Table of Contents

[Overview 3](#_Toc447570777)

[Walkthrough: The Abstraction Process 6](#_Toc447570778)

[Initializing the TNM staging library 6](#_Toc447570779)

[Selecting a schema 6](#_Toc447570780)

[Obtaining a schema discriminator 7](#_Toc447570781)

[Building a table picklist 8](#_Toc447570782)

[Abstracting required data items 9](#_Toc447570783)

[Calculating derived stage group 11](#_Toc447570784)

[Enumerated Types 12](#_Toc447570785)

[DataElement 12](#_Toc447570786)

[TNMTableRole 12](#_Toc447570787)

[ErrorCode 13](#_Toc447570788)

[Explanation of codes 14](#_Toc447570789)

[Notable Classes 18](#_Toc447570790)

[Datacard 18](#_Toc447570791)

[TNMTable 23](#_Toc447570792)

[TNMSchema 27](#_Toc447570793)

[TNM\_Staging 29](#_Toc447570794)

[API reference 30](#_Toc447570795)

[Data transfer 30](#_Toc447570796)

[PopulateDatacardFromBuffer 30](#_Toc447570797)

[PopulateBufferFromDatacard 30](#_Toc447570798)

[Formatting Error Messages 31](#_Toc447570799)

[FormatErrorCode 31](#_Toc447570800)

[Formatting Storage and Display codes 31](#_Toc447570801)

[GetStorageCode 32](#_Toc447570802)

[GetDisplayCode 33](#_Toc447570803)

[Schema selection and access 33](#_Toc447570804)

[GetSchemaNumber 34](#_Toc447570805)

[GetSchemaName 35](#_Toc447570806)

[GetNumberOfSchemas 36](#_Toc447570807)

[GetSchema 36](#_Toc447570808)

[Table access 37](#_Toc447570809)

[GetNumberOfTables 37](#_Toc447570810)

[GetTable 37](#_Toc447570811)

[GetTableByDataElement 38](#_Toc447570812)

[GetTableByRole 38](#_Toc447570813)

[Coding required data elements 39](#_Toc447570814)

[GetRequiredDataElements 40](#_Toc447570815)

[GetRequiredSSFDataElements 40](#_Toc447570816)

[GetRequiredNonSSFDataElements 40](#_Toc447570817)

[Stage calculation 41](#_Toc447570818)

# Overview

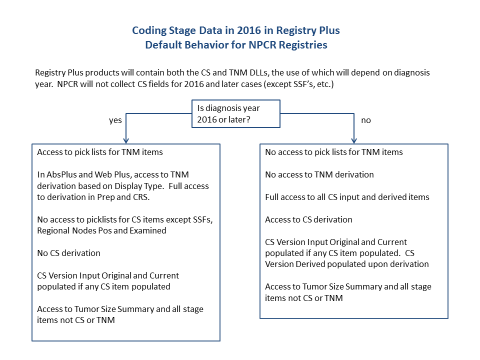
The TNM staging library implements derivation of TNM stage through the use of NAACCR data elements in conjunction with domain-specific business rules. TNM staging is valid for NAACCR v16 cases diagnosed in 2016 and later.

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 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.

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:
    - Find 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.
* Build picklists from the schema’s clinical and pathologic T, N, M, and directly coded stage tables.
* Call the staging library to determine if there are additional data items required to derive stage for the schema.
* For each required data item, construct a picklist and display it
* The abstractor selects a value for each required data item
* Populate a Datacard data structure with input values
* Call the library’s stage calculation function
* Store calculation results in file or database table

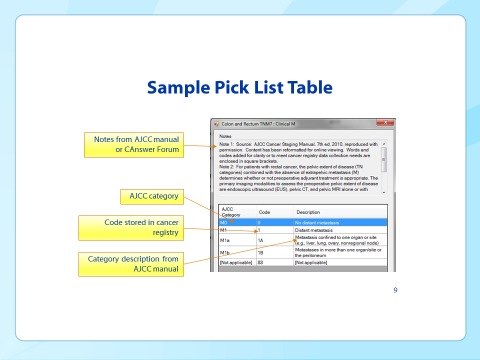


Figure : Example picklist table

# 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 is more than one schema that can be selected.

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 TNM data items. In the third case, an additional element is required to select the schema: a **schema discriminator**.

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

{

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

}

else 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

}

## Obtaining a schema discriminator

The next step is to create a discriminator picklist and display it to the user. This step is only necessary when a schema discriminator is required.

To obtain a schema discriminator value, one must retrieve the schema discriminator table, then select a value from it. In a group of schemas that share a discriminator, each schema will have a copy of the discriminator table, therefore, any of those schemas can be used to retrieve the discriminator table. When GetSchemaNumber() returns a code of SCHEMA\_DISCRIMINATOR\_REQUIRED, it will choose one of the schemas as a default and set *schemaNumber* to its number. It will also set *requiredDataElement* to the identity of the discriminator (even if the Datacard already contains a valid discriminator value).

## Building a table picklist

The function GetTableByDataElement() retrieves a schema table, given a schema number and a DataElement (a data item) associated with the table. For example, passing 0 and DataElement.CLINICAL\_T will retrieve the Clinical T table for the first schema. This function is intended for retrieving input tables only.

The function returns the table via a pass-by-reference parameter. If a table is not found, the parameter is set to null. The function returns an ErrorCode indicating if a matching table was found. (OK if the table was found, TABLE\_NOT\_FOUND otherwise).

A schema only contain tables that are specific to that schema, so it is not uncommon for GetTableByDataElement() to fail to find a table for a particular data item. Site-specific factors are schema-specific, while fields like Age, Behavior, and Grade are not, and thus, do not have schema tables. If a data item does not have a defined table, the picklist must be constructed from the data item’s definition in the NAACCR manual.

For the purpose of this walkthrough, we will assume that a table was returned. The API user can iterate over the table’s notes, column titles, and individual cells to construct a picklist form.

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 remaining TNM data items.

Clinical and Pathologic T, N, and M are always required, so we can retrieve the tables using GetTableByDataElement(). Directly coded Clinical and Pathologic stage group are also required.

Schemas may have additional required data items for staging TNM. The required data items vary from schema to schema. Note: this applies to TNM staging only, not to other data collection requirements.

The API functions GetRequiredDataElements(), GetRequiredSSFDataElements(), and GetRequiredNonSSFDataElements() return the required data elements for a schema, and the required data elements split by SSF and non-SSF respectively.

// Create picklists for clinical and pathologic T/N/M

// Display picklists

// Populate datacard with values

// For this example, we will set some values manually

dc.PutDatacardValue(DataElement.CLINICAL\_T, "c1");

dc.PutDatacardValue(DataElement.CLINICAL\_N, "c0");

dc.PutDatacardValue(DataElement.CLINICAL\_M, "c0");

dc.PutDatacardValue(DataElement.PATHOLOGIC\_T, "p2");

dc.PutDatacardValue(DataElement.PATHOLOGIC\_N, "p1");

dc.PutDatacardValue(DataElement.PATHOLOGIC\_M, "p1");

dc.PutDatacardValue(DataElement.CLINICAL\_STAGE\_GRP, "1");

dc.PutDatacardValue(DataElement.PATHOLOGIC\_STAGE\_GRP, "4");

// Get required SSFs

List<DataElement> requiredSSFs = stageObject.GetRequiredSSFDataElements(schemaNumber);

// Get other required data items

List<DataElement> requiredNonSSFs = stageObject.GetRequiredNonSSFDataElements(schemaNumber);

// Iterate over the required SSFs

foreach (DataElement ssf in requiredSSFs)

{

TNMTable table = null;

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

if (errorCode == ErrorCode.OK)

{

// Build a picklist for the SSF

// Select a value from the picklist

}

}

// Iterate over the required non-SSFs

//

## 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 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);

}

}

# Enumerated Types

## DataElement

**DataElement** represents a data element used in the calculation of TNM stage. DataElements are returned by functions to indicate a specific data element is needed (for example, to supply data to the datacard, or retrieve an associated table).

A DataElement is usually a NAACCR data item, but also includes intermediate variables for stage calculation defined only within the scope of the TNM staging project.

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.

## 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

};

## 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. | 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. |
| **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 Clinical and Pathologic Stage Group fields with default for Not Applicable (generally 88). |
| 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 Clinical and Pathologic Stage Group fields 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 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.

| Member variable | NAACCR Field Name | Item # | Columns | Field Length |
| --- | --- | --- | --- | --- |
| 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 |
| 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.

## 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 indicatd 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:

**RequiredDataElements**

List<DataElement> RequiredDataElements()

The function returns the list of DataElements 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.

**RequiredSSFDataElements**

List<DataElement> RequiredSSFDataElements()

The function returns the list of site-specific factor DataElements used to stage the schema.

If a site-specific factor required for stage derivation is also a schema discriminator, it is excluded from the list.

**RequiredNonSSFDataElements**

List<DataElement> RequiredNonSSFDataElements()

The function returns the list of non-site-specific factor DataElements used to stage the schema.

If primary site, histologic type, or a schema discriminator is also required for stage derivation, it is excluded from the list.

## 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.

## 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 PopulateDatacardFromBuffer(ref 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 PopulateBufferFromDatacard(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 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 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 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 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 GetSchemaNumber(ref Datacard dc, ref int schemaNumber, ref 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 |
| 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.
* **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.

To retrieve a picklist table, the API user should call the function GetTableByDataElement(), passing in *requiredElement*. If a table is successfully retrieved, it can be used as a picklist. If a table is not successfully retrieved, then the API user must construct a picklist using the data item’s definition.

**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.)

### GetSchemaName

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 GetSchema(int schemaNumber, ref 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.

## Table access

This section contains functions for retrieving schema tables.

### GetNumberOfTables

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 GetTable(int schemaNumber, int tableNumber, ref TNMTable table)

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| schemaNumber | Int | Input schema number |
| tableNumber | Int | Input table number |
| 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 GetTableByDataElement(int schemaNumber, DataElement dataElement, ref TNMTable table)

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| schemaNumber | Int | Input schema number |
| 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.

Tables that are used to provide input values for staging can be retrieved by the DataElement that is the input variable.

### GetTableByRole

ErrorCode GetTableByRole(int schemaNumber, TNMTableRole tableRole, ref TNMTable table)

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| schemaNumber | Int | Input schema number |
| 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 every table has a role, but not every table has an associated data element. The typical API user does not need to use this function, but it has been included for completeness sake.

## Coding required data elements

Some schemas require data elements other than Clinical and Pathologic T, N, and M to derive stage. The API has functions to retrieve these data elements so their values can be entered and passed to the stage calculation function.

The list of required data elements excludes schema discriminators, as the API user will have already selected a schema by the time it is necessary to identify the data elements required for staging. For the same reason, the list of required data elements excludes primary site and histologic type, because they have already been selected.

The API does not indicate which data elements are required for collection, as this depends on the standard setter and is 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. The function GetTableByDataElement() is useful for retrieving tables to make picklists.

### GetRequiredDataElements

List<DataElement> GetRequiredDataElements(int schemaNumber)

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| schemaNumber | Int | Input schema number |

The function takes a schema number and returns the list of required DataElements for the associated schema. If the schema number is invalid, or the schema has no required data elements, the function returns an empty List.

In addition, a pair of functions have been provided to return the required site-specific factor (SSF) data elements and non-SSF data elements separately. The reason for separate functions is that SSFs for the TNM staging library have defined schema tables, while non-SSFs do not and must be coded from other sources. The API user may find it useful to treat SSFs and non-SSFs separately.

### GetRequiredSSFDataElements

List<DataElement> GetRequiredSSFDataElements(int schemaNumber)

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| schemaNumber | Int | Input schema number |

The function takes a schema number and returns the list of required site-specific factor DataElements for the associated schema. If the schema number is invalid, or the schema has no required data elements, the function returns an empty List.

### GetRequiredNonSSFDataElements

List<DataElement> GetRequiredNonSSFDataElements(int schemaNumber)

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| schemaNumber | Int | Input schema number |

The function takes a schema number and returns the list of required non-site-specific factor DataElements for the associated schema. If the schema number is invalid, or the schema has no required data elements, the function returns an empty List.

## 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 CalculateStage(ref Datacard dc, ref ErrorCode clinicalError, ref 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 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 and LymphomaOcularAdnexa, which take 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.