ACS NTDS Validator for the NTDB

SDK Programmer's Guide

Release 2017 v1

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 Data Standard (NTDS) 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 NTDS 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 NTDS the ACS-COT made provision for the development and support of an NTDS Software Development Kit (SDK). The NTDS 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 NTDS into NTDB by providing:

- NTDS 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 NTDS 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 NTDS 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 NTDS 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 NTDS 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.

The NTDS SDK was first released in 2006 and used for the submission of data to NTDB for the first time in 2008.

SDK Overview

The NTDS SDK primarily centers on the NTDSSDK.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.

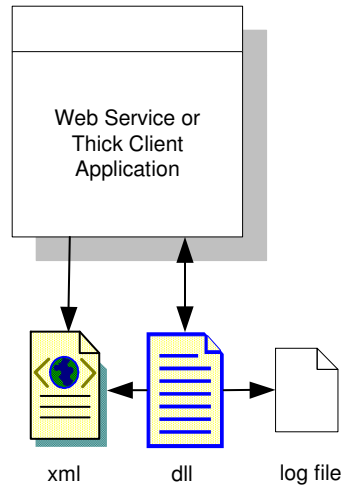

Fig. 1, the context for the NTDS SDK DLL

Flexible Application Architecture Options - The major functions of the DLL are exposed through the NTDSSDK.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 NTDS XSD evolves and matures. This provides a form of "snap-in" approach to supporting the latest standards for the NTDS XSD. Furthermore, a special function call can inform the developer of the exact version of the SDK that is being used.

Multi-Channel – from 2010 onwards the SDK supports two or more sets of validation logic in each deployment. The developer chooses a "channel" that corresponds to a Call For Data (CFD) rule set. Typically any given version of the Validator will support a CFD year as well as the year following so that the rules applied to each year can be different from each other. The Trauma Registrar is therefore permitted to finalize records to be submitted in the CFD year according to one rule set and is also able to

enter new validated records into their Registry for the following year according to a different set of validation rules.

Aggregate Rules – Also added from v2010.5 onwards are new API methods for working with xml files at the aggregate level. The new methods enable a special set of Validation Rules to be added that work at a file or aggregate level within the selected channel. The idea being that we are able to check consistency and coherence of all the records in the xml submission file with each other (e.g. verify that all the Facility IDs are the same across the file).

The inclusion of these new API methods does not change any of the existing methods, so that current use of the Validator is not affected by the inclusion of these new methods. Developers are therefore able to introduce support for Aggregate Validation Rules as the need arises.

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

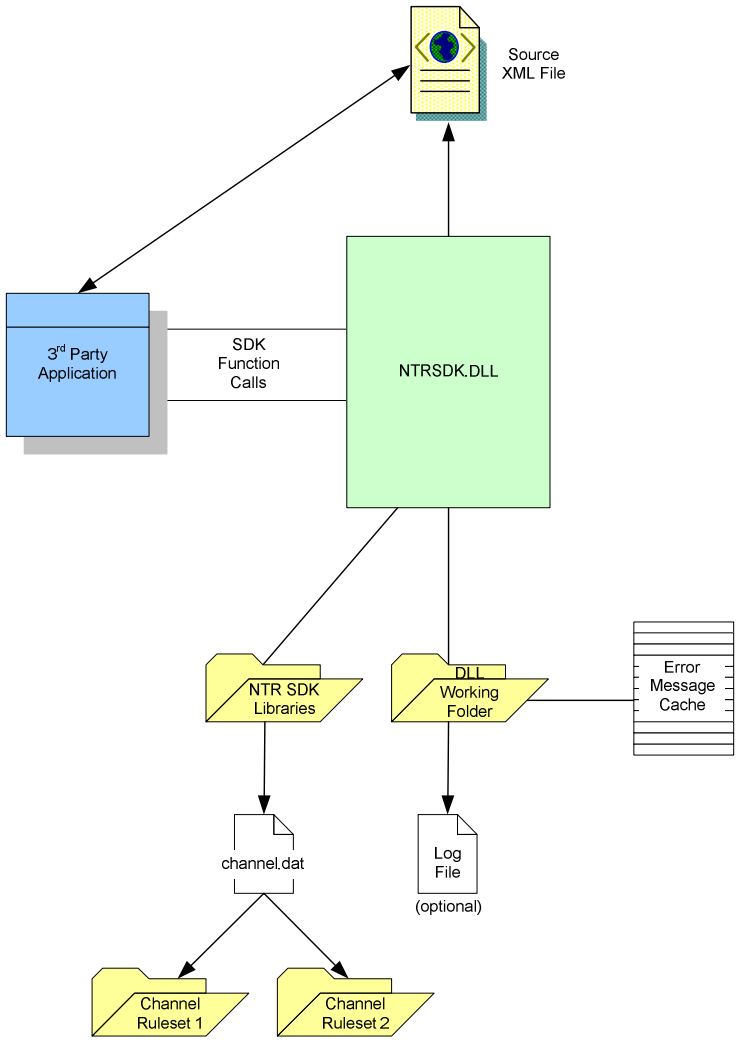

Fig 2, integrating the NTDS SDK DLL

The calling application tells the DLL where to locate the:

- The channel or version of the Validator rules to use
- Source XML file to be parsed
- Libraries that accompany the NTDSSDK.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 NTDS 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 NTDS 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 2017 v 1

(Please consult the release notes included with your zip file package. The NTDB channel file is called *v2017.1 NTDB Release Notes.pdf.* The TQIP channel file is called *v2017.1 TQIP Release Notes.pdf*.)

This release supports the 2017 Data Dictionary for admissions from 2017. Highlights in this release are:

1. This release is a "channel only" release that includes the new channels 109 and 209 for the 2017 Data Dictionary (v2017.1.1). If you need a full installation, then contact Digital Innovation at NtdsSdkSupport@dicorp.com for assistance.
2. Channel 109 is like its predecessors and supports validation of only the core NTDS data elements. The "100" series of channel IDs will continue to be reserved for the core NTDS data element validation Rules.
3. Channel 209 supports all the core NTDS data elements plus the Process Measures/TQIP data elements in the 2017 Data Dictionary. The "200" series of channel IDs will be reserved for Process Measure validation Rules and be a superset of that includes the core NTDS data elements.
4. There is a separate XSD that supports each of the two new channels. Note that AIS coding is now mandatory. It used to be that AIS was the only "optional" set of elements but now all elements in the XSD require a data value (if only a BIU attribute).
5. Introduced late 2015 was a new optional attribute called "icd10". If the attribute is not present in the xml the Validator will assume that the version of ICD-10 used is the US ICD-10-CM. If the attribute is used it can have the value "ICD10CA" which will alter the Validator's behavior to apply ICD-10-CA (Canada) coding. Over time it is expected that other systems of ICD-10 coding will be added using this mechanism.

Note that the CHANNEL.DAT file has been updated in this release to include the new channels.

Channel Lineup

The complement of deployed Validator channels now numbers three as follows:

Channel ID	Description
101	The original prechannel SDK circa 2009)
102	2010 Data Dictionary (used for 2009 and 2010 admission years)
103	2011 Data Dictionary (used for 2011 admission year)
104	2012 Data Dictionary (used for 2012 admission year – NTDS elements only)
204	2012 Data Dictionary (used for 2012 admission year – including Process Measure/TQIP data elements as well as NTDS)
105	2013 Data Dictionary (used for 2013 admission year – NTDS elements only)
205	2013 Data Dictionary (used for 2013 admission year – including Process Measure/TQIP data elements as well as NTDS)
106	2014 Data Dictionary (used for 2014 admission year – NTDS elements only)

206	2014 Data Dictionary (used for 2014 admission year – including Process Measure/TQIP data elements as well as NTDS)
107	2015 Data Dictionary (used for 2015 admission year – NTDS elements only)
207	2015 Data Dictionary (used for 2015 admission year – including Process Measure/TQIP data elements as well as NTDS)
108	2016 Data Dictionary (used for 2016 admission year – NTDS elements only)
208	2016 Data Dictionary (used for 2016 admission year – including Process Measure/TQIP data elements as well as NTDS)
109	2017 Data Dictionary (used for 2017 admission year – NTDS elements only)
209	2017 Data Dictionary (used for 2017 admission year – including Process Measure/TQIP data elements as well as NTDS)

Data Types

The table below explains the data types used in the API.

Standard Data Types	
STATUS	This is a 16-bit signed integer that is used to indicate the status of a function. If the returned value is negative the operation failed; otherwise the operation succeeded.
LONG_STATUS	This is a 32-bit signed integer that is used to indicate the status of a function. If the returned value is negative the operation failed; otherwise the operation succeeded. Might be used to return a record count.
LOGICAL	This is a 16-bit signed 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

- NtdsSdkVersion
- InitializeNtdsSdk
- ShutdownNtdsSdk
- OpenNtdsChannel
- CloseNtdsChannel
- StartValidatingNtdsFile
- ValidateNextNtdsRecord
- ValidateNtdsFieldValue
- GetNtdsFieldValue
- StopValidatingNtdsFile
- StartNtdsAggregateComputing

- StopNtdsAggregateComputing
- ClearNtdsAggregateData
- NtdsFileVersion
- NumNtdsMessages
- GetNtdsMessageEntry
- NumNtdsAggregateMessages
- GetNtdsAggregateMessageEntry
- GetNtdsMessageEntryTagName
- ValidateNtdsFile
- ValdateNtdsFileWithModeOptions

A typical "flow" of API calls to begin validating an xml file is in the following sequence. Each function is important to complete before execution of the one following:

1. *InitializeNtdsSdk* – this initializes the NTDSSDK.DLL for use
2. *OpenNtdsChannel* – this call selects a channel or ruleset that will be applied to the validation as it proceeds.
3. *StartValidatingNtdsFile* – this function opens the xml file ready to begin parsing.
4. *ValidateNextNtdsRecord* – moves forward one xml record into the xml file and applies the ruleset from the channel to produce the error flags that apply (if there are any).

DLL Functions

NtdsSdkVersion

Purpose: This function performs the following task(s):

- Displays the version of the NTDS SDK DLL if a channel has not yet been selected.
- Displays the version of the DLL associated with the channel if a channel has been successfully opened.

Prototype:

```
VOID NtdsSdkVersion(
                    USHORT& majorVersion,
                    USHORT& minorVersion,
                    USHORT& buildNumber
                    )
```

Parameters:

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

Return Value: None

Remarks:

- Used to retrieve the version information of the NTDS SDK DLL that is being initialized to select one of the channels available.
- Used to retrieve the version information of the channel DLL that is being called to validate the source XML files.
- This function can be used in any context and does not require *InitializeNtdsSdk* or *ShutdownNtdsSdk* as prerequisites.

InitializeNtdsSdk

Purpose: This function prepares the DLL for use.

Prototype:
```
STATUS  InitializeNtdsSdk(
                    LPCSTR ntdsSdkPath,
                    LPCSTR workingPath
                    )
```

Parameters:

Name	Type	Description
ntdsSdkPath	LPCSTR	[in] Pointer to a null-terminated string that specifies the full path name for the location of the NTDS 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 are invalid.

Remarks:

- NTDS SDK consists of the NTDSSDK.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 *ShutdownNtdsSdk* must be called once the DLL is finished being used.
- After a successful call, the *OpenNtdsChannel* function may be called.

ShutdownNtdsSdk

Purpose: This function cleans up internal data.

Prototype: VOID ShutdownNtdsSdk()

Parameters: None.

Return Value: None.

Remarks: This function is to only be called after a successful call to *InitializeNtdsSdk*. No functions other then *InitializeNtdsSdk* are to be called once *ShutdownNtdsSdk* has been called

OpenNtdsChannel

Purpose: This functionselects and initializes one of the channel or "rulesets" available in the SDK. Only one channel can be open at any time.

Prototype:
```
STATUS  OpenNtdsChannel(
                        USHORT channel
                        )
```

Parameters:

Name	Type	Description
Channel	USHORT	[in] The number of the channel that has been assigned in the `channel.dat` file.

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 are invalid.
-20	Unknown channel number
-21	Channel already open
-22	General error: Could not open channel.

Remarks:

- Only call this function after successfully calling *InitializeNtdsSdk*.
- This function must be called successfully prior to calling any function upon which a channel selection is required. For example, this function must be completed before *StartValidatingNtdsFile*.
- After a successful call, the *ValidateNtdsFile* function may be called.

CloseNtdsChannel

Purpose: This function closes the channel that is currently open.

Prototype: STATUS CloseNtdsChannel()

Parameters: None.

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 are invalid.
-11	No channel is currently open.
-20	Unknown channel number

Remarks:

- This function is only to be called after a successful call to *OpenNtdsChannel*.
- This function must be called before opening a different channel to use for validation.
- If a channel is currently open then this function must be called before the *ShutdownNtdsSdk* function can be called.

StartValidatingNtdsFile

Purpose: 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 *ValidateNtdsFile*.

Prototype:

```
STATUS  StartValidatingNtdsFile(
                    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 + Inclusion Logic • 3 = Schema + Inclusion + Major Logic • 4 = Schema + Inclusion + Major + Minor Logic

Return Value: Return value is of type STATUS.

Value	Meaning
0	Function succeeded.
-11	No channel is currently open.
-n	Failed.

Remarks:

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

ValidateNextNtdsRecord

Purpose: This function performs the following task(s):

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

Prototype:

```
STATUS  ValidateNextNtdsRecord(
                                    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 processed. General error.
-2	Record not processed. End of file reached.
-11	No channel has been selected.

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 succeed.
- If EOF is reached the next function that should be used is *StopValidatingNtdsFile*. (i.e., specifically, you should NOT try to retrieve any messages or other details about the current record processing – since no such processing occurred.)
- A -11 status indicates that this function has been used before a channel was selected.
- 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 *StopValidatingNtdsFile* function should be called.

ValidateNtdsFieldValue

Purpose: This function performs the following task(s):

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

Prototype:

```
STATUS  ValidateNtdsFieldValue(
                                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 processed. General error.
-11	No channel has been selected.

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 *StopValidatingNtdsFile* function should be called.
- This function inherits the validationLevel commenced with the *StartValidatingNtdsFile* function as modified by ignoreRequiredChecks in *ValidateNextNtdsRecord*. These settings cannot be overridden.
- **Note**: this function only performs schema checks.

GetNtdsFieldValue

Purpose: This function performs the following task(s):

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

Prototype:

```
STATUS  GetNtdsFieldValue(
                    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 processed. General error.
-11	No channel has been selected.

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 *StopValidatingNtdsFile* 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.

StopValidatingNtdsFile

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

Prototype: VOID StopValidatingNtdsFile()

Parameters: None.

Return Value: None.

Remarks:

- Should not be used unless *StartValidatingNtdsFile* has already been successfully called.
- Must be called before *ShutdownNtdsSdk*.

StartNtdsAggregateComputing

Purpose: Using the file and channel already opened by *StartValidatingNtdsFile* and *OpenNtdsChannel*, this function marks the starting point in the file where Aggregate processing will commence.

Prototype:
```
VOID StartNtdsAggregateComputing(
                                    USHORT validationLevel
                                    )
```

Parameters:

Name	Type	Description
validationLevel	USHORT	[in]] Valid entries are: • 0 = No validation • 1 = Schema errors only • 2 = Schema + Inclusion Logic • 3 = Schema + Inclusion + Major Logic • 4 = Schema + Inclusion + Major + Minor Logic

Return Value: None.

Remarks:

- Can be called after successful *OpenNtdsChannel*.
- Should not be called again until after a successful call with a corresponding *StopNtdsAggregateComputing*.
- Message buffers can be read at any time using the *NumNtdsAggregateMessages* and *GetNtdsAggregateMessageEntry* functions.
- Message buffers can be cleared or reset with *ClearNtdsAggregateData*.

StopNtdsAggregateComputing

Purpose: Ceases to generate message buffers holding Aggregate message buffer data.

Prototype: VOID StopNtdsAggregateComputing()

Parameters: None.

Return Value: None.

Remarks:

- Should not be used unless *StartNtdsAggregateComputing* has been successfully called.
- Must be called before *CloseNtdsChannel*.
- Performs an implicit clearing of the message buffers.

ClearNtdsAggregateData

Purpose: Clears the message buffers generated after a successful *StartNtdsAggregateComputing*.

Prototype: VOID ClearNtdsAggregateData()

Parameters: None.

Return Value: None.

Remarks:

- Cannot be called until after a successful *StartNtdsAggregateComputing*.
- Cannot be called after a successful *StopNtdsAggregateComputing*.
- Typical usage of this function would be to reset aggregate counts or processing after a specific subset of records has been parsed.

NtdsFileVersion

Purpose: Retrieves the version of the NTDS file stored in the tag <NTDSVersion> (i.e. <xs:attribute name="NTDSVersion" use="required" fixed="v2010.1.5"/>).

Prototype:

```
STATUS  NtdsFileVersion (
                        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 the Buffer to be used above.

Return Value: Return value is of type STATUS.

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

Remarks:

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

NumNtdsMessages

Purpose: 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  NumNtdsMessages(
                        USHORT messageType
                        )
```

Parameters:

Name	Type	Description
messageType	USHORT	[in] Valid enumerated input value(s): • 0 = all (UPPER all of these) • 1 = schema • 2 = inclusion logic • 3 = major logic • 4 = minor logic

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 level that have not been captured with the options parameter in *StartValidationNtdsFile*, will always return 0.
- The number of error messages retrieved would normally be used in conjunction with the *GetNtdsMessageEntry* function.
- It is ok to call this function in between calls of *ValidateNtdsFieldValue*, so that messages specific to the last call to this function can be retrieved.

GetNtdsMessageEntry

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

Prototype:
```
STATUS GetNtdsMessageEntry (
                    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	UNIT&	[out] Valid enumerated input value(s): • 0 = all • 1 = schema • 2 = inclusion logic • 3 = major logic • 4 = minor logic
messageCode	UINT	[in] Should be specified as 0. Other values are reserved for future usage.
messageId	UINT&	[out] Contains the Rule ID integer / error code associated with the message. (See Appendix 2 or NTDS 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 was successful.
-11	No channel selected.
-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 *GetNtdsMessageEntry* 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 *NumNtdsMessages* function (using the ALL parameter).

NumNtdsAggregateMessages

Purpose: Retrieves the number of aggregate error messages of a specified type that have been buffered during the commencement of *StartNtdsAggregateComputing*.

Prototype:
```
ULONG NumNtdsAggregateMessages(
                    USHORT messageType
                    )
```

Parameters:

Name	Type	Description
messageType	USHORT	[in] Valid enumerated input value(s): • 0 = all (UPPER all of these) • 1 = schema • 2 = inclusion logic • 3 = major logic • 4 = minor logic

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 level that have not been captured with the options parameter in *StartNtdsAggregateComputing*, will always return 0.
- The number of error messages retrieved would normally be used in conjunction with the *GetNtdsAggregateMessageEntry* function.

GetNtdsAggregateMessageEntry

Purpose: Retrieves the buffered error message information related to the Aggregate XML records that have been parsed to date.

Prototype:

```
STATUS GetNtdsAggregateMessageEntry(
                    ULONG whichMessage,
                    UNIT& messageType,
                    UNIT 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	UNIT&	[out] Valid enumerated input value(s): • 0 = all • 1 = schema • 2 = inclusion logic • 3 = major logic • 4 = minor logic
messageCode	UINT	[in] Should be specified as 0. Other values are reserved for future usage.
messageId	UINT&	[out] Contains the Rule ID integer / error code associated with the message. (See Appendix 2 or NTDS 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 was successful.
-11	No channel selected.
-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 records to date, then *GetNtdsAggregateMessageEntry* 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 *NumNtdsAggregateMessages* function (using the ALL parameter).

GetNtdsMessageEntryTagName

Purpose: Retrieves the tag name associated with the buffered error message information retrieved by *GetNtdsMessageEntry*.

Prototype:
```
STATUS GetNtdsMessageEntryTagName (
                                    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 was successful.
-11	No channel selected.
-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 *GetNtdsMessageEntryTagName* should not be called.

ValidateNtdsFile

Purpose: Validates an entire source XML file at one time.

Prototype:

```
LONG_STATUS  ValidateNtdsFile (
                                LPCSTR xmlFilename,
                                LPCSTR logFilename,
                                LPCSTR  options,
                                LOGICAL& eofFlag,
                                USHORT validationLevel
                                )
```

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.
validationLevel	USHORT	[in] Valid entries are: • 0 = No validation • 1 = Schema errors only • 2 = Schema + Inclusion Logic • 3 = Schema + Inclusion + Major Logic • 4 = Schema + Inclusion + Major +Minor Logic

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 *InitializeNtdsSdk* function must precede it and the *ShutdownNtdsSdk* function would normally come after it.

ValdateNtdsFileWithModeOptions

Purpose: This function is similar in it's usage to ValidateNtdsFile but can work at either a record-by-record level or at a file level (i.e. includes aggregate Rule checks) or both.

Prototype:

```
LONG_STATUS  ValidateNtdsFile (
                    LPCSTR xmlFilename,
                    LPCSTR logFilename,
                    LPCSTR  options,
                    LOGICAL& eofFlag,
                    USHORT validationLevel,
                    USHORT validationMode
                    )
```

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.
validationLevel	USHORT	[in] Valid entries are: • 0 = No validation • 1 = Schema errors only • 2 = Schema + Inclusion Logic • 3 = Schema + Inclusion + Major Logic • 4 = Schema + Inclusion + Major +Minor Logic
validationMode	USHORT	[in] Valid entries are: • 1 = Record Level • 2 = File Level • 3 = Both

Return Value: Return value is of type: LONG_STATUS and will return the number of records processed (if successful), and a negative value otherwise (-1 for Failed and -2 for Invalid File).

Remarks:

- It is expected that over time this function will be used over and above *ValidateNtdsFile* since it offers the option to include aggregate checking as well as record level checking. However, *ValidateNtdsFIle* is being kept to maintain a consistent interface with code already developed by the Trauma Registry community.
- A channel must be open before using this function.

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 NTDS edit checks specified by the ACS-COT (as well as NTDB edit checks, as NTDS 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 NtdsSdkSupport@dicorp.com.

DI maintains a distribution list of developers who are interested in receiving the SDK and participating in various NTDS 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.

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

As with previous versions of the Validator, this release of the NTDS SDK has complete support for the error messages defined in the NTDS Data Dictionary – Appendix 2: Edit Checks. The SDK supports the Rule ID and Message text for the Edit Checks as well as Error Levels – a scheme that categorizes all errors on a scale of 1 to 5.

The Error Level scheme is important to developers and to users alike and should be used to facilitate deciding what Rules (or Errors) must be addressed before submitting to NTDB. Some errors are mandatory to address and some are somewhat discretionary. Ultimately the number of errors resolved in the submitted data is up to the individual submitter and the quality of data that is available for reporting and research in NTDB.

The Error Levels can be explained as follows:

1. Format / Schema[1] – any error that does not conform to the "rules" of the XSD. That is, these are errors that arise from XML data that cannot be parsed or would otherwise not be legal XML. Some errors in this Level do not have a Rule ID – for example: illegal tag, commingling of null values and actual data, out of range errors, etc.
2. Inclusion Criteria[1] – an error that affects the fields needed to determine if the record meets the inclusion criteria for NTDB.
3. Major Logic – data consistency checks related to variables commonly used for reporting. Examples include DOB, Arrival Date, Gender, ECode, etc.
4. Minor Logic – data consistency checks (e.g. dates) and blank fields that are acceptable to create a "valid" XML record but may cause certain parts of the record to be excluded from future research inclusion. For example, an intentionally blank Total Ventilator Days field is ok to submit but a researcher interested in doing a study on outcomes based on records with this field being valued would not include records with blanks.

Process Measure/TQIP Rules start in the '10000' range so Rule IDs can now use up to 5 digits.

Aggregate Rules have their own range and all start with a '99'. Error Levels are the same as for the standard API Rules. In the output produced by the API record numbers and patient ID references are set to all 9s since specific record references are no longer relevant. Aggregate Rules are still not in active use yet until such time as ACS and the Vendor Community agree on how best to use them.

[1] Any XML file submitted to NTDB that contains one or more Level 1 or 2 Errors will result in the entire file being rejected. These kinds of errors must be resolved before a submission will be accepted.

Appendix 2: XSD Implementation Notes

The creation of a new dataset will always result in iterative improvements over time. This section is intended to be "FAQ-like" and capture notes of clarification as the SDK and the XSD are used for the first time.

BIU Notation

The opening sections of the NTDS Data Dictionary v2.2 outline the usage of "Common Null Values". These are the various implementations of the meaning of a null stored in one or more of the fields. Not all data elements have the notation implemented but many do. In the Data Dictionary the property "Accepts Null Values" denotes whether the Common Null Values are enabled or not. In the XSD this is described as an attribute on the data element called "biu". This attribute means "blank inappropriate unknown" and is used as a shorthand reference to the Common Null Values. Valid values are 1 or 2 only – the Validator will flag any other values.

Facility ID

The value used in this "undocumented" data element should use the Facility IDs produced by the National Trauma Data Bank (NTDB). For any given implementation of a trauma XML file for uploading to NTDB, this data element should be set to the Facility's NTDB ID for each record. If the Facility ID is not known, use the contact details on www.ntdbdatacenter.com to get in touch with the appropriate folks.

Patient ID

The intention of this "undocumented" data element is primarily to match the output from the Validator with the source data being used to create the XML file. For this reason, this Patient ID can typically be any unique identifier given to the source trauma record in the application. However, developers should be aware that this Patient ID is used by NTDB to identify and flag duplicate records in the database. It is common practice to re-submit records that have been submitted previously whether they have been updated or not. The ability to match duplicates based on Facility ID, Patient ID and LastModifiedDateTime ensures the integrity of the dataset as a whole.

Last Modified Date / Time

This "undocumented" data element should contain the most recent date/time stamp from the application or Registry that produced the patient record. The LastModifiedDateTime element is used to ensure that duplicates of the same record at the NTDB are removed or ignored from further processing.

NtdsVersion attribute

This attribute is required and as of 2017 has the value of either "Ntds_v2017" or "Tqip_v2017".

Icd10 attribute

This attribute is optional. If used it has one value of "ICD10CA" (ICD-10-CA – Canada). Specifying this attribute in the xml will make the Validator apply the ICD-10-CA coding system instead of the default ICD-10-CM.

Appendix 3: Notes on the NTDS XSD

Until version 2010.5 the NTDS and XSD have been kept in sync with regards to Error Level 1 Schema checks and how the XSD would behave in standard W3C validation. However, with the introduction of multiple channels and the anticipated arrival of a new channel for TQIP Process Measures, it is has been decided to "relax" some of the specific checks that the XSD performs – most notably in the area of minOccurs = 1, making most elements mandatory. This shift enables the evolution of different Validators (State and National) and rule sets without having multiple XSDs. So now the XSD's primary purpose is to define the structure of a conforming XML and the specific rules governing the compliance and logic of the data will be left to the related Validator.

Known areas of difference between the Validator and the XSD include:

- **Out of Range values** – The XSD has always fixed ranges to be much larger than necessary. The Validator has "conditional ranges" that offer more flexibility. For example, Respiration Rate for an adult is very unlikely to go beyond 100 but for an infant this is possible, so the range for an infant is conditionally higher than for an adult.
- **Date/Time elements** – The default format of a date for an XSD is the W3C standard. However the Validator accept both W3C and non-W3C standards. For example a date of '20010101' would be rejected by the XSD but is acceptable for the Validator.
- **Sequencing** – XSD no longer requires elements to be in a defined order. The Validator does not either. However, it is preferred for readability reasons to retain the order specified in the XSD and the Data Dictionary.