR* flags is set, the array is an x-array. In each dimension only the lower bound or the upper bound (not both) can be variable. Therefore for instance the flag IF4_FLG_LBVAR_0 may not be combined with IF4_FLG_UBVAR_0.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
NNI_RC_OK	
NNI_RC_NOT_INIT	
NNI_RC_PARM_ERROR	
NNI_RC_ILL_PNUM	
NNI_RC_NO_MEMORY	
NNI_RC_BAD_FORMAT	
NNI_RC_BAD_LENGTH	
NNI_RC_BAD_DIM	
NNI_RC_BAD_BOUNDS	

nni_init_parm_d

Syntax

```
int nni_init_parm_d(void* pnni_func, int iParm, void* parmhandle, char chFormat, int iFlags);
```

Initializes a parameter in a parameter set to a dynamic data type.

Parameters

Parameter	Meaning
pnni_func	Pointer to an NNI interface instance.
iParm	Index of the parameter. The first parameter in the set has the index 0.
parmhandle	Pointer to a parameter set.
chFormat	Natural data type of the parameter (NNI_TYPE_ALPHA or NNI_TYPE_BIN).
iFlags	Parameter flags. The following flags can be used: NNI_FLG_PROTECTED

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
NNI_RC_OK	
NNI_RC_NOT_INIT	
NNI_RC_PARM_ERROR	
NNI_RC_ILL_PNUM	
NNI_RC_NO_MEMORY	
NNI_RC_BAD_FORMAT	

nni_init_parm_da

Syntax

```
int nni_init_parm_da (void* pnni_func, int iParm, void* parmhandle, char chFormat, int iDim, int* rgiOcc, int iFlags);
```

Initializes a parameter in a parameter set to an array of a dynamic data type.

Parameters

Parameter	Meaning
pnni_func	Pointer to an NNI interface instance.
iParm	Index of the parameter. The first parameter in the set has the index 0.
parmhandle	Pointer to a parameter set.
chFormat	Natural data type of the parameter (NNI_TYPE_ALPHA or NNI_TYPE_BIN).
iDim	Array dimension of the parameter.
rgiOcc	Three dimensional array of int values, indicating the occurrence count for each dimension. The occurrence count for unused dimensions must be specified as 0.
iFlags	Parameter flags. The following flags can be used: NNI_FLG_PROTECTED NNI_FLG_LBVAR_0 NNI_FLG_UBVAR_0 NNI_FLG_LBVAR_1 NNI_FLG_UBVAR_1 NNI_FLG_LBVAR_2 NNI_FLG_UBVAR_2 If one of the NNI_FLG_*VAR* flags is set, the array is an x-array. In each dimension only the lower bound or the upper bound (not both) can be variable. Therefore for instance the flag IF4_FLG_LBVAR_0 may not be combined with IF4_FLG_UBVAR_0.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
NNI_RC_OK	
NNI_RC_NOT_INIT	
NNI_RC_PARM_ERROR	
NNI_RC_ILL_PNUM	
NNI_RC_NO_MEMORY	
NNI_RC_BAD_FORMAT	
NNI_RC_BAD_DIM	
NNI_RC_BAD_BOUNDS	

nni_get_parm_info

Syntax

```
int nni_get_parm_info (void* pnni_func, int iParm, void* parmhandle, struct parameter_description* pDesc);
```

Returns detailed information about a specific parameter in a parameter set.

Parameters

Parameter	Meaning
`pnni_func`	Pointer to an NNI interface instance.
`iParm`	Index of the parameter. The first parameter in the set has the index 0.
`parmhandle`	Pointer to a parameter set.
`pDesc`	Parameter description structure.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
`NNI_RC_OK`	
`NNI_RC_NOT_INIT`	
`NNI_RC_PARM_ERROR`	
`NNI_RC_ILL_PNUM`	

nni_get_parm

Syntax

```
int nni_get_parm(void* pnni_func, int iParm, void* parmhandle, int iBufferLength, void* pBuffer);
```

Returns the value of a specific parameter in a parameter set. The value is returned in the buffer at the address specified in `pBuffer`, with the size specified in `iBufferLength`. On successful return, the buffer contains the data in Natural internal format. See *Natural Data Types* on how to interpret the contents of the buffer.

If the length of the parameter according to the Natural data type is greater than `iBufferLength`, Natural truncates the data to the given length and returns the code `NNI_RC_DATA_TRUNC`. The caller can use the function `nni_get_parm_info` to request the length of the parameter value in advance.

If the length of the parameter according to the Natural data type is smaller than `iBufferLength`, Natural fills the buffer according to the length of the parameter and returns the length of the copied data in the return code.

If the parameter is an array, the function returns the whole array in the buffer. This makes sense only for fixed size arrays of fixed size elements, because in other cases the caller cannot interpret the contents of the buffer. In order to retrieve an individual occurrence of an arbitrary array use the function `nni_get_parm_array`.

If no memory of the size specified in `iBufferLength` is allocated at the adress specified in `pBuffer`, the results of the operation are unpredictable. Natural only checks that `pBuffer` is not null.

Parameters

Parameter	Meaning
`pnni_func`	Pointer to an NNI interface instance.
`iParm`	Index of the parameter. The first parameter in the set has the index 0.
`parmhandle`	Pointer to a parameter set.
`iBufferLength`	Length of the buffer specified in `pBuffer`.
`pBuffer`	Buffer in which the value is returned.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
`NNI_RC_OK`	
`NNI_RC_NOT_INIT`	
`NNI_RC_PARM_ERROR`	
`NNI_RC_ILL_PNUM`	
`NNI_RC_DATA_TRUNC`	
= *n*, where *n* > 0	Successful operation, but only *n* bytes were returned in the buffer.

nni_get_parm_array

Syntax

```
int nni_get_parm_array(void* pnni_func, int parmnum, void* parmhandle, int iBufferLength, void* pBuffer, int* rgiInd);
```

Returns the value of a specific occurrence of a specific array parameter in a parameter set. The only difference to `nni_get_parm` is that array indices can be specified. The indices for unused dimensions must be specified as 0.

Parameters

Parameter	Meaning
`pnni_func`	Pointer to an NNI interface instance.
`iParm`	Index of the parameter. The first parameter in the set has the index 0.
`parmhandle`	Pointer to a parameter set.
`iBufferLength`	Length of the buffer specified in `pBuffer`.
`pBuffer`	Buffer in which the value is returned.
`rgiInd`	Three dimensional array of int values, indicating a specific array occurrence. The indices start with 0.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
`NNI_RC_OK`	
`NNI_RC_NOT_INIT`	
`NNI_RC_PARM_ERROR`	
`NNI_RC_ILL_PNUM`	
`NNI_RC_DATA_TRUNC`	
`NNI_RC_NOT_ARRAY`	
`NNI_RC_BAD_INDEX_0`	
`NNI_RC_BAD_INDEX_1`	
`NNI_RC_BAD_INDEX_2`	
= *n*, where *n* > 0	Successful operation, but only *n* bytes were returned.

nni_get_parm_array_length

Syntax

```
int nni_get_parm_array_length(void* pnni_func, int iParm, void* parmhandle, int* piLength, int* rgiInd);
```

Returns the length of a specific occurrence of a specific array parameter in a parameter set.

Parameters

Parameter	Meaning
pnni_func	Pointer to an NNI interface instance.
iParm	Index of the parameter. The first parameter in the set has the index 0.
parmhandle	Pointer to a parameter set.
piLength	Pointer to an int in which the length of the value is returned.
rgiInd	Three dimensional array of int values, indicating a specific array occurrence. The indices start with 0.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
NNI_RC_OK	
NNI_RC_ILL_PNUM	
NNI_RC_DATA_TRUNC	
NNI_RC_NOT_ARRAY	
NNI_RC_BAD_INDEX_0	
NNI_RC_BAD_INDEX_1	
NNI_RC_BAD_INDEX_2	

nni_put_parm

Syntax

```
int nni_put_parm(void* pnni_func, int iParm, void* parmhandle, int iBufferLength, const void* pBuffer);
```

Assigns a value to a specific parameter in a parameter set. The value is passed to the function in the buffer at the address specified in `pBuffer`, with the size specified in `iBufferLength`. See *Natural Data Types* on how to prepare the contents of the buffer.

If the length of the parameter according to the Natural data type is smaller than the given buffer length, the data will be truncated to the length of the parameter. The rest of the buffer will be ignored. If the length of the parameter according to the Natural data type is greater than the given buffer length, the data will copied only to the given buffer length, the rest of the parameter value stays unchanged. See *Natural Data Types* on the internal length of Natural data types.

If the parameter is a dynamic variable, it is automatically resized according to the given buffer length.

If the parameter is an array, the function expects the whole array in the buffer. This makes sense only for fixed size arrays of fixed size elements, because in other cases the caller cannot provide the correct contents of the buffer. In order to assign a value to an individual occurrence of an arbitrary array use the function `nni_put_parm_array`.

Parameters

Parameter	Meaning
`pnni_func`	Pointer to an NNI interface instance.
`iParm`	Index of the parameter. The first parameter in the set has the index 0.
`parmhandle`	Pointer to a parameter set.
`iBufferLength`	Length of the buffer specified in `pBuffer`.
`pBuffer`	Buffer in which the value is passed.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
`NNI_RC_OK`	
`NNI_RC_NOT_INIT`	
`NNI_RC_PARM_ERROR`	
`NNI_RC_ILL_PNUM`	
`NNI_RC_WRT_PROT`	
`NNI_RC_DATA_TRUNC`	
`NNI_RC_NO_MEMORY`	
= *n*, where *n* > 0	Successful operation, but only *n* bytes of the buffer were used.

nni_put_parm_array

Syntax

```
int nni_put_parm_array(void* pnni_func, int iParm, void* parmhandle, int iBufferLength, const void* pBuffer, int* rgiInd);
```

Assigns a value to a specific occurrence of a specific array parameter in a parameter set. The only difference to `nni_get_parm` is that array indices can be specified. The indices for unused dimensions must be specified as 0.

Parameters

Parameter	Meaning
`pnni_func`	Pointer to an NNI interface instance.
`iParm`	Index of the parameter. The first parameter in the set has the index 0.
`parmhandle`	Pointer to a parameter set.
`iBufferLength`	Length of the buffer specified in `pBuffer`.
`pBuffer`	Buffer in which the value is passed.
`rgiInd`	Three dimensional array of int values, indicating a specific array occurrence. The indices start with 0.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
`NNI_RC_OK`	
`NNI_RC_NOT_INIT`	
`NNI_RC_PARM_ERROR`	
`NNI_RC_ILL_PNUM`	
`NNI_RC_WRT_PROT`	
`NNI_RC_DATA_TRUNC`	
`NNI_RC_NO_MEMORY`	
`NNI_RC_NOT_ARRAY`	
`NNI_RC_BAD_INDEX_0`	
`NNI_RC_BAD_INDEX_1`	
`NNI_RC_BAD_INDEX_2`	
= *n*, where *n* > 0	Successful operation, but only *n* bytes of the buffer were used.

nni_resize_parm_array

Syntax

```
int nni_resize_parm_array(void* pnni_func, int iParm, void* parmhandle, int* rgiOcc);
```

Changes the occurrence count of a specific x-array parameter in a parameter set. For an *n*-dimensional array an occurrence count must be specified for all *n* dimensions. If the dimension of the array is less than 3, the value 0 must be specified for the not used dimensions.

The function tries to resize the occurrence count of each dimension either by changing the lower bound or the upper bound, whatever is appropriate for the given x-array.

Parameters

Parameter	Meaning
`pnni_func`	Pointer to an NNI interface instance.
`iParm`	Index of the parameter. The first parameter in the set has the index 0.
`parmhandle`	Pointer to a parameter set.
`rgiOcc`	Three dimensional array of int values, indicating the new occurrence count of the array.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
`NNI_RC_OK`	
`NNI_RC_NOT_INIT`	
`NNI_RC_PARM_ERROR`	
`NNI_RC_ILL_PNUM`	
`NNI_RC_WRT_PROT`	
`NNI_RC_DATA_TRUNC`	
`NNI_RC_NO_MEMORY`	
`NNI_RC_NOT_ARRAY`	
`NNI_RC_NOT_RESIZABLE`	
> 0	Natural error number.

nni_delete_parm

Syntax

```
int nni_delete_parm(void* pnni_func, void* parmhandle);
```

Deletes the specified parameter set.

Parameters

Parameter	Meaning
`pnni_func`	Pointer to an NNI interface instance.
`parmhandle`	Pointer to a parameter set.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
NNI_RC_OK	
NNI_RC_NOT_INIT	
NNI_RC_PARM_ERROR	

nni_from_string

Syntax

```
int nni_from_string(void* pnni_func, const char* szString, char chFormat, int iLength, int iPrecision, int iBufferLength, void* pBuffer);
```

Converts the string representation of a Natural P, N, D or T value into the internal representation of the value, as it is used in the functions nni_get_parm, nni_get_parm_array, nni_put_parm and nni_put_parm_array.

The string representations of these Natural data types look like this:

Format	String representation
P, N	E. g. "-3.141592", where the decimal character defined in the DC parameter is used.
D	Date format as defined in the DTFORM parameter, (e. g. "2004-07-06", if DTFORM=I).
T	Date format as defined in the DTFORM parameter, combined with a Time value in the form hh:ii:ss:t (e. g. "2004-07-06 11:30:42:7", if DTFORM=I) or Time value in the form hh:ii:ss:t (e. g. "11:30:42:7").

Parameters

Parameter	Meaning
pnni_func	Pointer to an NNI interface instance.
szString	String representation of the value.
chFormat	Natural data type of the value.
iLength	Natural length of the value. The total number of significant digits in the case of NNI_TYPE_NUM and NNI_TYPE_PACK, 0 otherwise.
iPrecision	Number of decimal places in the case of NNI_TYPE_NUM and NNI_TYPE_PACK, 0 otherwise.
iBufferLength	Length of the buffer provided in pBuffer.
pBuffer	Buffer that contains the internal representation of the value on return. The buffer must be large enough to hold the internal Natural representation of the value. The required sizes are documented in *Format and Length of User-Defined Variables*.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
NNI_RC_OK	
NNI_RC_NOT_INIT	
NNI_RC_PARM_ERROR	
> 0	Natural error number

nni_to_string

Syntax

```
int nni_to_string(void* pnni_func, int iBufferLength, const void* pBuffer, char chFormat, int iLength, int iPrecision, int iStringLength, char* szString);
```

Converts the internal representation of a Natural P, N, D or T value, as it is used in the functions nni_get_parm, nni_get_parm_array, nni_put_parm and nni_put_parm_array, into a the string representation.

The string representations of these Natural data types look as described with the function nni_from_string.

Parameters

Parameter	Meaning
pnni_func	Pointer to an NNI interface instance.
iBufferLength	Length of the buffer provided in pBuffer.
pBuffer	Buffer that contains the internal representation of the value. The required sizes are documented in *Format and Length of User-Defined Variables*.
chFormat	Natural data type of the value.
iLength	Natural length of the value. The total number of significant digits in the case of NNI_TYPE_NUM and NNI_TYPE_PACK, 0 otherwise.
iPrecision	Number of decimal places in the case of NNI_TYPE_NUM and NNI_TYPE_PACK, 0 otherwise.
iStringLength	Length of the string buffer provided in szString including the terminating zero.
szString	String buffer that contains the string representation of the value on return. The string buffer must be large enough to hold the external representation including the terminating zero.

Return Codes

The meaning of the return codes is explained in the section *Return Codes*.

Return Code	Remark
`NNI_RC_OK`	
`NNI_RC_NOT_INIT`	
`NNI_RC_PARM_ERROR`	
> 0	Natural error number