ACS NTR Validator

for the National Trauma Registry

SDK Programmer's Guide

Release 1.2

Digital Innovation, Inc.

Table of Contents

Overview

The American College of Surgeons Committee on Trauma (ACS-COT) has led the development of the national trauma registry concept in support of the need for trauma registry data collection at a National level. Recently, federal agencies have made investments to fortify the establishment of a national trauma registry. The result has been changes to the National Trauma Data Bank™(NTDB) to ensure uniformity and data quality and transform it into a data set that is compatible with the National Trauma Registry (NTR) dataset. The ACS-COT Subcommittee also characterized a core set of trauma registry inclusion criteria that would maximize participation by all state, regional and local trauma registries.

Institutionalizing the basic standards provided in the NTR dataset will greatly increase the likelihood that a national trauma registry will provide clinical information beneficial in characterizing traumatic injury and enhancing the ability to improve trauma care in the United States.

In support of the NTR the ACS-COT made provision for the development and support of an NTR Software Development Kit (SDK). The NTR SDK is available for use by any and all trauma registry system vendors and developers to support participation in NTR/NTDB by their systems users, and is offered by ACS at no charge.

The SDK will help support the rapid integration of NTR into NTDB by providing:

- NTR XSD-specific validity checks to ensure proper syntax and structure
- A toolset that is freely available to ALL trauma registry vendors and developers to implement a comprehensive commercial-grade NTR XML Schema validation process into their Windows, Web, and Batch oriented trauma applications.

The SDK has been architected to serve as a robust “chassis” that allows existing and future schema and data validation checks defined by the ACS COT to be “rolled out” to all trauma registry applications on a nationwide basis for NTR and NTDB. Notably, this can be done on an on-going basis as new edits are developed without requiring end users or their vendors to make any programming modifications to their system. That is, the edits would simply “snap in” to any existing applications that utilize the SDK.

The NTR SDK is created, maintained and supported by Digital Innovation, Inc. (DI), the technical and operational partner for NTDB for ACS. DI and American College of Surgeons collaborate to provide clinical and technological advances for NTDB and other trauma initiatives, as part of the strategic business relationship that has been established between the two organizations. Technical support for the NTR SDK is being provided by DI's comprehensive Technical Assistance Center. Technical training webinars and phone

and email-based support services for trauma registry developers in the use of the SDK, such as engineer-to-engineer support of the toolset, are also part of DI's service. These are available at no charge to trauma vendors.

Vision

It is the intention of ACS and DI to continue to enhance the NTR SDK over time. The initial focus of the SDK is on supporting the validation of data files as part of an NTDB/NTR data submission process. In addition, to functionality enhancements, it is expected that the validation logic will also be expanded during the course of this year, and on an on-going basis.

As will be seen in the next section, there is already a stated intent to include support for converting NEMSIS files to NTR ones. NEMSIS compatibility and interoperability is a key design principle of the NTR SDK. Additionally, while the current focus is to validate files at the record level, future versions of the SDK will also support field-level validation. This will enable developers to not only implement data file validation, but to also use the SDK as a parser and to help implement various application logic and functionality above-and-beyond NTDB validation and submission.

Such other possible uses for the SDK have been proposed, and these will be discussed with ACS for possible future software updates. The end objective of the NTR SDK is to assist developers in the implementation of NTR and to provide tools that help result in higher quality NTDB data.

SDK Overview

The NTR SDK primarily centers on the NTRSDK.DLL. The functions described in the next section provide the detail of how the SDK works. This section explains at a higher level what the DLL is, how it works, and where you would use it.

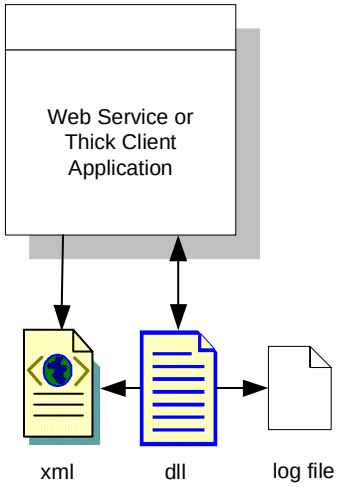

Fig 1, the context for the NTR SDK DLL

Flexible Application Architecture Options - The major functions of the DLL are exposed through the NTRSDK.DLL. Any Windows-based application, whether it's a .NET app, web service or some other form of thick client, can incorporate the DLL into its distribution. The DLL functions are called from the main application and need only provide information about the location of the XML file to be validated along with the name and location for an optional log file of parsed results.

Another major benefit of this approach is that validation logic updates can be incorporated into third party applications without recompiles or programming changes. It is only the DLL (and its associated configuration files provided with the SDK) that will essentially change as the standard for the NTR XSD evolves and matures. This provides a form of "snap-in" approach to supporting the latest standards for the NTR XSD. Furthermore, a special function call can inform the developer of the exact version of the SDK that is being used.

NEMSIS Compatibility - In a bigger context, the NTR SDK incorporates functionality to translate NEMSIS XML files to standard NTR XML files to provide a full range of functions from Hospital EMS systems to the National Trauma Data Bank, as depicted below. The NTR standard contains a number of variables that can be used to link it from NEMSIS (such as name, address, EMS Run number, Date of Service) as well as approximately 20 variables (36%) that can be auto-populated based on the information available in the NEMSIS data set.

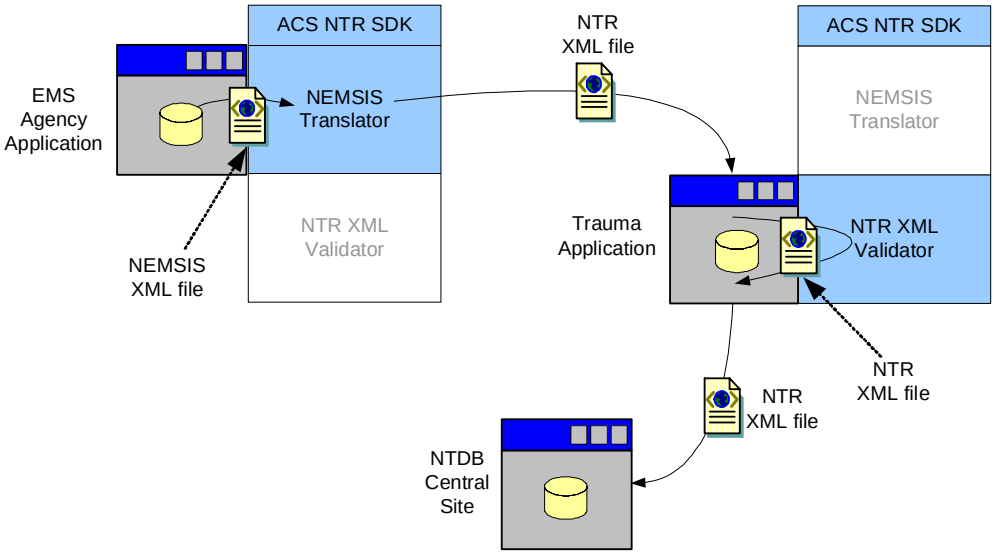

Fig 2, use of the NTR SDK in the NEMSIS / NTR context

Here it can be seen that the NTR SDK is used to convert a NEMSIS XML file from an EMS Agency into an NTR XML file that is transferred to a Trauma application. The NTR SDK is then used to validate the received NTR XML file before it is sent onto the National Trauma Data Bank (NTDB). All of these functions exist in the same SDK.

SDK Usage - At a more detailed level the diagram below demonstrates where the DLL fits into the context of the overall application:

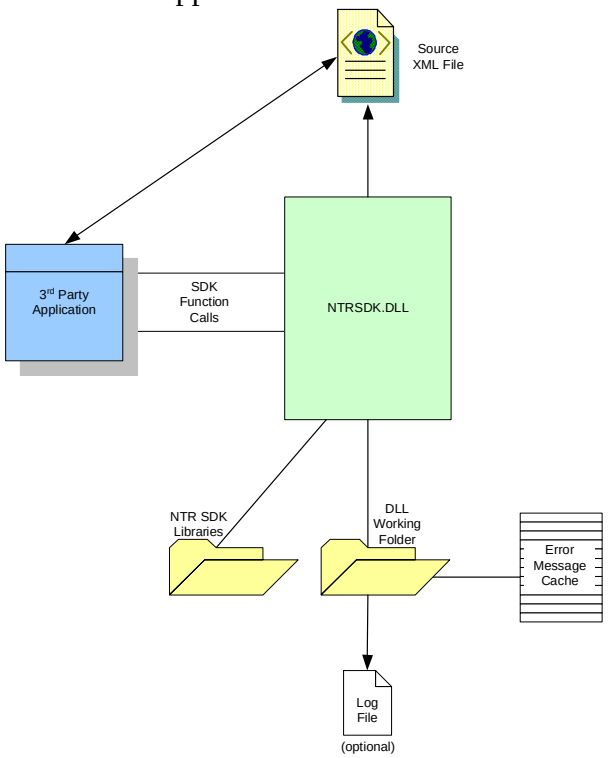

Fig 3, integrating the NTR SDK DLL

The calling application tells the DLL where to locate the:

- Source XML file to be parsed
- Libraries that accompany the NTRSDK.DLL
- Working directory for temporary files created during the parsing
- Optional log file that contains the results of the XML file parsing

Unless specifically requested, the results of the file parsing are contained in a special cache that can be read from the calling application. This gives the developer the opportunity to correct any schema or logic problems before the parsing is completed.

The algorithmic flow in the DLL functions gives the developer the opportunity to parse the whole file in one stroke or to process the file one record at a time and investigate any errors that occur along the way. This allows for multiple application approaches – i.e., one approach of scanning the resulting log file, or the latter approach of traversing the NTR XML file a record at a time and handling any associated messages as desired. A sample program for the latter approach is included in Appendix 2.

Finally, it should be noted that the NTR SDK can be distributed royalty-free with any US-copyrighted application whose license agreement prohibits any form of reverse engineering.

Function Reference

What's New in Version 1.2

This latest release of the NTR SDK has focused on fulfilling the following objectives:

- Making the API functions complete so that coding changes are not required in future release of the SDK. Therefore the lists of API functions and their respective parameters should not change.
- Including support for the Error Check scheme published in the final version of the NTR Data Dictionary as well as supporting the final XSD changes.
- Providing a greater degree of control over XML file validation by introducing a new function that can validate at an NTR field level as well as the previously provided record-level functions. Notably this new function is called *ValidateNtrFieldValue*.
- Provide more information in the error message buffers as well as in the output log file to assist the developer in correlating the validation results with the source record data. New functions included are: *GetNtrFieldValue* and *GetNtrMessageEntryTagName*. Functions *StartValidatingNtrFile*, *ValidateNextNtrRecord* and *GetNtrMessageEntry* have new options included to provide more information than previously.
- The format of the log file output by the validator has changed – including substantially more information than previously. See Appendix 1 for details.

This release will not contain a complete list of the error checks published in the NTR Data Dictionary but will progressively introduce these checks over subsequent releases. Be sure to review the Release Notes that accompany each SDK release for reference to the list of included error checks.

Data Types

Standard Data Types	
STATUS	This is a 32-bit integer that is used to indicate the status of a function. If the returned value is negative the operation failed; otherwise the operation succeeded.
LOGICAL	This is a 32-bit integer that is used to indicate whether a condition is true or false. A value of 0 indicates false, any other value indicates true.
USHORT	This is a 16-bit unsigned integer (values may not be negative).
UINT	This is a 32-bit unsigned integer (values may not be negative).

ULONG	This is a 32-bit unsigned integer (values may not be negative).
LPCSTR	This is a pointer to a null (0) terminated string (C-style string). In C this would be a const char*. The characters are standard 8-bit ASCII characters.
& (Reference):	This modifies one of the above types; indicating that the associated parameter is passed 'by reference' (as opposed to 'by value').

Function Summary

- NtrSdkVersion
- InitializeNtrSdk
- ShutdownNtrSdk
- StartValidatingNtrFile
- ValidateNextNtrRecord
- ValidateNtrFieldValue
- GetNtrFieldValue
- StopValidatingNtrFile
- NtrFileVersion
- NumNtrMessages
- GetNtrMessageEntry
- GetNtrMessageEntryTagName
- ValidateNtrFile
- TranslateNemsisFileToNtr

DLL Functions

NtrSdkVersion

This function performs the following task(s):

- Displays the version of the NTR SDK DLL.

Prototype: void NtrSdkVersion(
USHORT& majorVersion,
USHORT& minorVersion,
USHORT& buildNumber
)

Parameters:

Name	Type	Description
majorVersion	USHORT	[out] The major version number of the release.
minorVersion	USHORT	[out] The minor version number of the release.
buildNumber	USHORT	[out] The build number of the release.

Return value: None.

Remarks:

- Used to retrieve the version information of the NTR SDK DLL that is being called to validate the source XML files.
- This function can be used in any context and does not require *InitializeNtrSdk* or *ShutdownNtrSdk* as prerequisites.

InitializeNtrSdk

This function performs the following task(s):

- Prepares the DLL for use.

Prototype:

```
STATUS  InitializeNtrSdk(
                LPCSTR ntrSdkPath,
                LPCSTR workingPath
                )
```

Parameters:

Name	Type	Description
ntrSdkPath	LPCSTR	[in] Pointer to a null-terminated string that specifies the full path name for the location of the NTR SDK.
workingPath	LPCSTR	[in] Pointer to a null-terminated string that specifies the full path name for the location of a temporary directory where temporary files are created.

Return value:

Return value is of type: STATUS.

Value	Meaning
0	Function succeeded.
-1	Function failed. General error.
-2	Function failed. The SDK files are not found or invalid.

Remarks:

- NTR SDK consists of the NTRSDK.DLL file along with the necessary libraries and configuration files that the DLL uses.
- This function must be called successfully prior to calling any other function. If this function is successfully called then *ShutdownNtrSdk* must be called once the DLL is finished being used.
- After a successful call, the *ValidateNtrFile* function may be called.

ShutdownNtrSdk

This function performs the following task(s):

- Clean up internal data.

Prototype: void ShutdownNtrSdk()

Parameters: None.

Return value: None.

Remarks:

- This function is to only be called after a successful call to *InitializeNtrSdk*. No functions other then *InitializeNtrSdk* are to be called once *ShutdownNtrSdk* has been called.

StartValidatingNtrFile

This function performs the following task(s):

- Opens the source XML file to prepare the DLL for record-by-record processing of the file. If you simply want to validate an entire XML file as a single batch process, you may call *ValidateNtrFile*.

Prototype:

```
STATUS StartValidatingNtrFile(
                    LPCSTR xmlFilename,
                    USHORT validationLevel
                )
```

Parameters:

Name	Type	Description
xmlFilename	LPCSTR	[in] The name of the source XML file to parse.
validationLevel	USHORT	[in] Valid entries are: • 0 = No validation • 1 = Schema errors only • 2 = Schema + Critical errors • 3 = Schema + Critical + Logic errors.

Return value:

Return value is of type: STATUS.

Value	Meaning
0	Ok.
-n	Failed.

Remarks:

- Can be called after successful *InitializeNtrSdk*. Should not be called again after a successful call until the corresponding *StopValidatingNtrFile* function is called.
- A successful call to *StartValidatingNtrFile* is required before *ValidateNextRecord* can be used.

ValidateNextNtrRecord

This function performs the following task(s):

- Parses a single record in the XML file that has been opened by a *StartValidatingNtrFile* function.
- Places any error messages matching the selected validationLevel into the error message buffer for later interrogation.
- Positions itself for validating the next NTR record in sequence.

Prototype:

```
STATUS  ValidateNextNtrRecord(
                    LOGICAL ignoreRequiredChecks
                    )
```

Parameters:

Name	Type	Description
ignoreRequiredChecks	LOGICAL	[in] When set to FALSE the error checks are normal. When set to TRUE, required checks in each of the Schema, Critical and Logic error types are suppressed.

Return value:

Return value is of type: STATUS.

Value	Meaning
0	Record successfully processed.
-1	Record not successfully processed. (General error)
-2	Record not successfully processed. End of File reached.

Remarks:

- A return STATUS of 0 or -2 can be construed as normal behavior. However, as with all STATUS functions, any negative return value indicates that the operation (in this case 'validating the next record') did not succeeed.
- If EOF is reached the next function that should be used is *StopValidatingNtrFile*. (i.e., specifically, you should NOT try to retrieve any messages or other details about the current record processing – since no such processing occurred.)
- Any other STATUS indicates the parsing has encountered a problem and processing of the XML file has ceased. Once an error occurs, further processing will not be possible, and the *StopValidatingNtrFile* function should be called.

ValidateNtrFieldValue

This function performs the following task(s):

- Parses a single XML tag in the current record pointed at by the ValidateNextNtrRecord.
- Places any error messages matching the selected validationLevel into the error message buffer for later interrogation.
- Remains at the current NTR record.

Prototype: STATUS ValidateNtrFieldValue(
LPCSTR xmlTagname
)

Parameters:

Name	Type	Description
xmlTagname	LPCSTR	[in] The name of the XML tag to parse.

Return value:

Return value is of type: STATUS.

Value	Meaning
n	Any positive number denotes the number of error messages generated.
0	Tag successfully processed.
-1	Tag not successfully processed. (General error)

Remarks:

- Only outer level tags are allowed. If the tag is a container then the function validates all fields in all items of the container.
- Returns # of messages if successful.
- If 0 is returned then the function completed successfully without any errors.
- Any STATUS less than 0 indicates the parsing has encountered a problem and processing of the XML file has ceased. Once an error occurs, further processing will not be possible, and the *StopValidatingNtrFile* function should be called.
- This function inherits the validationLevel commenced with the *StartValidatingNtrFile* function as modified by ignoreRequiredChecks in *ValidateNextNtrRecord.* These settings cannot be overridden.
- Note: this function only performs schema checks.

GetNtrFieldValue

This function performs the following task(s):

- Retrieves the field contents of a nominated tag.
- Functions on the current record selected by *ValidateNextNtrRecord*.

Prototype: STATUS ValidateNtrTag(
LPCSTR tagName,
LPCSTR messageBuffer,
USHORT messageBufferSize
)

Parameters:

Name	Type	Description
tagName	LPCSTR	[in] The name of the XML tag to retrieve the contents.
messageBuffer	LPCSTR	[in] The contents of the Buffer to store the Field Value.
messageBufferSize	USHORT	[in] This should specify the size of the messageBuffer (in bytes) that is passed to the function.

Return value:

Return value is of type: STATUS.

Value	Meaning
0	Tag successfully processed.
-1	Tag not successfully processed. (General error)

Remarks:

- Only outer level, non-container tags are allowed.
- If 0 is returned then the function completed successfully without any errors.
- Any STATUS less than 0 indicates the parsing has encountered a problem and processing of the XML file has ceased. Once an error occurs, further processing will not be possible, and the *StopValidatingNtrFile* function should be called.
- A possible use of this function would be to retrieve the contents of the FacilityID and PatientID fields for output to a log file or correlation with source data.

StopValidatingNtrFile

This function performs the following task(s):

- Closes the source XML file and cleans up any internal message buffers.

Prototype: void StopValidatingNtrFile(
)

Parameters: None.

Return value: None.

Remarks:

- Should not be used unless *StartValidatingNtrFile* has already been successfully called.
- Must be called before *ShutdownNtrSdk*.

NtrFileVersion

This function performs the following task(s):

- Retrieves the version of the NTR file stored in the tag <NTRVersion> (i.e. <xs:attribute name="NTRVersion" use="required" fixed="v2.0.0"/>).

Prototype: STATUS NtrFileVersion (
LPCSTR versionBuffer,
USHORT versionBufferSize
)

Parameters:

Name	Type	Description
versionBuffer	LPCSTR	[in] The name of a Buffer to hold the version information from the XML file.
versionBufferSize	USHORT	[in] The size of Buffer to be used above.

Return value:

Return value is of type: STATUS.

Value	Meaning
0	Retrieval of NTRVersion was successful.
-n	Retrieval failed.

Remarks:

- Should not be used unless *StartValidatingNtrFile* or *ValidateNtrFile* has already been successfully called.
- Returns FAILED if no version was specified OR there was never a call to *StartValidatingNtrFile* or *ValidateNtrFile*.

NumNtrMessages

This function performs the following task(s):

- Retrieves the number of error messages of a specified type that have been buffered during an XML file parse of the current record.

Prototype: ULONG NumNtrMessages(
USHORT messageType
)

Parameters:

Name	Type	Description
messageType	USHORT	[in] Valid enumerated input value(s): • 0 = all (UPPER all of these) • 1 = schema • 2 = critical • 3 = warning

Return value:

Return value is of type: ULONG and is the number of messages of the specified messageType.

Remarks:

- If the value returned is 0 then there are no error messages of the type specified.
- Note: checking for error messages of a type that have not been captured with the options parameter in *StartValidationNtrFile*,ill always return 0.
- The number of error messages retrieved would normally be used in conjunction with the *GetNtrMessageEntry* function.
- It is ok to call this function in between calls of *ValidateNtrFieldValue*, so that messages specific to the last call to this function can be retrieved.

GetNtrMessageEntry

This function performs the following task(s):

- Retrieves the buffered error message information related to the XML record that has just been parsed.

Prototype: STATUS GetNtrMessageEntry (

ULONG	whichMessage,
UINT&	messageType,
UINT&	messageCode,
UNIT&	messageId,
LPCSTR	messageBuffer,
USHORT	messageBufferSize
)	

Parameters:

Name	Type	Description
whichMessage	ULONG	[in] The number of the message that should be retrieved. Will be limited to minimum of 1 and the maximum value returned from *NumMsg*.
messageType	UINT&	[out] Valid enumerated input value(s): • 0 = all • 1 = schema • 2 = critical • 3 = warning
messageCode	UINT&	[out] Should be specified as 0. Other values are reserved for future usage.
messageId	UNIT&	[out] Contains the Rule ID integer / error code associated with the message. (See Appendix 1 or NTR Data Dictionary for further detail).
messageBuffer	LPCSTR	[in] The Buffer to store the message. It is recommended that this be at least 128 bytes in size.
messageBufferSize	USHORT	[in] This should specify the size of the messageBuffer (in bytes) that is passed to the function.

Return value:

Return value is of type: STATUS.

Value	Meaning
0	Function succeeded.
- n	Function failed.

Remarks:

- Typical uses of this function could include writing the contents of the error messages buffers to a log file or used to correct identified problems, depending on programmer choice.
- Note: if there are 0 messages for the current record, then *GetNtrMessageEntry* should not be called. The function will prevent the program from specifying a value for messageNum that is higher than the number of messages obtained from the *NumNtrMessages* function (using the ALL parameter).

GetNtrMessageEntryTagName

This function performs the following task(s):

- Retrieves the tag name associated with the buffered error message information retrieved by *GetNtrMessageEntry*.

Prototype:

```
STATUS GetNtrMessageEntryTagName (
                ULONG       whichMessage,
                LPCSTR      tagName,
                USHORT      tagBufferSize
                )
```

Parameters:

Name	Type	Description
whichMessage	ULONG	[in] The number of the message that should be retrieved. Will be limited to minimum of 1 and the maximum value returned from *NumMsg*.
tagName	LPCSTR	[in] Contains the name of the XML tag associated with the error message.
tagBufferSize	USHORT	[in] This should specify the size of the tagName buffer (in bytes) that is passed to the function.

Return value:

Return value is of type: STATUS.

Value	Meaning
0	Function succeeded.
- n	Function failed.

Remarks:

- Typical uses of this function could include writing the tag name to a log file or used to correct identified problems, depending on programmer choice.
- Note: if there are 0 messages for the current record, then *GetNtrMessageEntryTagName* should not be called.

ValidateNtrFile

This function performs the following task(s):

- Validates an entire source XML file at one time.

Prototype:

```
LONG STATUS  ValidateNtrFile (
                    LPCSTR      xmlFilename,
                    LPCSTR      logFilename,
                    LPCSTR      options,
                    LOGICAL     eofFlag
                    )
```

Parameters:

Name	Type	Description
xmlFilename	LPCSTR	[in] The name of the source XML file to open and parse.
logFilename	LPCSTR	[in] The name of the log file to open for writing the results of each record parse.
options	LPCSTR	[in] Valid switch values are: • -ok = log successful parse • -error = log an error found Note: these option switches are included for future use and are currently ignored by the SDK.
eofFlag	LOGICAL&	[out] Set to TRUE if the function succeeds and the end of file was reached. If the function succeeds (in processing some records), but the EOF was not reached, then it will be set to FALSE. If the function is not successful, the value of this parameter is not defined and should not be used.

Return value:

Return value is of type: LONG_STATUS and will return the number of records processed (if successful), and a negative value otherwise.

Remarks:

- This is an alternative method for parsing a whole file without having to handle a record at a time.
- The *InitializeNtrSdk* function must precede it and the *ShutdownNtrSdk* function would normally come after it.

TranslateNemsisFileToNtr

This function performs the following task(s):

- Translates the relevant portion of a NEMSIS standard XML file into an NTR standard XML file.

Prototype: void TranslateNemsisFileToNtr(
LPCSTR sourceXmlFilename,
LPCSTR targetXmlFilename
)

Parameters:

Name	Type	Description
sourceXmlFilename	LPCSTR	[in] The name of the source XML (NEMSIS) file to open, parse and translate.
targetXmlFilename	LPCSTR	[in] The name of the target XML (NTR) file to open and write to.

ReturnValue: None.

Remarks:

- This function requires *InitializeNtrSdk* to be called before use. As always, the *ShutdownNtrSdk* function should be called when done using the DLL function.
- This function should NOT be called in the context of a *StartValidatingNtrFile* sequence. Instead, you must first close any such file using *StopValidatingNtrFile.*

Support

The SDK will be supported and maintained by DI. These services will include the following:

1. Providing email and telephone technical support for all trauma registry system vendors and developers in the use and integration of the SDK in their software applications.
2. Maintaining the SDK to include periodic updates, including NTR edit checks specified by the ACS-COT (as well as NTDB edit checks, as NTR becomes integrated into NTDB).
3. Software distribution of the SDK and associated documentation (in electronic formats) to any trauma registry vendor or developer requesting the SDK
4. Maintenance of a SDK Request Form on the NTDB data center web site.
5. Web-based seminars in the usage of the SDK for interested developers.

The best method to engage DI support for the SDK is via email at ntrsdksupport@dicorp.com.

DI maintains a distribution list of developers who are interested in receiving the SDK and participating in various NTR SDK product feedback forums that may be scheduled from time to time. Interested individuals should send an email to the same email address requesting to be added to the list.

As an alternative, support queries can be referred by phone to the software engineers on 800.344.3668 x 248, or to fax 410.893.3199 and reference NTR SDK in the subject heading. Further contact information for Digital Innovation can be found on DI's website at http://www.dicorp.com/Contact_Admin.html.

SDK support is available between the hours of 9 am and 5 pm EST, Monday to Friday on regular business days.

Appendix 1: Error Messages

New to this release of the NTR SDK is support for the error messages as defined in the NTR Data Dictionary – Appendix 3: Edit Checks. The SDK supports the Rule ID and Message text for the Edit Checks but further defines the Level of error according to the previous scheme implemented by the SDK. That is: 1=Schema, 2=Critical, 3=Logic.

Due to the large number of Edit Checks, the SDK will progressively phase in the complete set of Checks over the next few releases.

The format of the error log file is comma-separated with eight columns:

1. Facility ID – the value populated into the XML file in the field FacilityID
2. Patient ID – the value populated into the XML record in the field PatientID
3. Record number (in order that they occur in the XML file)
4. The field/tag name associated with the Error
5. Error type (1=Schema, 2=Critical, 3=Logic)
6. Error Code – a generic description of the error. For example "Required Field". The combination of tag name, error type, and error code correlates to the list of Rule ID error codes contained in the NTR Data Dictionary.
7. The Rule ID of the Error – the code is per the NTR Data Dictionary but is formatted as a 5 digit code instead of 3. Codes starting with 10xxx will be system-oriented codes unique to the SDK and 20xxx will denote the NTR Data Dictionary codes.
8. The Edit Check description from the NTR Data Dictionary.

A list of Error Codes, as defined in column 6 above, will be included in future versions of the SDK Programmer's Guide. The Error Codes and Rule IDs implemented in the next release of the SDK will be included in the Release Notes.

Appendix 2: Sample Code

```
    // Sample C++ code to initialize the SDK and validate an NTR XML file called "file.xml"

#include <ntrsdk.hpp>

STATUS SampleCodeExample()
{
    if (InitializeNtrSdk(".\\NTRSDK", ".\\") < 0)   // The SDK must be initialized before use
        return -1;                                  // Leave if the SDK cannot be initialized

    if (StartValidatingNtrFile("file.xml", 3) < 0) {
            // Start validating the file, generating all messages (i.e., option 3), and check for error
        ShutdownNtrSdk();
            // Be sure to shutdown the SDK before exiting if there is an error
        return -1;
        }

    UINT messageType;                               // These variables are used to store a message
    UINT messageCode;
    const USHORT MESSAGE_BUFFER_SIZE = 128;
    CHAR messageBuffer[MESSAGE_BUFFER_SIZE];

    while (ValidateNextNtrRecord() >= 0) {
                // Loop through, processing one record at a time, until there is an error or EOF

        ULONG numMessages = NumNtrMessages(0);
                // Get the number of messages for the current record.  (Note: the 0 means 'ALL' types)

        for (ULONG whichMessage = 1; whichMessage <= numMessages; ++whichMessage) {
                // Loop through each message  (Note: the loop does nothing and is skipped if numMessages = 0)

            GetNtrMessageEntry(whichMessage, messageType, messageCode, messageBuffer, MESSAGE_BUFFER_SIZE);
                // This sets the above variables with the details about the specified message for the current record

            // ... application code goes here ...
            // This is where you could generate a log, or do some other application-specific process with the messages
            }
        }

    StopValidatingNtrFile();
            // Must be sure to always call this function if there is a successful StartValidatingNtrFile

    ShutdownNtrSdk();
            // Must be sure to always call this function if there is a successful InitializeNtrSdk

    return 0;
}
}
```