Nail to Nail Fingerprint Capture Challenge

Auto-generated API Documentation

January 26, 2017

Contents

1 Main Page

1.1 Overview

This is the API for participant-specific one-to-many template generation and template matching algorithms for Intelligence Advanced Research Projects Activity's (IARPA) 2017 Nail to Nail Fingerprint Capture Challenge. This API is based off the API used for Fingerprint Vendor Technology Evaluation (FpVTE) 2012, by the National Institute of Standards and Technology (NIST), released in the public domain.

1.2 Implementation

A pure-virtual (abstract) class called N2N::Interface has been created. Participants must implement all methods of N2N::Interface in a subclass, and submit this implementation as a shared library. The name of the library must follow the instructions in N2N::Interface::getIDs(). A test application will link against the submitted library, instantiate an instance of the implementation by calling N2N::Interface↩ ::getImplementation(), and perform various template generation and template matching operations.

1.3 Contact

Additional information regarding the Nail to Nail Fingerprint Capture Challenge can be obtained by emailing N2NChallenge@iarpa.gov. Additional information regarding this API and the associated software test can be obtained by emailing N2NChallenge@nist.gov.

1.4 License

2 Namespace Documentation

2.1 N2N Namespace Reference

Contains all the structures and functions used by the API.

Classes

- struct Candidate

  Object used to report a single candidate in a candidate list.
- struct FingerImage

  Fingerprint image and image attributes.
- class Interface

  The interface to the implementations.
- struct ReturnStatus

  Information about the completion status of a method.

Typedefs

- using FingerImage = struct FingerImage

  Convenience type for struct FingerImage.
- using Candidate = struct Candidate

  Convenience type for struct Candidate.
- using ReturnStatus = struct ReturnStatus

  Convenience definition of struct ReturnStatus.

Enumerations

- enum StatusCode { StatusCode::Success = 0, StatusCode::ImageSizeNotSupported = 1, StatusCode::TemplateTypeNotSupported = 2, StatusCode::FailedToExtract = 3, StatusCode::FailedToSearch = 4, StatusCode::FailedToParseInput = 5, StatusCode::InsufficientResources = 6, StatusCode::Vendor = 7 }

  The status codes that are returned from a function call.
- enum InputType { InputType::Capture, InputType::Latent }

  Classes of imagery that can be provided.

Functions

- std::ostream & operator<< (std::ostream &s, const StatusCode &sc)

  Output stream operator for a StatusCode object.
- std::ostream & operator<< (std::ostream &s, const ReturnStatus &rs)

  Output stream operator for a ReturnStatus object.

2.1.1 Detailed Description

Contains all the structures and functions used by the API.

2.1.2 Typedef Documentation

2.1.2.1 Candidate

```
using N2N::Candidate = typedef struct Candidate
```

Convenience type for struct Candidate.

2.1.2.2 FingerImage

```
using N2N::FingerImage = typedef struct FingerImage
```

Convenience type for struct FingerImage.

2.1.2.3 ReturnStatus

```
using N2N::ReturnStatus = typedef struct ReturnStatus
```

Convenience definition of struct ReturnStatus.

2.1.3 Enumeration Type Documentation

2.1.3.1 InputType

```
enum N2N::InputType  [strong]
```

Classes of imagery that can be provided.

Enumerator

Capture	Images where the subject was present during collection.
Latent	Images where a subject was not present during collection.

2.1.3.2 StatusCode

```
enum N2N::StatusCode  [strong]
```

The status codes that are returned from a function call.

Enumerator

Success	Successful completion.
ImageSizeNotSupported	Image size too small or large.
TemplateTypeNotSupported	Unsupported template type.
FailedToExtract	Could not extract template from image.
FailedToSearch	Could not search enrollment set.
FailedToParseInput	Failure to parse data.
InsufficientResources	There are not enough resources to complete the task.
Vendor	Vendor-defined error.

2.1.4 Function Documentation

2.1.4.1 operator<<() [1/2]

```
std::ostream& N2N::operator<< (
              std::ostream & s,
              const StatusCode & sc )  [inline]
```

Output stream operator for a StatusCode object.

2.1.4.2 operator<<() [2/2]

```
std::ostream& N2N::operator<< (
              std::ostream & s,
              const ReturnStatus & rs )  [inline]
```

Output stream operator for a ReturnStatus object.

3 Class Documentation

3.1 N2N::Candidate Struct Reference

Object used to report a single candidate in a candidate list.

`#include <n2n.h>`

Public Member Functions

- Candidate ()=default
  Constructor.
- Candidate (const std::string &templateID, double similarity)
  Constructor.

Public Attributes

- std::string templateID {}
  Candidate's ID, as provided during finalizeEnrollment().
- double similarity {-1}
  Score reflecting similarity between candidate represented by `templateID` *and search template.*

3.1.1 Detailed Description

Object used to report a single candidate in a candidate list.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Candidate() [1/2]

```
N2N::Candidate::Candidate ( )  [default]
```

Constructor.

3.1.2.2 Candidate() [2/2]

```
N2N::Candidate::Candidate (
              const std::string & templateID,
              double similarity )  [inline]
```

Constructor.

Parameters

in	*templateID*	Candidate ID, as provided during finalizeEnrollment().
in	*similarity*	Similarity of `templateID` to search template.

3.1.3 Member Data Documentation

3.1.3.1 templateID

```
std::string N2N::Candidate::templateID {}
```

Candidate's ID, as provided during finalizeEnrollment().

In a candidate list, the empty string represents that no candidate was found at this position.

3.2 N2N::FingerImage Struct Reference

Fingerprint image and image attributes.

```
#include <n2n.h>
```

Public Member Functions

- FingerImage ()=default
  Constructor.
- FingerImage (const BiometricEvaluation::Finger::Position &fgp, const BiometricEvaluation::↩ Finger::Impression &imp, const uint8_t nfiq2, const std::shared_ptr< BiometricEvaluation::↩ Image::Raw > &rawImage)
  Constructor.

Public Attributes

- BiometricEvaluation::Finger::Position fgp
  Finger position of finger in ***rawImage****.*
- BiometricEvaluation::Finger::Impression imp
  Impression type of finger in ***rawImage****.*
- uint8_t nfiq2 {254}
  NFIQ2 value.
- std::shared_ptr< BiometricEvaluation::Image::Raw > rawImage {}
  Input image data, containing one finger.

3.2.1 Detailed Description

Fingerprint image and image attributes.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 FingerImage()

```
N2N::FingerImage::FingerImage (
              const BiometricEvaluation::Finger::Position & fgp,
              const BiometricEvaluation::Finger::Impression & imp,
              const uint8_t nfiq2,
              const std::shared_ptr< BiometricEvaluation::Image::Raw > & rawImage )  [inline]
```

Constructor.

Parameters

in	*fgp*	Finger position of finger in `rawImage`.
in	*imp*	Impression type of finger in `rawImage`.
in	*nfiq2*	NFIQ2 value of `rawImage`.
in	*rawImage*	Input image data.

3.2.3 Member Data Documentation

3.2.3.1 fgp

```
BiometricEvaluation::Finger::Position N2N::FingerImage::fgp
```

Initial value:

```
{
                BiometricEvaluation::Finger::Position::Unknown}
```

Finger position of finger in `rawImage`.

3.2.3.2 imp

```
BiometricEvaluation::Finger::Impression N2N::FingerImage::imp
```

Initial value:

```
{
                BiometricEvaluation::Finger::Impression::Unknown}
```

Impression type of finger in `rawImage`.

Note

No differentiation is provided between "traditional" and "participant sensor" rolled impressions.

3.2.3.3 nfiq2

```
uint8_t N2N::FingerImage::nfiq2 {254}
```

NFIQ2 value.

Meaning	Value
Quality	0 (low) – 100 (high)
Not Calculated	254
Error During Calculation	255

3.2.3.4 rawImage

```
std::shared_ptr<BiometricEvaluation::Image::Raw> N2N::FingerImage::rawImage {}
```

Input image data, containing one finger.

3.3 N2N::Interface Class Reference

The interface to the implementations.

`#include <n2n.h>`

Public Member Functions

- virtual void getIDs (std::string &identifier, uint32_t &revision, std::string &email)=0

  Obtain identifying information about the software under test.
- virtual ReturnStatus initMakeEnrollmentTemplate (const std::string &configurationDirectory)=0

  Prepare for calls to makeEnrollmentTemplate().
- virtual ReturnStatus makeEnrollmentTemplate (const std::vector< FingerImage > &standard↩ Images, const std::vector< BiometricEvaluation::Memory::uint8Array > &proprietaryImages, BiometricEvaluation::Memory::uint8Array &enrollmentTemplate)=0

  Create an enrollment template for one subject.
- virtual ReturnStatus finalizeEnrollment (const std::string &configurationDirectory, const std↩ ::string &enrollmentDirectory, const uint8_t nodeCount, const uint64_t nodeMemory, Biometric↩ Evaluation::IO::RecordStore &enrollmentTemplates)=0

  Form an enrollment set from one or more enrollment templates.
- virtual ReturnStatus initMakeSearchTemplate (const std::string &configurationDirectory, const InputType &inputType)=0

  Prepare for calls to makeSearchTemplate().
- virtual ReturnStatus makeSearchTemplate (const std::vector< FingerImage > &standardImages, const std::vector< BiometricEvaluation::Memory::uint8Array > &proprietaryImages, Biometric↩ Evaluation::Memory::uint8Array &searchTemplate)=0

  Create a search template for one subject.
- virtual ReturnStatus initIdentificationStageOne (const std::string &configurationDirectory, const std::string &enrollmentDirectory, const InputType &inputType, const uint8_t nodeNumber)=0

  Prepare for calls to identifyTemplateStageOne().
- virtual ReturnStatus identifyTemplateStageOne (const std::string &searchID, const Biometric↩ Evaluation::Memory::uint8Array &searchTemplate, const std::string &stageOneDataDirectory)=0

  Search a template against the partial enrollment set.
- virtual ReturnStatus initIdentificationStageTwo (const std::string &configurationDirectory, const std::string &enrollmentDirectory, const InputType &inputType)=0

  Prepare for calls to identifyTemplateStageTwo().
- virtual ReturnStatus identifyTemplateStageTwo (const std::string &searchID, const std::string &stageOneDataDirectory, std::vector< Candidate > &candidates)=0

  Produce a candidate list from the results of all calls to identifyTemplateStageOne() for a particular search ID.
- virtual ~Interface ()=default

  Destructor.

Static Public Member Functions

- static std::shared_ptr< Interface > getImplementation ()
  Obtain a managed pointer to an implementation of this interface.

3.3.1 Detailed Description

The interface to the implementations.

The implementation under test will implement this interface by subclassing this class and implementing each method.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 ~Interface()

```
virtual N2N::Interface::~Interface ( )  [virtual], [default]
```

Destructor.

3.3.3 Member Function Documentation

3.3.3.1 finalizeEnrollment()

```
virtual ReturnStatus N2N::Interface::finalizeEnrollment (
                const std::string & configurationDirectory,
                const std::string & enrollmentDirectory,
                const uint8_t nodeCount,
                const uint64_t nodeMemory,
                BiometricEvaluation::IO::RecordStore & enrollmentTemplates )  [pure virtual]
```

Form an enrollment set from one or more enrollment templates.

This finalization step will prepare the enrolled templates to be distributed across multiple nodes. The enrollment directory will then be read-only throughout the duration of the identification process.

Parameters

in	*configurationDirectory*	A read-only directory containing vendor-supplied configuration parameters or run-time data files.
in	*enrollmentDirectory*	The top-level directory in which all enrollment data will reside. Access permission will be read-write and the application can populate this directory as needed. The directory is initially empty. After this method returns, the directory and its contents will become read-only.
in	*nodeCount*	The number of nodes the enrollment set will be spread across. It is up to the implementation to determine how best to spread the enrolled templates across the blades in order to get best performance. If `nodeCount` is not enough nodes, StatusCode::InsufficientResources
		should be returned.
in	*nodeMemory*	Amount of memory available to this process on each node, in kibibytes.
in	*enrollmentTemplates*	A read-only RecordStore of enrollment templates, as returned by makeEnrollmentTemplate().

Returns

Completion status of the operation.

Exceptions

BiometricEvaluation::Error::Exception	There was an error processing this request, and the exception string may contain additional information.

Note

All implementations must be capable of performing searches using <= 5 nodes. A larger value may be provided for speed, or a smaller value provided to conserve resources. If a smaller value is not feasible, StatusCode::InsufficientResources should be returned. Implementations that do not return successfully for values >= 5 will be disqualified.
The file system does not perform well with the creation of millions of small files, so the application should consolidate templates into some sort of database file within `enrollmentDirectory`.
This method must return within 90 minutes per 1-million subjects (e.g., if 5-million enrollment templates are provided, this method must return within 7.5 hours).
Reasonable multithreading is permitted. This method will only be called once.

3.3.3.2 getIDs()

```
virtual void N2N::Interface::getIDs (
               std::string & identifier,
               uint32_t & revision,
               std::string & email )  [pure virtual]
```

Obtain identifying information about the software under test.

Participants will receive an identifier from the project sponsor, and use this method to hard-code the identifier into the submission. The information obtained by this method must form the name of the submitted library, in the form `libN2N_<identifier>_<revision>.so`.

Parameters

`out`	*identifier*	The identifier provided to you by the project sponsor.
`out`	*revision*	A unique revision number for this submission. No two submission revision numbers may be the same, and subsequent submissions should only ever increase this value.
`out`	*email*	Point of contact email address.

Note

This method must return immediately.

3.3.3.3 getImplementation()

```
static std::shared_ptr<Interface> N2N::Interface::getImplementation ( )  [static]
```

Obtain a managed pointer to an implementation of this interface.

Returns

A managed pointer to the Interface subclass implementation.

3.3.3.4 identifyTemplateStageOne()

```
virtual ReturnStatus N2N::Interface::identifyTemplateStageOne (
                const std::string & searchID,
                const BiometricEvaluation::Memory::uint8Array & searchTemplate,
                const std::string & stageOneDataDirectory )  [pure virtual]
```

Search a template against the partial enrollment set.

Parameters

in	*searchID*	The ID of the search template. This ID does not identify subject, but is merely an identifier used to distinguish different searches performed by the system. It will be used as the input to identifyTemplateStageTwo().
in	*searchTemplate*	A template from makeSearchTemplate().
in	*stageOneDataDirectory*	This directory will have read-write access. The output information from identifyTemplateStageOne() that is needed in identifyTemplateStageTwo() is written in this directory. This directory will be unique for each search performed.

Returns

Completion status of the operation.

Exceptions

BiometricEvaluation::Error::Exception	There was an error processing this request, and the exception string may contain additional information.

Note

All calls to combined identification functions (identifyTemplateStageOne() + identifyTemplate↩ StageTwo()) for a single `searchID` must return within 60 seconds for InputType::Capture and 300 seconds for InputType::Latent. If identifyTemplateStageOne() takes 55 seconds for searchID XYZ (InputType::Capture), identifyTemplateStageTwo() must complete within 5 seconds for the same search ID.

Attention

Multithreading and other multiprocessing techniques are absolutely not permitted. The testing application will be calling this method from multiple processes on the same node.

3.3.3.5 identifyTemplateStageTwo()

```
virtual ReturnStatus N2N::Interface::identifyTemplateStageTwo (
              const std::string & searchID,
              const std::string & stageOneDataDirectory,
              std::vector< Candidate > & candidates )  [pure virtual]
```

Produce a candidate list from the results of all calls to identifyTemplateStageOne() for a particular search ID.

identifyTemplateStageOne() with searchID was called >= 1 times on separate nodes, ideally searching different subsets of the full enrolled set. In this method, the implementation should parse the results of the first search stage to form a final candidate list. This method will only be called once per searchID and only on a single node.

Parameters

`in`	*searchID*	The ID of the search template. This ID does not identify subject, but is merely an identifier used to distinguish different searches performed by the system.
`in`	*stageOneDataDirectory*	A read-only version of the data generated by identifyTemplateStageOne().
`out`	*candidates*	The candidate list.

Returns

Completion status of the operation.

Exceptions

BiometricEvaluation::Error::Exception	There was an error processing this request, and the exception string may contain additional information.

Note

All calls to combined identification functions (identifyTemplateStageOne() + identifyTemplate↩ StageTwo()) for a single `searchID` must return within 60 seconds for InputType::Capture and 300 seconds for InputType::Latent. If identifyTemplateStageOne() takes 55 seconds for searchID XYZ (InputType::Capture), identifyTemplateStageTwo() must complete within 5 seconds for the same search ID.
`candidates` will have `reserve()` called prior to calling this method.
There shall be [0,100] objects in `candidates` after the successful return of this method.
`candidates` shall be sorted by descending similarity score before returning.

Attention

Multithreading and other multiprocessing techniques are absolutely not permitted. The testing application will be calling this method from multiple processes on the same node.

3.3.3.6 initIdentificationStageOne()

```
virtual ReturnStatus N2N::Interface::initIdentificationStageOne (
              const std::string & configurationDirectory,
              const std::string & enrollmentDirectory,
              const InputType & inputType,
              const uint8_t nodeNumber )  [pure virtual]
```

Prepare for calls to identifyTemplateStageOne().

The function will be called to initialize each node that will contain a portion of the enrolled templates. The number of nodes will be the same as provided in finalizeEnrollment().

Parameters

in	*configurationDirectory*	A read-only directory containing vendor-supplied configuration parameters or run-time data files.
in	*enrollmentDirectory*	The top-level read-only directory in which all finalized enrollment data resides. The contents of this directory is identical to the enrollmentDirectory parameter from finalizeEnrollment(), but the path may not be the same.
in	*inputType*	The types of images that will be provided during all subsequent calls to identifyTemplateStageOne().
in	*nodeNumber*	Node number from nodes in finalizeEnrollment() that is being initialized. This parameter lets the callee know which piece of the enrolled templates to load into memory. Nodes are numbered 0 to (N - 1).

Returns

Completion status of the operation.

Exceptions

BiometricEvaluation::Error::Exception	There was an error processing this request, and the exception string may contain additional information.

Note

This method must complete with 5 minutes. Reasonable multithreading is permitted.

3.3.3.7 initIdentificationStageTwo()

```
virtual ReturnStatus N2N::Interface::initIdentificationStageTwo (
              const std::string & configurationDirectory,
              const std::string & enrollmentDirectory,
              const InputType & inputType )  [pure virtual]
```

Prepare for calls to identifyTemplateStageTwo().

This second stage of identification uses the results from identifyTemplateStageOne() to produce a candidate list for the search subject.

Parameters

in	*configurationDirectory*	A read-only directory containing vendor-supplied configuration parameters or run-time data files.
in	*enrollmentDirectory*	The top-level directory in which all finalized enrolled data resides. The directory will have read-only access.
in	*inputType*	The types of images that will be provided during all subsequent calls to identifyTemplateStageTwo().

Returns

Completion status of the operation.

Exceptions

BiometricEvaluation::Error::Exception	There was an error processing this request, and the exception string may contain additional information.

Note

This method must complete with 5 minutes. Reasonable multithreading is permitted.

3.3.3.8 initMakeEnrollmentTemplate()

```
virtual ReturnStatus N2N::Interface::initMakeEnrollmentTemplate (
              const std::string & configurationDirectory )  [pure virtual]
```

Prepare for calls to makeEnrollmentTemplate().

The function is called once by the testing application before N >= 1 calls to makeEnrollmentTemplate() on the current node. The implementation must tolerate execution of this initialization function and other N >= 1 calls to makeEnrollmentTemplate() running simultaneously and independently on the same and/or multiple machines.

Parameters

in	*configurationDirectory*	A read-only directory containing vendor-supplied configuration parameters or run-time data files.

Returns

Completion status of the operation.

Exceptions

BiometricEvaluation::Error::Exception	There was an error processing this request, and the exception string may contain additional information.

Note

This method must complete with 5 minutes. Reasonable multithreading is permitted.

3.3.3.9 initMakeSearchTemplate()

```
virtual ReturnStatus N2N::Interface::initMakeSearchTemplate (
                const std::string & configurationDirectory,
                const InputType & inputType )  [pure virtual]
```

Prepare for calls to makeSearchTemplate().

The function is called once by the testing application before N >= 1 calls to makeSearchTemplate() on the current node. The implementation must tolerate execution of this initialization function and other N >= 1 calls to makeSearchTemplate() running simultaneously and independently on the same and/or multiple machines.

Parameters

in	*configurationDirectory*	A read-only directory containing vendor-supplied configuration parameters or run-time data files.
in	*inputType*	The types of images that will be provided during all subsequent calls to makeSearchTemplate().

Returns

Completion status of the operation.

Exceptions

BiometricEvaluation::Error::Exception	There was an error processing this request, and the exception string may contain additional information.

Note

90% of calls to this method must return in three seconds or less.
This method must complete with 5 minutes. Reasonable multithreading is permitted.

3.3.3.10 makeEnrollmentTemplate()

```
virtual ReturnStatus N2N::Interface::makeEnrollmentTemplate (
                const std::vector< FingerImage > & standardImages,
                const std::vector< BiometricEvaluation::Memory::uint8Array > & proprietaryImages,
                BiometricEvaluation::Memory::uint8Array & enrollmentTemplate )  [pure virtual]
```

Create an enrollment template for one subject.

This method provides one or more fingerprints from a subject and tasks the implementation with creating and returning an object that can represent this subject in an enrollment set.

Parameters

in	*standardImages*	One or more finger images from a single subject.
in	*proprietaryImages*	One or more proprietary representations of fingers, as returned from the participant's sensor.
out	*enrollmentTemplate*	A non-regulated representation of fingers for an enrollment set.

Returns

Completion status of the operation.

Exceptions

BiometricEvaluation::Error::Exception	There was an error processing this request, and the exception string may contain additional information.

Note

This method should call BiometricEvaluation::Memory::uint8Array::resize() *before* any writes to `enrollmentTemplate` to ensure it is large enough to contain the write. This method should also call BiometricEvaluation::Memory::uint8Array::resize() *before* returning so that `enrollmentTemplate` is the exact required size. All BiometricEvaluation::Memory::uint8Array::size() bytes of `enrollment↩ Template` will be provided to the N2N::Interface implementation during finalizeEnrollment().
90% of calls to this method must return in three seconds or less.

Attention

Multithreading and other multiprocessing techniques are absolutely not permitted. The testing application will be calling this method from multiple processes on the same node.

3.3.3.11 makeSearchTemplate()

```
virtual ReturnStatus N2N::Interface::makeSearchTemplate (
              const std::vector< FingerImage > & standardImages,
              const std::vector< BiometricEvaluation::Memory::uint8Array > & proprietaryImages,
              BiometricEvaluation::Memory::uint8Array & searchTemplate )  [pure virtual]
```

Create a search template for one subject.

This method provides one or more fingerprints from a subject and tasks the implementation with creating and returning an object that can represent this subject as a search initiator.

Parameters

in	*standardImages*	One or more finger images from a single subject.
in	*proprietaryImages*	One or more proprietary representations of fingers, as returned from the participant's sensor.
out	*searchTemplate*	A non-regulated representation of fingers used to initiate a search.

Returns

Completion status of the operation.

Exceptions

BiometricEvaluation::Error::Exception	There was an error processing this request, and the exception string may contain additional information.

Note

This method should call BiometricEvaluation::Memory::uint8Array::resize() *before* any writes to `searchTemplate` to ensure it is large enough to contain the write. This method should also call BiometricEvaluation::Memory::uint8Array::resize() *before* returning so that `searchTemplate` is the exact required size. All BiometricEvaluation::Memory::uint8Array::size() bytes of `enrollment↩Template` will be provided to the N2N::Interface implementation during finalizeEnrollment().

Attention

Multithreading and other multiprocessing techniques are absolutely not permitted. The testing application will be calling this method from multiple processes on the same node.

3.4 N2N::ReturnStatus Struct Reference

Information about the completion status of a method.

`#include <n2n.h>`

Public Member Functions

- ReturnStatus ()=default
  Constructor.
- ReturnStatus (const StatusCode code, const std::string info)
  Constructor.

Public Attributes

- StatusCode code {StatusCode::Success}
  Completion status of the returning method.
- std::string info {}
  Additional clarifying information about `code`.

3.4.1 Detailed Description

Information about the completion status of a method.

An object of this class allows the software to return some information from a method call. The string within this object can be optionally set to provide more information for debugging. The status code will be set by the function to Success on success, or one of the other codes on failure. In failure cases, processing will proceed with further calls to the function.

Note

If the SDK encounters a non-recoverable error, an exception should be thrown and processing will stop.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 ReturnStatus() [1/2]

```
N2N::ReturnStatus::ReturnStatus ( )  [default]
```

Constructor.

3.4.2.2 ReturnStatus() [2/2]

```
N2N::ReturnStatus::ReturnStatus (
            const StatusCode code,
            const std::string info )  [inline]
```

Constructor.

Parameters

in	*code*	The return status code.
in	*info*	The optional information string.

3.4.3 Member Data Documentation

3.4.3.1 code

```
StatusCode N2N::ReturnStatus::code {StatusCode::Success}
```

Completion status of the returning method.

3.4.3.2 info

```
std::string N2N::ReturnStatus::info {}
```

Additional clarifying information about `code`.

Index