Table of Contents

[Overview 3](#_Toc456618099)

[Walkthrough: The Abstraction Process 6](#_Toc456618100)

[Initializing the TNM staging library 6](#_Toc456618101)

[Selecting a schema 6](#_Toc456618102)

[Obtaining a schema discriminator 7](#_Toc456618103)

[Retrieving a discriminator table 8](#_Toc456618104)

[Example 1 8](#_Toc456618105)

[Example 2 9](#_Toc456618106)

[Building a table picklist 10](#_Toc456618107)

[Abstracting required data items 11](#_Toc456618108)

[Optional validation 12](#_Toc456618109)

[Calculating derived stage group 12](#_Toc456618110)

[Schemas 14](#_Toc456618111)

[Enumerated Types 16](#_Toc456618112)

[DataElement 16](#_Toc456618113)

[SchemaIdentifier 18](#_Toc456618114)

[TNMTableRole 19](#_Toc456618115)

[ErrorCode 20](#_Toc456618116)

[Explanation of codes 22](#_Toc456618117)

[Notable Classes 26](#_Toc456618118)

[Datacard 26](#_Toc456618119)

[TableInfo 31](#_Toc456618120)

[TNMTable 31](#_Toc456618121)

[TNMSchema 36](#_Toc456618122)

[TNM\_Staging 39](#_Toc456618123)

[API reference 40](#_Toc456618124)

[General information 40](#_Toc456618125)

[GetVersion 40](#_Toc456618126)

[Data transfer 40](#_Toc456618127)

[PopulateDatacardFromBuffer 40](#_Toc456618128)

[PopulateBufferFromDatacard 41](#_Toc456618129)

[Formatting Error Messages 41](#_Toc456618130)

[FormatErrorCode 41](#_Toc456618131)

[Formatting Storage and Display codes 42](#_Toc456618132)

[GetStorageCode 42](#_Toc456618133)

[GetDisplayCode 43](#_Toc456618134)

[Schema selection and access 44](#_Toc456618135)

[GetSchemaNumber 44](#_Toc456618136)

[GetSchemaNumberBySchemaIdentifier 46](#_Toc456618137)

[GetSchemaName 46](#_Toc456618138)

[GetNumberOfSchemas 47](#_Toc456618139)

[GetSchema 47](#_Toc456618140)

[GetSchemaBySchemaIdentifier 48](#_Toc456618141)

[Table access 48](#_Toc456618142)

[GetNumberOfTables 48](#_Toc456618143)

[GetTable 49](#_Toc456618144)

[GetTableByDataElement 49](#_Toc456618145)

[GetTableByRole 50](#_Toc456618146)

[Coding required data elements 50](#_Toc456618147)

[GetRequiredDataItems 51](#_Toc456618148)

[Stage calculation 51](#_Toc456618149)

[CalculateStage 52](#_Toc456618150)

# Overview

The purpose of CDC’s TNM staging library is two-fold: to support collection of AJCC 7th edition staging data from reporting facilities by facilitating generation of authoritative site-specific pick-lists for clinical and pathologic T, N, M, and directly entered stage groups; and to provide an algorithm for deriving clinical and pathologic stage groups for use by NPCR central registries for quality control and data consolidation. The library is intended to be used for cancer cases diagnosed in 2016.

For the year 2016, there will be overlap between cases abstracted and diagnosed in 2016 and cases diagnosed earlier but not abstracted until 2016. For NPCR, the TNM staging library is intended to replace Collaborative Staging. Since TNM staging is only valid for 2016 and later, and Collaborative Staging is no longer used in 2016, for 2016, vendor software is expected to handle multiple staging systems and change workflows based on the diagnosis year. See [Figure 1](#Figure_1) for a description of how NPCR handles the two staging systems by year of diagnosis.

The scope of the TNM staging library does not include collection of data items other than those needed to derive TNM stage. For other collection requirements, contact your standard setter.

The TNM staging library exposes an API (Application Programming Interface) to support abstraction of TNM data items, generation of picklists, and derivation of TNM stage. Some familiarity with the NAACCR format is required and can be found here: <http://www.naaccr.org/StandardsandRegistryOperations/VolumeII.aspx>

The library is written in C#/.NET. A basic understanding of C# is required to use the API. A Windows environment with .NET Framework version 4.5 or higher is required.



Figure

A general workflow of how TNM staging fits into the abstraction process:

* The abstractor enters the year of diagnosis. If the year is 2016 or later, continue with TNM staging. Otherwise, exit this workflow.
* The abstractor enters values for primary site and histologic type.
* Call the schema selection function to obtain a schema number
	+ If the function returns with a valid schema number, continue to the next step (building picklists)
	+ If the function indicates that staging is not applicable, fill in the derived clinical and pathologic stage group fields with default values of “88”. Exit this workflow.
	+ If the function indicates that a schema discriminator is required:
		- Locate the schema discriminator table
		- Construct a picklist and display it
			* For site-specific factor (SSF) discriminators, construct a picklist using the schema discriminator table
			* For non-SSF discriminators, construct a picklist using valid values from the NAACCR manual
		- The abstractor selects a discriminator value
		- Call the schema selection function again with the discriminator value included. Handle the return value according to the scenarios listed for the first call to the schema selection function. Repeat if necessary.
* Call the staging library to determine which data items are required to derive stage for the schema.
* For each required data item, construct a picklist and display it. See [Figure 2](#Figure_2) for a sample picklist.
* The abstractor selects a value for each required data item
* Populate a Datacard with input values
* Call the library’s stage calculation function
* Store calculation results in file or database table



Figure : Example picklist table

# Walkthrough: The Abstraction Process

What follows is a basic explanation of how TNM data items are abstracted. The walkthrough is divided into sections, with sample C# code snippets included in each section. Some code has been omitted for clarity or left as an exercise to the user.

A sample C# code file for the walkthrough has also been included with the library.

## Initializing the TNM staging library

The first step to using the TNM staging library is to create an instance of TNM\_Stage. This instance should be kept in memory until it is no longer needed by the calling program.

// The namespace used by the TNM staging library

using TNM\_Staging;

// Declare program namespace, class, and containing method here

//

TNM\_Stage stageObject = new TNM\_Stage();

## Selecting a schema

Next, create a Datacard (a data structure that supplies input values and receives derived stage group values) and set the values for primary site (NAACCR item #400, Primary Site) and histologic type (NAACCR item #522, Histologic Type ICD-O-3). These data items are always required to select a schema.

Then, call the schema selection function, GetSchemaNumber().

Datacard dc = new Datacard();

// Pick some default values for demonstration purposes

dc.site = "C111";

dc.histology = "8000";

// Get a schema number

int schemaNumber = 0;

DataElement requiredElement = DataElement.NONE;

ErrorCode errorCode = stageObject.GetSchemaNumber(ref dc, ref schemaNumber, ref requiredElement);

GetSchemaNumber() uses the Datacard values to identify a schema. The function return value indicates if a schema could be selected. The function takes two additional pass-by-reference parameters: one to hold the schema number, and one to indicate if another data item is required to select the schema when there are multiple candidates.

**NOTE: Schema numbers (and all other index numbers for tables, columns, notes, etc.) start from 0.**

There are three possible outcomes to selecting a schema:

* A schema could be identified
* A schema could not be identified because no schema is staged for that combination of values
* A schema could not be identified because multiple schemas are staged for that combination of values

In the first case, we proceed directly to abstracting TNM data items. In the second case, since a schema cannot be identified, we fill in default values for derived clinical and pathologic stage group, and skip abstracting the other TNM data items. In the third case, an additional element is required to select the schema: a **schema discriminator**.

if (errorCode == ErrorCode.OK)

{

// Do nothing, proceed directly to abstracting TNM data items

}

else if (errorCode == ErrorCode.SCHEMA\_NOT\_APPLICABLE)

{

// Fill in default values for derived clinical/pathologic stage group

// Skip abstracting TNM data items

}

else if (errorCode == ErrorCode.SCHEMA\_DISCRIMINATOR\_REQUIRED)

{

// Build a schema discriminator picklist and let the user select a value

}

## Obtaining a schema discriminator

The next step is to identify which data item is the schema discriminator. If a schema discriminator is not required, this step is skipped.

When a schema discriminator is required, GetSchemaNumber() will return the necessary information to the caller.

* The function returns a value of SCHEMA\_DISCRIMINATOR\_REQUIRED to indicate that a discriminator is required.
* Parameter schemaNumber is set to the number of one of the schemas that share the discriminator.
* Parameter ***requiredDataElement*** is set to the [DataElement](#_DataElement) that matches the schema discriminator.

**NOTE: In TNM staging 7th edition, only the following data items can be a schema discriminator**

* Site-specific factor 10 (NAACCR Item # 2864)
* Site-specific factor 25 (NAACCR Item # 2879)
* Sex (NAACCR Item # 220)
* Grade (NAACCR Item # 440)

## Retrieving a discriminator table

Discriminator tables are not included for certain schemas. The reason is that tables have only been mapped for data items that are TNM-specific. As a general rule of thumb, site-specific factors that are needed for staging have tables, while basic item such as Age, Behavior, Grade, and Sex do not.

Since the discriminator is returned as a DataElement, the API user should call GetTableByDataElement() to retrieve the table. If the function returns a value of TABLE\_NOT\_FOUND or the returned table is null, the user must construct their own discriminator table using the data item’s definition.

### Example 1

Here’s an example where a discriminator table can be retrieved.

Choose the following values:

Site = C111, Hist = 8000

int schemaNumber = 0;

DataElement requiredElement = DataElement.NONE;

ErrorCode errorCode = stageObject.GetSchemaNumber(ref dc, ref schemaNumber, ref requiredElement);

GetSchemaNumber returns the following values:

errorCode == SCHEMA\_DISCRIMINATOR\_REQUIRED, schemaNumber = 56, requiredElement == SSF25

Retrieve the table:

TNMTable table = null;

// Get the table corresponding to the discriminator

errorCode = stageObject.GetTableByDataElement(schemaNumber, requiredElement, ref table);

errorCode == OK, table is assigned a valid TNMTable

### Example 2

Here’s an example where a discriminator table cannot be retrieved.

Choose the following values:

Site = C481, Hist = 8000

int schemaNumber = 0;

DataElement requiredElement = DataElement.NONE;

ErrorCode errorCode = stageObject.GetSchemaNumber(ref dc, ref schemaNumber, ref requiredElement);

GetSchemaNumber returns the following:

errorCode = SCHEMA\_DISCRIMINATOR\_REQUIRED, schemaNumber = 60, requiredElement = SEX

Sex (NAACCR Item #220) is not coded as a TNM table. Instead, the API user must create their own table.

TNMTable table = null;

// Get the table corresponding to the discriminator

errorCode = stageObject.GetTableByDataElement(schemaNumber, requiredElement, ref table);

errorCode = TABLE\_NOT\_FOUND, table == null

## Building a table picklist

Once a table has been obtained, the API user can iterate over the table’s notes, column titles, and individual cells to construct a picklist form. The picklist should then be displayed to the abstractor. Once a value has been selected, the datacard should be populated with the value, and GetSchemaNumber() called again. If the user selected a discriminator value for a valid schema, that schema will be selected.

TNMTable table = null;

// Get the table corresponding to the discriminator, if possible

errorCode = stageObject.GetTableByDataElement(schemaNumber, requiredElement, ref table);

// Discriminator table found, iterate over the table to build a picklist

// For the purposes of this sample, assume that a table is returned

if (errorCode == ErrorCode.OK)

{

string value = "";

// Iterate over notes

for (int i = 0; i < table.NumNotes(); i++)

errorCode = table.GetNote(i, out value);

// Iterate over columns

for (int i = 0; i < table.NumColumns(); i++)

errorCode = table.GetColumnTitle(i, out value);

// Iterate over cells

// Storage codes are usually in the first column

for (int i = 0; i < table.NumRows(); i++)

{

 for (int j = 0; j < table.NumColumns(); j++)

errorCode = table.GetValue(i, j, out value);

}

// Build a picklist form

// Left as an exercise for the reader

}

else

{

// No table found, check the identity of the discriminator and build a picklist manually

// Left as an exercise for the reader

}

// Display picklist to user

// Left as an exercise for the reader

string discriminatorValue = "";

// Assume discriminator value is set by the user, but set it manually for this example

discriminatorValue = "010";

// Set discriminator value in Datacard

dc.PutDatacardValue(requiredElement, discriminatorValue);

// Call GetSchemaNumber() with updated datacard

errorCode = stageObject.GetSchemaNumber(ref dc, ref schemaNumber, ref requiredElement);

while (errorCode == ErrorCode.SCHEMA\_DISCRIMINATOR\_REQUIRED)

{

// Repeat the discrimnator process until a valid schema discrimnator is obtained

// discriminatorValue = new value

dc.PutDatacardValue(requiredElement, discriminatorValue);

errorCode = stageObject.GetSchemaNumber(ref dc, ref schemaNumber, ref requiredElement);

}

## Abstracting required data items

Now that we have a valid schema number, we can abstract the TNM data items required for staging. The list of required data items varies from schema to schema, so a function has been provided to retrieve those items. Though the items are returned as a single list, they can be categorized by whether they have an associated table or not, just as in the case of the schema discriminator.

We introduce a new class, TableInfo, that contains information on which table to retrieve.

TableInfo has three member variables:

* tableNumber – the integer position of the table in its parent schema
* dataElement – denotes the identity of the data item
* naaccrItemNumber – the NAACCR Item # of the data item

If *tableNumber* is valid (greater than or equal to 0), then the schema contains a table for that data item. To retrieve the table, we use another function, GetTable(), that works like GetTableByDataElement() except it takes a table number instead of a data element.

Otherwise, the API user must use *dataElement* and *naaccrItemNumber* to identify the data item and create their own picklist.

// Get required data items

List<TableInfo> requiredItems = stageObject.GetRequiredDataItems(schemaNumber);

// Iterate over the required data items

foreach (TableInfo info in requiredItems)

{

TNMTable table = null;

if (info.tableNumber >= 0)

{

errorCode = stageObject.GetTable(schemaNumber, info.tableNumber, ref table);

if (errorCode == ErrorCode.OK)

{

// Build a picklist from the table

// See the schema discriminator example for code to iterate through a

// table's contents

}

}

else

{

 // Build a picklist from an external source

}

}

## Optional validation

Any optional validation goes here. This includes handling of blank values and other default values, whose treatment will vary from vendor to vendor.

## Calculating derived stage group

Once the datacard has been populated, call CalculateStage() to derive TNM stage group. If the API user is starting with a record buffer, PopulateDatacardFromBuffer() can be used to populate the datacard directly.

CalculateStage will return an ErrorCode indicating if schema selection was successful:

* OK: a schema was selected
* SCHEMA\_NOT\_APPLICABLE: staging is not applicable for this schema
SCHEMA\_DISCRIMINATOR\_REQUIRED: a schema could not be selected because a schema discriminator is required

CalculateStage takes two additional pass-by-reference parameters (ErrorCodes), one for clinical staging and one for pathologic staging. The ErrorCodes indicate the results of clinical and pathologic staging. For a comprehensive list of ErrorCodes, consult the [ErrorCodes](#_ErrorCode) section.

ErrorCodes can be translated into messages strings by the function FormatErrorCode() and displayed to the user, if necessary.

Serious errors that can prevent the library from functioning properly are classified under ErrorCode SYSTEM\_EXCEPTION and written to a log file, “system\_exception.txt”, in the library directory. This log is used for debugging system errors.

// Call the stage calculation function

ErrorCode clinicalError = ErrorCode.OK;

ErrorCode pathologicError = ErrorCode.OK;

errorCode = stageObject.CalculateStage(ref dc, ref clinicalError, ref pathologicError);

if (errorCode == ErrorCode.OK)

{

// Get error messages

if (clinicalError != ErrorCode.OK)

{

string msg = stageObject.FormatErrorCode(clinicalError);

}

if (pathologicError != ErrorCode.OK)

{

string msg = stageObject.FormatErrorCode(pathologicError);

}

}

# Schemas

The full list of TNM schemas has been included here for reference. TNM schemas were developed specifically for TNM staging and are not the same as Collaborative Stage schemas, so this is the official list.

Each schema is listed with its display name, an identifier (that can be used to look up the schema), and the number of the AJCC chapter from which it originated.

|  |  |  |
| --- | --- | --- |
| **Schema Name** | **Schema Identifier** | **AJCC Chapter #** |
| Adrenal TNM7 | Adrenal | 47 |
| Ampulla of Vater TNM7 | AmpullaOfVater | 23 |
| Anus TNM7 | Anus | 15 |
| Appendix: Carcinoid TNM7 | AppendixCarcinoid | 13 |
| Appendix: Carcinoma TNM7 | AppendixCarcinoma | 13 |
| Bile Ducts: Cystic Duct TNM7 | BileDuctsCysticDuct | 20 |
| Bile Ducts: Distal TNM7 | BileDuctsDistal | 22 |
| Bile Ducts: Intrahepatic TNM7 | IntrahepaticBileDucts | 19 |
| Bile Ducts: Perihilar TNM7 | BileDuctsPerihilar | 21 |
| Bone TNM7 | Bone | 27 |
| Breast TNM 7 | Breast | 32 |
| Cervix Uteri TNM7 | CervixUteri | 35 |
| Colon and Rectum TNM7 | ColonRectum | 14 |
| Conjunctiva, Carcinoma TNM7 | ConjunctivaCarcinoma | 49 |
| Corpus Uteri Adenosarcoma TNM7 | CorpusUteriAdenosarcoma | 36 |
| Corpus Uteri Carcinomas TNM7 | CorpusUteriCarcinomas | 36 |
| Corpus Uteri Sarcomas TNM7 | CorpusUteriSarcomas | 36 |
| Eyelid, Carcinoma TNM7 | Eyelid | 48 |
| Fallopian Tube TNM7 | FallopianTube | 38 |
| Gallbladder TNM7 | Gallbladder | 20 |
| Gastrointestinal Stromal Tumor, Colon, Rectum, Appendix | GISTColonRectumAppendix | 16 |
| Gastrointestinal Stromal Tumor, Gastric | GISTStomach | 16 |
| Gastrointestinal Stromal Tumor, Omentum | GISTOmentum | 16 |
| Gastrointestinal Stromal Tumor, Peritoneum | GISTPeritoneum | 16 |
| Gastrointestinal Stromal Tumor, Small Intestine and Esophagus | GISTSmallIntestineEsophagus | 16 |
| Gestational Trophoblastic Disease TNM7 | GestationalTrophoblasticDisease | 39 |
| Kidney TNM7 | Kidney | 43 |
| Lacrimal Gland, Carcinoma TNM7 | LacrimalGland | 53 |
| Larynx: GlotticTNM7 | LarynxGlottic | 5 |
| Larynx: SubglotticTNM7 | LarynxSubglottic | 5 |
| Larynx: SupraglotticTNM7 | LarynxSupraglottic | 5 |
| Lip and Oral Cavity TNM7 | LipOralCavity | 3 |
| Liver TNM7 | Liver | 18 |
| Lung TNM7 | Lung | 25 |
| Lymphomas, Hodgkin and Non-Hodgkin TNM7 | Lymphoma | 57A |
| Lymphomas, Primary Cutaneous TNM7 | LymphomaCutaneousMF | 57B |
| Lymphoma, Ocular Adnexa TNM7 | LymphomaOcularAdnexa | 55 |
| Major Salivary Glands (Parotid, Submandibular, and Sublingual) TNM7 | SalivaryGlandsMajor | 7 |
| Malignant Melanoma of Ciliary Body and Choroid TNM7 | MelanomaCiliaryBody | 51 |
| Malignant Melanoma of the Conjunctiva TNM7 | MelanomaConjunctiva | 50 |
| Malignant Melanoma of the Iris TNM7 | MelanomaIris | 51 |
| Melanoma Skin TNM7 | MelanomaSkin | 31 |
| Merkel Cell Carcinoma TNM7 | MerkelCellCarcinoma | 30 |
| Mucosal Melanoma of the Head and Neck TNM7 | MelanomaHeadAndNeck | 9 |
| Nasal Cavity and Sinuses: Maxillary Sinus TNM7 | MaxillarySinus | 6 |
| Nasal Cavity and Sinuses: Nasal Cavity and Ethmoid Sinus TNM7 | NasalCavityEthmoidSinus | 6 |
| Neuroendocrine Tumors: Colon or Rectum TNM7 | NeuroendocrineTumorsColonRectum | 17 |
| Neuroendocrine Tumors: Sm. Intestine, Ampulla TNM7 | NeuroendocrineTumorsSmIntestineAmpulla | 17 |
| Neuroendocrine Tumors: Stomach TNM7 | NeuroendocrineTumorsStomach | 17 |
| Orbit, Sarcoma TNM7 | Orbit | 54 |
| Ovary TNM7 | Ovary | 37 |
| Pancreas, Exocrine and Endocrine TNM7 | Pancreas | 24 |
| Penis TNM7 | Penis | 40 |
| Pharynx: Hypopharynx TNM7 | Hypopharynx | 4 |
| Pharynx: Nasopharynx TNM7 | Nasopharynx | 4 |
| Pharynx: Oropharynx TNM7 | Oropharynx | 4 |
| Pharynx: Pharyngeal Tonsil TNM7 | PharyngealTonsil | 4 |
| Pleural Mesothelioma TNM7 | PleuralMesothelioma | 26 |
| Primary Peritoneal Carcinoma [Female] TNM7 | PeritoneumFemaleGen | 37 |
| Prostate TNM7 | Prostate | 41 |
| Renal Pelvis and Ureter TNM7 | RenalPelvisUreter | 44 |
| Retinoblastoma TNM7 | Retinoblastoma | 52 |
| Skin, inc. Scrotum, Squamous Cell and Other TNM7 | SkinScrotumSquamousAndOther | 29 |
| Small Intestine TNM7 | SmallIntestine | 12 |
| Soft Tissue Sarcoma TNM7 | SoftTissueSarcoma | 28 |
| Stomach TNM7 | Stomach | 11 |
| Testis TNM7 | Testis | 42 |
| Urethra TNM7 | Urethra | 46 |
| Urinary Bladder TNM7 | Bladder | 45 |
| Vagina TNM7 | Vagina | 34 |
| Vulva TNM7 | Vulva | 33 |
| Esophagogastric Junction, Adenocarcinoma TNM7 | EsophagusGEJunctionAdeno | 10 |
| Esophagogastric Junction, Squamous TNM7 | EsophagusGEJunctionSquamous | 10 |
| Esophagus, Adenocarcinoma TNM7 | EsophagusAdenocarcinoma | 10 |
| Esophagus, Squamous Cell TNM7 | EsophagusSquamous | 10 |
| Thyroid, Anaplastic TNM7 | ThyroidAnaplastic | 8 |
| Thyroid, Medullary TNM7 | ThyroidMedullary | 8 |

# Enumerated Types

TNM staging uses enumerated types to refer to values without the need for constants.

## DataElement

**DataElement** represents a data element used in the calculation of TNM stage. Most DataElements have a one-to-one correspondence with NAACCR data items. For example, CLINICAL\_T is “TNM Clin T” (Item # 940).

Some DataElements are intermediate variables used for stage calculation. These variables are only defined within the scope of the TNM staging library.

 DataElements are used to indicate a specific data element is needed:

* As input to/output from the datacard
* To select a schema when there are multiple candidates
* To retrieve a table to use as a picklist for abstraction

public enum DataElement

{

NONE = -1, CLINICAL\_T, PATHOLOGIC\_T, CLINICAL\_N, PATHOLOGIC\_N, CLINICAL\_M, PATHOLOGIC\_M,

CLINICAL\_STAGE\_GRP, PATHOLOGIC\_STAGE\_GRP, SSF1, SSF2, SSF3, SSF4, SSF5, SSF6, SSF7, SSF8,

SSF9, SSF10, SSF11, SSF12, SSF13, SSF14, SSF15, SSF16, SSF17, SSF18, SSF19, SSF20, SSF21,

SSF22, SSF23, SSF24, SSF25,

PRIMARY\_SITE, HIST\_ICD\_O\_3, YEAR\_OF\_DIAGNOSIS, TUMORSIZE, BEHAV, GRADE, AGE, SEX,

 TNM\_EDITION, TNM\_CLIN\_DESCRIPTOR, TNM\_PATH\_DESCRIPTOR, DERIVED\_CLINICAL\_STAGE\_GRP,

DERIVED\_PATHOLOGIC\_STAGE\_GRP,

B\_VALUE, S\_VALUE, GRADE\_CATEGORY, MITOTIC\_RATE

};

Details:

* The “NONE” element denotes the absence of a valid data element.
* CLINICAL\_STAGE\_GRP/PATHOLOGIC\_STAGE\_GRP is used to denote directly coded stage group, and DERIVED\_CLINICAL\_STAGE\_GRP/DERIVED\_PATHOLOGIC\_STAGE\_GRP is used to denote derived TNM stage group.
* Intermediate data elements are placed at the higher end of the range, starting with B\_VALUE. Intermediate data elements are not abstracted.

DataElements by integer value

|  |  |
| --- | --- |
| Number | Enum |
| -1 | NONE |
| 0 | CLINICAL\_T |
| 1 | PATHOLOGIC\_T |
| 2 | CLINICAL\_N |
| 3 | PATHOLOGIC\_N |
| 4 | CLINICAL\_M |
| 5 | PATHOLOGIC\_M |
| 6 | CLINICAL\_STAGE\_GRP |
| 7 | PATHOLOGIC\_STAGE\_GRP |
| 8 | SSF1 |
| 9 | SSF2 |
| 10 | SSF3 |
| 11 | SSF4 |
| 12 | SSF5 |
| 13 | SSF6 |
| 14 | SSF7 |
| 15 | SSF8 |
| 16 | SSF9 |
| 17 | SSF10 |
| 18 | SSF11 |
| 19 | SSF12 |
| 20 | SSF13 |
| 21 | SSF14 |
| 22 | SSF15 |
| 23 | SSF16 |
| 24 | SSF17 |
| 25 | SSF18 |
| 26 | SSF19 |
| 27 | SSF20 |
| 28 | SSF21 |
| 29 | SSF22 |
| 30 | SSF23 |
| 31 | SSF24 |
| 32 | SSF25 |
| 33 | PRIMARY\_SITE |
| 34 | HIST\_ICD\_O\_3 |
| 35 | YEAR\_OF\_DIAGNOSIS |
| 36 | TUMOR\_SIZE |
| 37 | BEHAV |
| 38 | GRADE |
| 39 | AGE |
| 40 | SEX |
| 41 | TNM\_EDITION |
| 42 | TNM\_CLIN\_DESCRIPTOR |
| 43 | TNM\_PATH\_DESCRIPTOR |
| 44 | DERIVED\_CLINICAL\_STAGE\_GRP |
| 45 | DERIVED\_PATHOLOGIC\_STAGE\_GRP |
| 46 | B\_VALUE |
| 47 | S\_VALUE |
| 48 | GRADE\_CATEGORY |
| 49 | MITOTIC\_RATE |

## SchemaIdentifier

SchemaIdentifier is the enumerated type for a schema identifier. Schema identifiers are used to refer to schemas when writing schema-specific logic. For example, if you want to check for the schema “Adrenal”, you would use the schema identifier Adrenal. The full list of schemas is listed in the [Schemas](#_Schemas) section.

See the descriptions for GetSchemaNumberBySchemaIdentifier() and GetSchemaBySchemaIdentifier() for information on how to retrieve schema information by identifier.

A schema identifier is **NOT** a schema number and should not be treated as such.

public enum SchemaIdentifier

{

Adrenal,

AmpullaOfVater,

Anus,

 AppendixCarcinoid,

AppendixCarcinoma,

BileDuctsCysticDuct,

BileDuctsDistal,

IntrahepaticBileDucts,

BileDuctsPerihilar,

Bone,

Breast,

CervixUteri,

ColonRectum,

ConjunctivaCarcinoma,

CorpusUteriAdenosarcoma,

CorpusUteriCarcinomas,

CorpusUteriSarcomas,

EsophagusGEJunctionAdeno,

EsophagusGEJunctionSquamous,

EsophagusAdenocarcinoma,

EsophagusSquamous,

Eyelid,

FallopianTube,

Gallbladder,

GISTColonRectumAppendix,

GISTStomach,

GISTOmentum,

GISTPeritoneum,

GISTSmallIntestineEsophagus,

GestationalTrophoblasticDisease,

Kidney,

LacrimalGland,

LarynxGlottic,

LarynxSubglottic,

LarynxSupraglottic,

LipOralCavity,

Liver,

Lung,

Lymphoma,

LymphomaCutaneousMF,

LymphomaOcularAdnexa,

SalivaryGlandsMajor,

MelanomaCiliaryBody,

MelanomaConjunctiva,

MelanomaIris,

MelanomaSkin,

MerkelCellCarcinoma,

MelanomaHeadAndNeck,

MaxillarySinus,

NasalCavityEthmoidSinus,

NeuroendocrineTumorsColonRectum,

NeuroendocrineTumorsSmIntestineAmpulla,

NeuroendocrineTumorsStomach,

Orbit,

Ovary,

Pancreas,

Penis,

Hypopharynx,

Nasopharynx,

Oropharynx,

PharyngealTonsil,

PleuralMesothelioma,

PeritoneumFemaleGen,

Prostate,

RenalPelvisUreter,

Retinoblastoma,

SkinScrotumSquamousAndOther,

SmallIntestine,

SoftTissueSarcoma,

Stomach,

Testis,

ThyroidAnaplastic,

ThyroidMedullary,

ThyroidPapFollic,

Urethra,

Bladder,

Vagina,

Vulva

};

## TNMTableRole

TNMTableRole is the enumerated type for a table role. The typical API user will use DataElement instead of TNMTableRole to interact with tables, so this section can be treated as optional.

A table’s role is its purpose in TNM staging. Some tables are used for defining picklists for input values, others are for calculating derived stage group, and others are used to derive intermediate variables that are used in calculating derived stage group.

public enum TNMTableRole

{

NONE = -1, CLINICAL\_T, PATHOLOGIC\_T, CLINICAL\_N, PATHOLOGIC\_N, CLINICAL\_M, PATHOLOGIC\_M,

CLINICAL\_STAGE, PATHOLOGIC\_STAGE, SSF1, SSF2, SSF3, SSF4, SSF5, SSF6, SSF7, SSF8, SSF9,

SSF10, SSF11, SSF12, SSF13, SSF14, SSF15, SSF16, SSF17, SSF18, SSF19, SSF20, SSF21,

SSF22, SSF23, SSF24, SSF25,

 DERIVED\_CLINICAL\_STAGE, DERIVED\_PATHOLOGIC\_STAGE, EXTRA

};

Details:

* The value “NONE” is used to indicate that a table with the desired role does not exist.
* The value “EXTRA” is used for all tables used to calculate intermediate variables.
* If tables in the same schema have identical table roles, they are differentiated by sub-role (a string value).

TNMTableRole is also used to map DataElements to tables. Most TNMTableRoles (except EXTRA) have a corresponding DataElement. Conversely, DataElements that are either inputs to or outputs from tables can be mapped to a corresponding TNMTableRole.

## ErrorCode

ErrorCode is an enumerated type that defines all the possible error conditions that can occur when using the TNM staging library.

public enum ErrorCode

{

OK,

SYSTEM\_EXCEPTION,

SCHEMA\_NOT\_APPLICABLE,

SCHEMA\_DISCRIMINATOR\_REQUIRED,

SCHEMA\_INDEX\_OUT\_OF\_BOUNDS,

TABLE\_INDEX\_OUT\_OF\_BOUNDS,

INVALID\_CLINICAL\_T,

INVALID\_CLINICAL\_N,

INVALID\_CLINICAL\_M,

INVALID\_CLINICAL\_STAGE\_GROUP,

INVALID\_PATHOLOGIC\_T,

INVALID\_PATHOLOGIC\_N,

INVALID\_PATHOLOGIC\_M,

INVALID\_PATHOLOGIC\_STAGE\_GROUP,

CLINICAL\_T\_TABLE\_NOT\_FOUND,

CLINICAL\_N\_TABLE\_NOT\_FOUND,

CLINICAL\_M\_TABLE\_NOT\_FOUND,

CLINICAL\_STAGE\_GROUP\_TABLE\_NOT\_FOUND,

PATHOLOGIC\_T\_TABLE\_NOT\_FOUND,

PATHOLOGIC\_N\_TABLE\_NOT\_FOUND,

PATHOLOGIC\_M\_TABLE\_NOT\_FOUND,

PATHOLOGIC\_STAGE\_GROUP\_TABLE\_NOT\_FOUND,

DERIVED\_STAGE\_TABLE\_NOT\_FOUND,

TABLE\_NOT\_FOUND,

STAGE\_GROUP\_NOT\_DEFINED,

STAGE\_GROUP\_YP\_NOT\_REPORTABLE,

STAGE\_GROUP\_CANNOT\_BE\_CALCULATED,

STAGE\_GROUP\_IS\_ERROR,

STAGE\_GROUP\_NOT\_FOUND,

STAGING\_ELEMENT\_MISSING,

STORAGE\_CODE\_TYPE\_NOT\_VALID,

STORAGE\_CODE\_NOT\_FOUND,

DISPLAY\_CODE\_TYPE\_NOT\_VALID,

DISPLAY\_CODE\_NOT\_FOUND,

TABLE\_ROW\_OR\_CODE\_NOT\_VALID,

NOTE\_INDEX\_NOT\_VALID,

COLUMN\_INDEX\_NOT\_VALID,

SITE\_NOT\_VALID,

HISTOLOGY\_NOT\_VALID

};

ErrorCodes by integer value

|  |  |
| --- | --- |
| Number | Enum |
| 0 | OK |
| 1 | [SYSTEM\_EXCEPTION](#SYSTEM_EXCEPTION) |
| 2 | [SCHEMA\_NOT\_APPLICABLE](#SCHEMA_NOT_APPLICABLE) |
| 3 | [SCHEMA\_DISCRIMINATOR\_REQUIRED](#SCHEMA_DISCRIMINATOR_REQUIRED) |
| 4 | [SCHEMA\_INDEX\_OUT\_OF\_BOUNDS](#SCHEMA_INDEX_OUT_OF_BOUNDS) |
| 5 | [TABLE\_INDEX\_OUT\_OF\_BOUNDS](#TABLE_INDEX_OUT_OF_BOUNDS) |
| 6 | [INVALID\_CLINICAL\_T](#INVALID_CLINICAL_T) |
| 7 | [INVALID\_CLINICAL\_N](#INVALID_CLINICAL_N) |
| 8 | [INVALID\_CLINICAL\_M](#INVALID_CLINICAL_M) |
| 9 | [INVALID\_CLINICAL\_STAGE\_GROUP](#INVALID_CLINICAL_STAGE_GROUP) |
| 10 | [INVALID\_PATHOLOGIC\_T](#INVALID_PATHOLOGIC_T) |
| 11 | [INVALID\_PATHOLOGIC\_N](#INVALID_PATHOLOGIC_N) |
| 12 | [INVALID\_PATHOLOGIC\_M](#INVALID_PATHOLOGIC_M) |
| 13 | [INVALID\_PATHOLOGIC\_STAGE\_GROUP](#INVALID_PATHOLOGIC_STAGE_GROUP) |
| 14 | [CLINICAL\_T\_TABLE\_NOT\_FOUND](#CLINICAL_T_TABLE_NOT_FOUND) |
| 15 | [CLINICAL\_N\_TABLE\_NOT\_FOUND](#CLINICAL_N_TABLE_NOT_FOUND) |
| 16 | [CLINICAL\_M\_TABLE\_NOT\_FOUND](#CLINICAL_M_TABLE_NOT_FOUND) |
| 17 | [CLINICAL\_STAGE\_GROUP\_TABLE\_NOT\_FOUND](#CLINICAL_STAGE_GROUP_TABLE_NOT_FOUND) |
| 18 | [PATHOLOGIC\_T\_TABLE\_NOT\_FOUND](#PATHOLOGIC_T_TABLE_NOT_FOUND) |
| 19 | [PATHOLOGIC\_N\_TABLE\_NOT\_FOUND](#PATHOLOGIC_N_TABLE_NOT_FOUND) |
| 20 | [PATHOLOGIC\_M\_TABLE\_NOT\_FOUND](#PATHOLOGIC_M_TABLE_NOT_FOUND) |
| 21 | [PATHOLOGIC\_STAGE\_GROUP\_TABLE\_NOT\_FOUND](#PATHOLOGIC_STAGE_GROUP_TABLE_NOT_FOUND) |
| 22 | [DERIVED\_STAGE\_TABLE\_NOT\_FOUND](#DERIVED_STAGE_TABLE_NOT_FOUND) |
| 23 | [TABLE\_NOT\_FOUND](#TABLE_NOT_FOUND) |
| 24 | [STAGE\_GROUP\_NOT\_DEFINED](#STAGE_GROUP_NOT_DEFINED) |
| 25 | [STAGE\_GROUP\_YP\_NOT\_REPORTABLE](#STAGE_GROUP_YP_NOT_REPORTABLE) |
| 26 | [STAGE\_GROUP\_CANNOT\_BE\_CALCULATED](#STAGE_GROUP_CANNOT_BE_CALCULATED) |
| 27 | [STAGE\_GROUP\_IS\_ERROR](#STAGE_GROUP_IS_ERROR) |
| 28 | [STAGE\_GROUP\_NOT\_FOUND](#STAGE_GROUP_NOT_FOUND) |
| 29 | [STAGING\_ELEMENT\_MISSING](#STAGING_ELEMENT_MISSING) |
| 30 | [STORAGE\_CODE\_TYPE\_NOT\_VALID](#STORAGE_CODE_TYPE_NOT_VALID) |
| 31 | [STORAGE\_CODE\_NOT\_FOUND](#STORAGE_CODE_NOT_FOUND) |
| 32 | [DISPLAY\_CODE\_TYPE\_NOT\_VALID](#DISPLAY_CODE_TYPE_NOT_VALID) |
| 33 | [DISPLAY\_CODE\_NOT\_FOUND](#DISPLAY_CODE_NOT_FOUND) |
| 34 | [TABLE\_ROW\_OR\_CODE\_NOT\_VALID](#TABLE_ROW_OR_CODE_NOT_VALID) |
| 35 | [NOTE\_INDEX\_NOT\_VALID](#NOTE_INDEX_NOT_VALID) |
| 36 | [COLUMN\_INDEX\_NOT\_VALID](#COLUMN_INDEX_NOT_VALID) |
| 37 | [SITE\_NOT\_VALID](#SITE_NOT_VALID) |
| 38 | [HISTOLOGY\_NOT\_VALID](#HISTOLOGY_NOT_VALID) |

## Explanation of codes

The following table describes the error codes returned by the TNM staging library, organized by category.

Each code is accompanied by:

* An error message (which can be retrieved by calling a API function)
* A description of the error condition
* An explanation of whether the code represents an error (“E”) or non-error information (“I”)
* Suggested actions for the API user

| **Error Code** | **Error message** | **Explanation** | **Error or Informa-tional** | **Suggested behavior of calling program** |
| --- | --- | --- | --- | --- |
| **System Exceptions** |
| SYSTEM\_EXCEPTION | “A system exception has occurred. Check the log file ‘system\_exception.txt’ for more details.” | The library has caught an exception that would crash the library if not handled. | E | Check the log file and inform the developer. |
|  **Codes Produced During Schema Selection** |
| SCHEMA\_NOT\_APPLICABLE | "Schema not applicable." | A schema could not be identified from inputs (either from not being mapped, or from specifically being excluded) | I | Provide message to user that TNM staging is not applicable for this diagnosis and populate TNM fields with defaults for Not Applicable (generally 88). |
| SCHEMA\_DISCRIMINATOR\_REQUIRED | “A valid schema discriminator is required.” | A schema discriminator is required to identify a schema, but a valid discriminator value was not supplied. A discriminator value is valid if it has been included in a schema definition. | E | Present user with data item for discriminator and when value is selected, add it to the datacard and repeat function calls until valid value allows selection of schema. |
| SITE\_NOT\_VALID | “Site code not valid.” | An invalid site code was supplied. | E | Prompt user to enter a valid site code, update the datacard, and call the function again. |
| HISTOLOGY\_NOT\_VALID | “Histology code not valid.” | An invalid histology code was supplied. |  | Prompt user to enter a valid histology code, update the datacard, and call the function again. |
| **Codes Produced during Data Access** |
| SCHEMA\_INDEX\_OUT\_OF\_BOUNDS | "Schema index out of bounds." | Schema index is not valid (< 0 or >= number of schemas). | E | Report to developer. |
| TABLE\_INDEX\_OUT\_OF\_BOUNDS | "Table index out of bounds." | Table index is not valid (< 0 or >= number of tables for the schema). | E | Report to developer. |
| TABLE\_ROW\_OR\_CODE\_NOT\_VALID | "Table row or code not valid." | The specified table row or column is invalid. | E | Report to developer. |
| TABLE\_NOT\_FOUND | “Table not found.” | The specified table could not be found. | E | Report to developer. |
| NOTE\_INDEX\_NOT\_VALID | “Note index not valid.” | The specified note index is invalid. | E | Report to developer. |
| COLUMN\_INDEX\_NOT\_VALID | “Column index not valid.” | The specified column index is invalid. | E | Report to developer. |
| **Codes Produced Pre-Derivation** |
| INVALID\_CLINICAL\_T | "Invalid Clinical T value." | The value was not found in the associated table.  | E | Provide message to user that value entered is not valid and allow user to enter a new value. Then re-derive. |
| INVALID\_CLINICAL\_N | "Invalid Clinical N value." | E |
| INVALID\_CLINICAL\_M | "Invalid Clinical M value." | E |
| INVALID\_CLINICAL\_STAGE\_GROUP | “Invalid Clinical Stage Group value” | E |
| INVALID\_PATHOLOGIC\_T | "Invalid Pathologic T value." | E |
| INVALID\_PATHOLOGIC\_N | "Invalid Pathologic N value." | E |
| INVALID\_PATHOLOGIC\_M | "Invalid Pathologic M value." | E |
| INVALID\_PATHOLOGIC\_STAGE\_GROUP | “Invalid Pathologic Stage Group value” | E |
| CLINICAL\_T\_TABLE\_NOT\_FOUND | "Clinical T table not found." | The indicated table could not be found in the schema. This is a system error – every schema should have one of each. | E | Report to developer. |
| CLINICAL\_N\_TABLE\_NOT\_FOUND | "Clinical N table not found." | E |
| CLINICAL\_M\_TABLE\_NOT\_FOUND | "Clinical M table not found." | E |
| CLINICAL\_STAGE\_GROUP\_TABLE\_NOT\_FOUND | “Clinical Stage Group table not found” | E |
| PATHOLOGIC\_T\_TABLE\_NOT\_FOUND | "Pathologic T table not found." | E |
| PATHOLOGIC\_N\_TABLE\_NOT\_FOUND | "Pathologic N table not found." | E |
| PATHOLOGIC\_M\_TABLE\_NOT\_FOUND | "Pathologic M table not found." | E |
| PATHOLOGIC\_STAGE\_GROUP\_TABLE\_NOT\_FOUND | “Pathologic Stage Group not found.” | E |
| **Codes Produced During Calculation/Derivation** |
| DERIVED\_STAGE\_TABLE\_NOT\_FOUND | "Appropriate derived stage table not found." | An appropriate stage table could not be found. This can be due to missing values that are required to select a stage table. | E | Calling program must identify required data items and notify user that an element was missing and allow user to enter a value for the missing element. Then repeat derivation function. |
| STAGE\_GROUP\_NOT\_DEFINED | “Stage Groups are not defined for this schema.” | The schema is defined and T/N/M values can be supplied, but stage group is not calculated. | I | Provide message to user that no stage groups are defined and populate NPCR Derived Clin Stg Grp and NPCR Derived Path Stg Grp fields with default for Not Applicable (generally 88). |
| STAGE\_GROUP\_YP\_NOT\_REPORTABLE | “yP stage is not collected by NPCR.” | A pathologic stage group value is not calculated for cases with TNM Path Descriptor = 4 or 6. | I | Provide message to user that yP stage group is not collected by NPCR and populate NPCR Derived Path Stg Grp with default for Unknown (99). |
| STAGE\_GROUP\_CANNOT\_BE\_CALCULATED | "Stage group value cannot be calculated." | A stage group value cannot be calculated because of specific business rules. | E | Provide message to user that combination of T, N, and M values does not allow derivation of stage group and user needs to enter different values(s). Then call derivation function again. |
| STAGE\_GROUP\_IS\_ERROR | "Combination of TNM is logically or medically not possible.” | A stage group value of “ERROR” was obtained – the combination of input values is logically or medically not possible, for example, in situ tumor with mets. | E | Provide message to user that combination of T, N, and M values produces an error and ask user to change the T, N, and/or M values. Then call derivation function again. |
| STAGE\_GROUP\_NOT\_FOUND | "Stage group value not found in table." | A stage group value could not be found using the provided combination of input values. | I | Provide message to user that combination of T, N, and M values is an unknown stage group and populate NPCR Derived Clin Stg Grp and/or NPCR Derived Path Stg Grp with default for Unknown (99). |
| STAGING\_ ELEMENT\_MISSING | “Data item needed to derive stage group is missing.” | A data item needed to derive stage group is blank.  | I | Provide message to user that a data item needed to derive stage group(s) is blank and ask user if this is OK. If user responds Yes, populate NPCR Derived Clin Stg Grp and/or NPCR Derived Path Stg Grp stage group(s) with default for Unknown (99). If user responds No, allow user to fill in missing item and re-derive. |
| **Codes Produced During Data Presentation** |
| STORAGE\_CODE\_TYPE\_NOT\_VALID | "Storage code type not valid." | The specified storage type code does not exist.  | E | Calling program needs to provide a valid code. User cannot do this. |
| STORAGE\_CODE\_NOT\_FOUND | "Storage code not found." | The specified storage code does not exist. | E | Calling program needs to provide a valid code. User cannot do this. |
| DISPLAY\_CODE\_TYPE\_NOT\_VALID | "Display code type not valid." | The specified display type code does not exist. | E | Calling program needs to provide a valid code. User cannot do this. |
| DISPLAY\_CODE\_NOT\_FOUND | "Display code not found." | The specified display code does not exist. | E | Calling program needs to provide a valid code. User cannot do this. |

# Notable Classes

This section contains a description of the relevant classes used by the staging library.

## Datacard

The Datacard class stores the input and output fields used in staging. When the user populates the required input fields with values and calls the stage calculation function, the output fields will be populated with the derived values.

Note: all strings are initially of indeterminate length. After stage calculation, the output fields will contain values of the correct field length.

Note: The fields for directly coded clinical/pathologic stage are named “clinical\_stage” and “path\_stage”. The fields for derived clinical/pathologic stage are named “derived\_clinical\_stage” and “derived\_path\_stage”.

public class Datacard

{

public string site;

public string histology;

public string diagnosis\_year;

public string age;

public string behavior;

public string grade;

public string sex;

public string tnm\_edition\_num;

public string tnm\_clin\_descriptor;

public string tnm\_path\_descriptor;

public string clinical\_stage;

public string path\_stage;

public string ssf1;

public string ssf2;

public string ssf3;

public string ssf4;

public string ssf5;

public string ssf6;

public string ssf7;

public string ssf8;

public string ssf9;

public string ssf10;

public string ssf11;

public string ssf12;

public string ssf13;

public string ssf14;

public string ssf15;

public string ssf16;

public string ssf17;

public string ssf18;

public string ssf19;

public string ssf20;

public string ssf21;

public string ssf22;

public string ssf23;

public string ssf24;

public string ssf25;

public string clinical\_T;

public string clinical\_N;

public string clinical\_M;

public string path\_T;

public string path\_N;

public string path\_M;

public string derived\_clinical\_stage;

public string derived\_path\_stage;

};

Each member variable of Datacard corresponds to a NAACCR field. The following table describes the Datacard layout.

NOTE:  The items named SSF 1-25 were imported into the TNM DLL from the Collaborative Stage system, version 0205.  Only the SSFs required for TNM staging were imported.  Pick lists for these SSFs generated from the TNM DLL will differ in one way from the CS originals:  code 988 (Not applicable) is not included because these SSFs are required for staging.  Code 988 will appear in pick lists generated from the CS DLL for the same items.

| Member variable | NAACCR Field Name | Item # | Columns | Field Length |
| --- | --- | --- | --- | --- |
| Input variables |
| site | Primary Site | 400 | 540-543 | 4 |
| histology | Histologic Type ICD-O-3 | 522 | 550-553 | 4 |
| diagnosis\_year | Date of Diagnosis (Year portion only) | 390 | 530-533 (Year columns only) | 4 |
| age | Age at Diagnosis | 230 | 193-195 | 3 |
| behavior | Behavior Code ICD-O-3 | 523 | 554-554 | 1 |
| grade | Grade | 440 | 555-555 | 1 |
| sex | Sex | 220 | 192-192 | 1 |
| tnm\_edition\_num | TNM Edition Number | 1060 | 938-939 | 2 |
| tnm\_clin\_descriptor | TNM Clin Descriptor | 980 | 974-974 | 1 |
| tnm\_path\_descriptor | TNM Path Descriptor | 920 | 956-956 | 1 |
| clinical\_stage | TNM Clin Stage Group | 970 | 970-973 | 4 |
| path\_stage | TNM Path Stage Group | 910 | 952-955 | 4 |
| ssf1 | CS Site-Specific Factor 1 | 2880 | 1003-1005 | 3 |
| ssf2 | CS Site-Specific Factor 2 | 2890 | 1006-1008 | 3 |
| ssf3 | CS Site-Specific Factor 3 | 2900 | 1009-1011 | 3 |
| ssf4 | CS Site-Specific Factor 4 | 2910 | 1012-1014 | 3 |
| ssf5 | CS Site-Specific Factor 5 | 2920 | 1015-1017 | 3 |
| ssf6 | CS Site-Specific Factor 6 | 2930 | 1018-1020 | 3 |
| ssf7 | CS Site-Specific Factor 7 | 2861 | 1021-1023 | 3 |
| ssf8 | CS Site-Specific Factor 8 | 2862 | 1024-1026 | 3 |
| ssf9 | CS Site-Specific Factor 9 | 2863 | 1027-1029 | 3 |
| ssf10 | CS Site-Specific Factor 10 | 2864 | 1030-1032 | 3 |
| ssf11 | CS Site-Specific Factor 11 | 2865 | 1033-1035 | 3 |
| ssf12 | CS Site-Specific Factor 12 | 2866 | 1036-1038 | 3 |
| ssf13 | CS Site-Specific Factor 13 | 2867 | 1039-1041 | 3 |
| ssf14 | CS Site-Specific Factor 14 | 2868 | 1042-1044 | 3 |
| ssf15 | CS Site-Specific Factor 15 | 2869 | 1045-1047 | 3 |
| ssf16 | CS Site-Specific Factor 16 | 2870 | 1048-1050 | 3 |
| ssf17 | CS Site-Specific Factor 17 | 2871 | 1051-1053 | 3 |
| ssf18 | CS Site-Specific Factor 18 | 2872 | 1054-1056 | 3 |
| ssf19 | CS Site-Specific Factor 19 | 2873 | 1057-1059 | 3 |
| ssf20 | CS Site-Specific Factor 20 | 2874 | 1060-1062 | 3 |
| ssf21 | CS Site-Specific Factor 21 | 2875 | 1063-1065 | 3 |
| ssf22 | CS Site-Specific Factor 22 | 2876 | 1066-1068 | 3 |
| ssf23 | CS Site-Specific Factor 23 | 2877 | 1069-1071 | 3 |
| ssf24 | CS Site-Specific Factor 24 | 2878 | 1072-1074 | 3 |
| ssf25 | CS Site-Specific Factor 25 | 2879 | 1075-1077 | 3 |
| clinical\_T | TNM Clin T | 940 | 958-961 | 4 |
| clinical\_N | TNM Clin N | 950 | 962-965 | 4 |
| clinical\_M | TNM Clin M | 960 | 966-969 | 4 |
| path\_T | TNM Path T | 880 | 940-943 | 4 |
| path\_N | TNM Path N | 890 | 944-947 | 4 |
| path\_M | TNM Path M | 900 | 948-951 | 4 |
| Output variables |
| derived\_clinical\_stage | NPCR Derived Clin Stg Grp | 3650 | 896-899 | 4 |
| derived\_path\_stage | NPCR Derived Path Stg Grp | 3655 | 900-903 | 4 |

Notable Methods:

**GetDatacardValue**

String GetDatacardValue(DataElement element)

The function retrieves the value of the Datacard field for the given input DataElement.

**PutDatacardValue**

String PutDatacardValue(DataElement element, string value)

The function sets the given Datacard field to the given value.

**Clear**

Void Clear()

The function clears the Datacard fields.

## TableInfo

TableInfo holds information about a DataElement and an associated table. A TableInfo object is used to provide information for a user to retrieve a table from the TNM staging library, or to identify a data item whose picklist must be constructed from an external source instead.

**Note:** The TNM staging library does not provide picklists for items that are not TNM-specific and are commonly abstracted (ex: Age, Behavior, Grade, Sex). They must be constructed from external sources, such as the NAACCR manual or the vendor’s own sources. The reasoning is that these data items will need to be entered anyway, and should not be entered more than once.

public class TableInfo

{

public int tableNumber;

public DataElement dataElement;

public int naaccrItemNumber;

}

Member variables:

* tableNumber – the number of the associated table. The user can call GetTable() with the schema number and table number to retrieve the table.
	+ If the value is -1, there is no associated table in the TNM staging library. The user should use the value of dataElement and naaccrItemNumber instead to identify the correct data item and create a table from an external source.
* dataElement – the DataElement that the table is used to code for.
* naaccrItemNumber – the NAACCR Item # of the associated DataElement.

## TNMTable

A TNMTable holds the data for a schema table.

Schema tables have multiple purposes:

* converting input storage codes to display codes
* displaying picklists
* looking up stage group values
* calculating intermediate values.

API users will mainly use TNMTables to display picklists.

public class TNMTable

{

public string Title;

public string Subtitle;

public List<string> Notes;

public DiagnosisMode DiagnosisMode;

public TNMTableRole Role;

public string SubRole;

public TableLookupType LookupType;

public string ResultVar;

public List<TNMTableColumn> Columns;

public List<TNMTableRow> Rows;

}

Member Variables:

* Title: the table title
* Subtitle: the table subtitle
* Notes: a list of table notes. Notes provide instructions and clarification to the abstracter.
* DiagnosisMode: describes whether the table is used for clinical staging, pathologic staging, or both.
* Role: the table’s purpose in its parent schema
* SubRole: the table sub-role. Used to distinguish tables that share the same table role.
* LookupType: describes how to perform lookup in the table.
* ResultVar: the name of the data element returned by the table, if one exists
* Columns: a list of table columns. Contains the table column headers.
* Rows: a list of table rows. Contains the table data.

Notable Methods:

**GetValue**

ErrorCode GetValue(int row, int column, out string value)

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Row | Int | The row index (starting from 0) |
| Column | Int | The column index (starting from 0) |
| Value | String (out) | Receives the value of the indicated table cell |

The function takes a row index and column index and sets *value* to the value of the indicated table cell. If the row number or column number is invalid, the function returns TABLE\_ROW\_OR\_CODE\_NOT\_VALID.

By iterating over each column and row and calling this function for every cell, the API user can replicate the structure of a table in order to create a picklist.

**GetColumnTitle**

ErrorCode GetColumnTitle(int index, out string value)

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Index | Int | The column index (starting from 0) |
| Value | String (out) | Receives the value of the indicated column title |

The function takes a column index and sets *value* to the value of the associated table column. If the column index is invalid, the function returns COLUMN\_INDEX\_NOT\_VALID.

**NumColumns**

Int NumColumns()

The function returns the number of table columns.

**NumRows**

Int NumRows()

The function returns the number of table rows.

**GetNote**

ErrorCode GetNote(int index, out string value)

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Index | Int | The note index (starting from 0) |
| Value | String (out) | Receives the value of the associated note |

The function takes a note index and sets *value* to the value of the associated note. If the note index is invalid, the function returns NOTE\_INDEX\_NOT\_VALID.

Notes contain instructions for the abstractor.

**NumNotes**

Int NumNotes()

The function returns the number of notes.

## TNMSchema

A TNMSchema holds the data for a schema, a collection of site and histology codes that are staged with a common set of stage group values.

TNM staging schemas and Collaborative Stage schemas belong to distinct staging systems, so they cannot be treated interchangeably.

Schemas are ordered by schema number. By accessing schemas, the API user can obtain tables in order to create picklists.

public class TNMSchema

{

public string ID;

public string Title;

public string TNMChapter;

public List<SiteHistGrouping> Definition;

public List<string> Notes;

public List<TNMTable> Tables;

public Dictionary<DataElement, TNMTableRole> IntermediateVarMap;

public List<DataElement> requiredDataElementsForSchemaSelection;

}

Member variables:

* ID: a short identifier
* Title: the schema display name
* TNMChapter: AJCC chapter number
* Definition: a collection of site and histology code combinations that define the schema
* Tables: a List of tables
* IntermediateVarMap: a data structure containing information about intermediate variables used by the schema
* requiredDataElementsForSchemaSelection: a List of data elements (discriminators) that are necessary in order to select the schema

Notable Methods:

**RequiredDataItems**

List<[TableInfo](#_TableInfo)> RequiredDataItems()

The function returns the list of data items used to stage the schema, including both site-specific factors and non-site-specific factors. Primary site, histologic type, and schema discriminators are excluded from the list, as they have already been used to select the schema by the time the function is called.

Site-specific factors have schema tables which can be used as picklists; non-site-specific factors do not, and therefore must be coded from other sources, so to assist in the creation of picklists, separate functions for each type have been provided.

**SiteHistGrouping**

SiteHistGrouping is a helper class for defining a schema. A SiteHistGrouping contains the set of site codes, histology codes, and discriminator codes such that all codes in the grouping map to the same schema. A schema definition may consist of multiple SiteHistGroupings.

public SiteHistGrouping()

{

siteCodes = new List<string>();

histologyCodes = new List<string>();

discriminators = new List<string>();

notApplicable = false;

}

Member variables:

* siteCodes – the list of site codes in the grouping
* histologyCodes – the list of histology codes in the grouping
* discriminators – the list of discriminator codes in the grouping. The list will be empty if a discriminator is not needed. Does not identify the schema discriminator – that is done by GetSchemaNumber().
* notApplicable – Boolean flag. False if the grouping belongs to the schema definition; true if the grouping is excluded from the schema definition. (Some groupings are used to exclude codes from a schema definition; this usage is fairly rare).

## TNM\_Staging

All API functions are accessed from a class called **TNM\_Stage.** To access the API, create an instance of TNM\_Stage and dispose of it once it is no longer needed.

The API is described in detail in the section “API reference”.

# API reference

The API reference section contains a description of all the API functions. It is organized by general functionality, with a brief explanation preceding each section.

## General information

### GetVersion

String GetVersion()

The function returns the library version.

## Data transfer

The TNM staging library handles input and output via a Datacard. For existing records, a typical workflow is to extract a record from a flat file or database, store the record in a string buffer, transfer the record data to a Datacard, call the stage calculation function, and transfer the results from the Datacard to the buffer. To facilitate transfer between a buffer and a Datacard, a pair of functions have been provided.

### PopulateDatacardFromBuffer

[ErrorCode](#_ErrorCode) PopulateDatacardFromBuffer(ref [Datacard](#_Datacard) dc, string record)

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Dc | Datacard (ref) | Datacard that receives values from the record buffer |
| Record | String | Buffer used to populate the datacard |

The function populates the input datacard from the record buffer.

The function returns SYSTEM\_EXCEPTION if an exception occurs, and OK otherwise.

### PopulateBufferFromDatacard

[ErrorCode](#_ErrorCode) PopulateBufferFromDatacard([Datacard](#_Datacard) dc, ref string buffer)

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Dc | Datacard | Datacard used to populate the record buffer |
| Buffer | String (ref) | Buffer that receives values from the datacard |

The function populates the record buffer from the input datacard.

If the buffer is shorter than the NAACCR 16 standard record length, it will be padded with spaces before being populated.

The functions return SYSTEM\_EXCEPTION if an exception occurs, and OK otherwise.

## Formatting Error Messages

Functions in this section are used to format and display data for the abstractor.

The TNM staging library uses ErrorCodes to convey success or failure for API calls. Each ErrorCode can be converted to a descriptive error message.

### FormatErrorCode

String FormatErrorCode([ErrorCode](#_ErrorCode) error)

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Error | ErrorCode | The error code to be translated |

The function takes an input ErrorCode and returns the associated message string. If an API function returns an error code, the code can be passed to this function to obtain a descriptive message.

The full set of ErrorCodes and messages are detailed in the [ErrorCodes](#_ErrorCode) section.

## Formatting Storage and Display codes

The TNM staging library also provides functions to convert between storage codes and display codes. For the purposes of this API, the values stored in the NAACCR record are referred to as **storage codes**. The values that are readable by people are referred to as **display codes**. Both are defined in the AJCC manual. As abstractors generally work directly with storage codes, using the conversion functions is optional.

The TNM staging library uses the revised codes for clinical and pathologic AJCC T, N, and M defined in the NAACCR 16 manual.

Example codes (Clinical N):

**Storage code:** c2A

**Display code:** cN2A

### GetStorageCode

[ErrorCode](#_ErrorCode) GetStorageCode(string displayCode, int codeType, ref string storageCode)

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| displayCode | String | Input display code |
| codeType | int | Indicates the type of display code |
| storageCode | String (ref) | Receives the value of the storage code corresponding to the display string |

The function takes a display code and an integer indicating the display code’s field type, and assigns the equivalent storage code to *storageCode*.

Valid values for codeType:

|  |  |
| --- | --- |
| Value | Field |
| 0 | Clinical T |
| 1 | Clinical N |
| 2 | Clinical M |
| 3 | Derived Clinical Stage |
| 4 | Pathologic T |
| 5 | Pathologic N |
| 6 | Pathologic M |
| 7 | Derived Pathologic Stage |

If *codeType* is invalid, the function returns STORAGE\_CODE\_TYPE\_NOT\_VALID. If *codeType* is valid but *displayCode* is invalid, the function returns STORAGE\_CODE\_NOT\_FOUND.

### GetDisplayCode

[ErrorCode](#_ErrorCode) GetDisplayCode(string storageCode, int codeType, ref string displayCode)

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| storageCode | String | Input storage code |
| codeType | Int | Indicates the type of storage code |
| displayCode | String (ref) | Receives the value of the display code corresponding to the storage code |

The function takes a storage code and an integer indicating the storage code’s field type, and assigns the equivalent display code to *displayCode*.

If *codeType* is invalid, the function returns DISPLAY\_CODE\_TYPE\_NOT\_VALID. If *codeType* is valid but *storageCode* is invalid, the function returns DISPLAY\_CODE\_NOT\_FOUND.

See GetStorageCode for a description of valid codeTypes.

## Schema selection and access

Selecting a schema is an important part of the abstraction process. A schema must be selected before any TNM fields can be abstracted. In order to select a schema, values for primary site and histologic type (and if necessary, a schema discriminator) must be entered. Once these values have been obtained, they can be passed to the API to obtain a schema number in order to access a schema’s data.

A schema discriminator is an additional data element, necessary to distinguish between schemas that share a common set of primary site and histologic type codes. The identity of the discriminator varies from group to group.

If the TNM staging library’s schema selection function is called and a schema discriminator is required, the function returns a specific error code and the required data element. The API user can then access the appropriate schema table to create a picklist or generate a picklist using content from the NAACCR manual, present the picklist to the abstractor to select a value, then call the schema selection function again with the updated information.

### GetSchemaNumber

[ErrorCode](#_ErrorCode) GetSchemaNumber(ref [Datacard](#_Datacard) dc, ref int schemaNumber, ref [DataElement](#_DataElement) requiredElement)

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Dc | Datacard (ref) | Input datacard used to determine the schema number |
| schemaNumber | Int (ref) | Receives the number of the identified schema (starting from 0) |
| requiredElement | DataElement (ref) | Receives the DataElement type of the schema discriminator |

The function takes an input Datacard and uses its value to calculate a schema number, if possible. It returns an error code indicating if the calculation was successful.

Function return values:

* **OK**: a schema was successfully selected.
* **SITE\_ NOT\_VALID**: an invalid site code was supplied.
* **HISTOLOGY\_NOT\_VALID**: an invalid histology code was supplied.
* **SCHEMA\_NOT\_APPLICABLE**: a schema could not be selected using the input data because there is no schema for that combination of primary site and histologic type.
* **SCHEMA\_DISCRIMINATOR\_REQUIRED**: a schema could not be selected using the input data because a schema discriminator value is required and was not supplied.

Some schemas share a common set of primary site and histology codes. To distinguish between these schemas, an additional data element, a **schema discriminator,** is required. If a schema discriminator is required to select a schema, *requiredElement* will be set to the DataElement corresponding to the discriminator. This is done whether schema selection was successful or not.

If the function returns SCHEMA\_DISCRIMINATOR\_REQUIRED, then a valid discriminator value was not supplied to the function, and the value of *requiredElement* should be checked.

The API user can retrieve a table to use as picklist by calling the function GetTableByDataElement() with *requiredElement* set to the value of the desired DataElement. If a table is successfully retrieved, it can be used as a picklist. Otherwise, the API user must construct a picklist using the valid values defined in the NAACCR manual.

**Note**: the schema number returned by this function is only valid while the values of primary site, histologic type, and discriminator remain constant. If one of these fields is changed during the abstracting process, it can cause the selected schema to change. If one of these fields changes, the vendor software should call GetSchemaNumber() again and take appropriate action if the returned schema number changes.

When abstracting, the workflow of selecting a schema is as follows:

* Abstractor enters the year of diagnosis. If the year is 2016 or later, continue with TNM staging. Otherwise, exit this workflow and continue abstracting non-TNM data items.
* Abstractor enters values for primary site and histologic type
* Vendor software calls GetSchemaNumber() to identify a schema
	+ If the function returns a value of SCHEMA\_DISCRIMINATOR\_REQUIRED, continue to the next step.
	+ Otherwise, a valid schema number has been obtained, or a schema number cannot be obtained.
		- If the function returns a value of OK: a valid schema number has been obtained. Exit this workflow and continue abstracting TNM data items.
		- If the function returns a value of SCHEMA\_NOT\_APPLICABLE: TNM staging cannot be performed. Exit this workflow and fill in appropriate default values for TNM derived clinical/pathologic stage group.
* Vendor software calls GetTableByDataElement(), passing in the value of *requiredElement*, the schema discriminator.
	+ If the function returns a valid table, that table should be used to construct a picklist and displayed to the abstractor. See the section for TNMTable for useful functions.
	+ Otherwise, a picklist for *requiredElement* must be created from the field’s definition in the NAACCR 16 manual.
* The abstractor chooses a value from the picklist.
* Vendor software assigns the selected value to the Datacard.
* Vendor software calls GetSchemaNumber() again. Handle the result in the same way as the previous call (loop if the result is SCHEMA\_DISCRIMINATOR\_REQUIRED; finish otherwise.)

### GetSchemaNumberBySchemaIdentifier

[ErrorCode](#_ErrorCode) GetSchemaNumberBySchemaIdentifier([SchemaIdentifier](#_SchemaIdentifier) id)

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Id | SchemaIdentifier | The identifier for the requested schema |

The function returns the schema number of the schema that matches the input id. If for some reason there is no matching schema, the function returns -1.

### GetSchemaName

[ErrorCode](#_ErrorCode) GetSchemaName(int schemaNumber, ref string schemaName)

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| schemaNumber | Int | Input schema number |
| schemaName | String (ref) | Receives the name of the schema associated with the schema number |

The function takes a schema number and sets *schemaName* to the display name of the schema with the associated schema number.

If the schema number is invalid, the function returns SCHEMA\_INDEX\_OUT\_OF\_BOUNDS.

### GetNumberOfSchemas

Int GetNumberOfSchemas()

The function returns the total number of schemas in the TNM staging library. This value can be used in conjunction with GetSchema() to iterate through the list of schemas.

### GetSchema

[ErrorCode](#_ErrorCode) GetSchema(int schemaNumber, ref [TNMSchema](#_TNMSchema) schema)

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| schemaNumber | Int | Input schema number |
| schema | TNMSchema (ref) | Receives a copy of the schema associated with the schema number |

The function takes a schema number and assigns a copy of the associated schema to parameter *schema*. The caller is responsible for disposing of the schema object.

The function returns an ErrorCode indicating if the schema number is valid. If the schema number is invalid (less than 0 or greater than or equal to the total number of schemas), the function returns SCHEMA\_INDEX\_OUT\_OF\_BOUNDS and does not modify the *schema* parameter.

### GetSchemaBySchemaIdentifier

[ErrorCode](#_ErrorCode) GetSchemaBySchemaIdentifier([SchemaIdentifier](#_SchemaIdentifier) id, ref [TNMSchema](#_TNMSchema) schema)

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| id | SchemaIdentifier | The id of the requested schema |
| schema | TNMSchema | Receives a copy of the requested schema |

The function takes a SchemaIdentifier and assigns a copy of the associated schema to parameter *schema*.

The function returns ErrorCode OK if the SchemaIdentifier is valid. If the SchemaIdentifier is invalid, the function returns SCHEMA\_INDEX\_OUT\_OF\_BOUNDS and does not modify the *schema* parameter.

## Table access

This section contains functions for retrieving schema tables.

### GetNumberOfTables

[ErrorCode](#_ErrorCode) GetNumberOfTables(int schemaNumber, ref int numTables)

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| schemaNumber | Int | Input schema number |
| numTables | Int (ref) | Receives the number of tables in the associated schema |

The function takes a schema number and sets *numTables* to the number of tables that the associated schema contains.

The function returns an ErrorCode indicating if the schema number is valid. If the schema number is invalid (less than 0 or greater than or equal to the total number of schemas), the function returns SCHEMA\_INDEX\_OUT\_OF\_BOUNDS.

Alternatively, the user can access the schema object directly to obtain the number of schema tables.

### GetTable

[ErrorCode](#_ErrorCode) GetTable(int schemaNumber, int tableNumber, ref [TNMTable](#_TNMTable) table)

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| schemaNumber | Int | Input schema number (starting from 0) |
| tableNumber | Int | Input table number (starting from 0) |
| Table | TNMTable (ref) | Receives a copy of the associated table |

The function takes a schema number and a table number, and assigns a copy of the associated table to parameter *table*. The caller is responsible for disposing of the table object.

The function returns an ErrorCode indicating if the schema number and table number are valid. If the schema number is invalid (less than 0, or greater than or equal to the total number of schemas), the function returns SCHEMA\_INDEX\_OUT\_OF\_BOUNDS. If the table number is invalid (less than 0, or greater than or equal to the number of tables for the schema), the function returns TABLE\_INDEX\_OUT\_OF\_BOUNDS.

### GetTableByDataElement

[ErrorCode](#_ErrorCode) GetTableByDataElement(int schemaNumber, [DataElement](#_DataElement) dataElement, ref [TNMTable](#_TNMTable) table)

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| schemaNumber | Int | Input schema number (starting from 0) |
| dataElement | DataElement | Input data element |
| Table | TNMTable (ref) | Receives a copy of the associated table |

The function takes a schema number and a data element and assigns a copy of the associated table to parameter *table*. If there isn’t an associated table for the schema and data element, the function returns TABLE\_NOT\_FOUND and sets *table* to null.

If the user knows the table’s number, the user should call GetTable() instead.

This function only works for input tables.

### GetTableByRole

[ErrorCode](#_ErrorCode) GetTableByRole(int schemaNumber, [TNMTableRole](#_TNMTableRole) tableRole, ref [TNMTable](#_TNMTable) table)

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| schemaNumber | Int | Input schema number (starting from 0) |
| tableRole | TNMTableRole | Input table role |
| Table | TNMTable (ref) | Receives a copy of the associated table |

The function takes a schema number and a table role and assigns a copy of the associated table to parameter *table*.

The function returns an ErrorCode indicating if the schema number and table role are valid. If the schema number is invalid, the function returns SCHEMA\_INDEX\_OUT\_OF\_BOUNDS. If a table with the indicated role is not present in the schema, the function returns TABLE\_NOT\_FOUND.

This function performs a role similar to GetTableByDataElement(), except that all tables have a role, but not every table has an associated data element. The average API user does not need to use this function, but it has been included for completeness sake.

## Coding required data elements

The set of data items needed to derive stage varies from schema to schema. The API has a function to retrieve these data items so their values can be entered and passed to the stage calculation function.

The list of required data items excludes schema discriminators, as the API user must have already selected a schema before obtaining the list of data items required for staging. For the same reason, the list of required data items excludes primary site and histologic type, because they have already been used to select a schema.

The API only indicates which data items are required for staging. Collection requirements are dependent on the standard setter and are outside the scope of the project.

Once the list of required data elements has been obtained, the appropriate picklists should be created and displayed to the abstractor if they have not already been.

### GetRequiredDataItems

List<[TableInfo](#_TableInfo)> GetRequiredDataItems(int schemaNumber)

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| schemaNumber | Int | Input schema number (starting from 0) |

The function takes a schema number and returns the list of data items that are required to stage the associated schema. If the schema number is invalid, or the schema has no required data elements, the function returns an empty List.

The function returns the required items as a list of [TableInfo](#_TableInfo). This allows the function to mix data items that have mapped TNM tables with ones that do not. The user should examine each TableInfo to check if it has a valid table number. If the table number is valid, the user can call GetTable() to retrieve the table. Otherwise, the user must identify the data item using a DataElement and NAACCR item #, and build their own picklist.

## Stage calculation

The primary purpose of the TNM staging library is to derive TNM stage group. A function called CalculateStage has been provided for stage derivation. The remainder of the API is geared toward gathering the values necessary to derive stage and interpreting the results.

The TNM staging library attempts to derive as much information as possible. If a schema could not be selected because there is insufficient information, or a schema does not exist for the given inputs, CalculateStage will indicate as such. If a schema can be selected but is not stageable, CalculateStage will indicate that as well. Clinical and pathologic staging are handled separately, so it is possible for one to succeed and the other to fail. For this reason, success is reported separately for clinical and pathologic staging.

### CalculateStage

[ErrorCode](#_ErrorCode) CalculateStage(ref [Datacard](#_Datacard) dc, ref [ErrorCode](#_ErrorCode) clinicalError, ref [ErrorCode](#_ErrorCode) pathologicError)

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Dc | Datacard | Input datacard |
| clinicalError | ErrorCode (ref) | Receives the error code for clinical staging |
| pathologicError | ErrorCode (ref) | Receives the error code for pathologic staging |

The function takes an input datacard and derives TNM clinical and pathologic stage. After derivation, the stage group values are stored in the datacard.

The function returns an ErrorCode indicating the general status of staging. An ErrorCode of OK means a schema was selected and the schema is stageable. This indicates that stage derivation was attempted, but it does not necessarily mean clinical and pathologic staging were successful – a pair of pass-by-reference parameters, *clinicalError* and *pathologicError*, receive the ErrorCodes from clinical and pathologic staging respectively.

**Function return values**

The function returns an ErrorCode other than OK, indicating general failure, under the following conditions:

* If an invalid site code was supplied, the function returns SITE\_NOT\_VALID.
* If an invalid histology code was supplied, the function returns HISTOLOGY\_NOT\_VALID.
* If staging is not applicable for the supplied site, histology, and discriminator values, the function returns SCHEMA\_NOT\_APPLICABLE. Both derived clinical and pathologic stage group are defaulted to “88”.
* If a schema can be identified, but the schema does not have a derived stage table, the schema is not staged (Exception: Lymphoma, which takes derived stage group values from the directly coded stage group tables). The function returns STAGE\_GROUP\_NOT\_DEFINED. Both derived clinical and pathologic stage group are defaulted to “88”.
* If a schema cannot be identified because a valid schema discriminator is required but was not supplied, the function returns SCHEMA\_DISCRIMINATOR\_REQUIRED.

If the function returns any of these ErrorConditions, clinical and pathologic staging are not attempted.

**Clinical and pathologic staging**

The library’s performs the following steps for clinical/pathologic staging:

* validate the input values
* select a stage table
* look up the input values in the stage table
* assign the derived stage group value to the appropriate output field.

**Clinical and pathologic staging errors**

The library sets *ClinicalError* or *PathologicError* to an ErrorCode other than ErrorCode.OK under the following conditions:

* If a stage group value cannot be derived because the input values have been disallowed by the library’s business rules, the associated parameter is set to **STAGE\_GROUP\_CANNOT\_BE\_CALCULATED**.
* For pathologic staging with TNM Path Descriptor = 4 or 6, staging is not performed. *PathologicError* is set to **STAGE\_GROUP\_YP\_NOT\_REPORTABLE**.
* If a required input table for T, N, or M is missing, the associated parameter is set to the ErrorCode for the invalid table. For example, if the schema is missing a Clinical T table, the ErrorCode is CLINICAL\_T\_TABLE\_NOT\_FOUND.
	+ This is a system error and should not occur in practice.
* If one or more input values for clinical or pathologic T, N, or M are invalid, the associated parameter is set to the ErrorCode for the first invalid value. For example, if Clinical T is the first invalid parameter, the error code is INVALID\_CLINICAL\_T.
* If a derived stage table could not be selected, the associated parameter will be set to **DERIVED\_STAGE\_TABLE\_NOT\_FOUND**.
	+ Inability to select a stage table can be caused by an invalid selection of input values. If this code is returned, examine the input values and reselect them if needed.
* If the derived stage group value is “ERROR”, the combination of input values is present in the stage table but not permitted under staging rules, and the relevant parameter is set to **STAGE\_GROUP\_IS\_ERROR**.
* If the combination of input values could not be found in the stage table, the associated parameter is set to **STAGE\_GROUP\_NOT\_FOUND**.
* If a valid value for a required data element for staging (other than T/N/M) was not supplied, the associated parameter is set to **STAGING\_ELEMENT\_MISSING**.

The ErrorConditions are described in the section on ErrorConditions, including a recommendation on how the software vendor should treat each one. Some are not errors in the strict sense, but provide information on why a stage group value was not derived.