# A Windows DLL and sample C and VB programs for IMDG Code Substances and Calculations (TUPS Libraries)

Author: Exis Technologies Limited, March 2021

This is a toolkit for software developers wanting to integrate IMDG Code functionality into their own in-house cargo management systems. In this transition year it can be used with a data file for either IMDG Amendment 39-18 or 40-20.

With these routines you can retrieve substance data from the **Dangerous Goods List** (chapter 3.2). Fields such as Name, Class, Marine Pollutant, Special provisions, Packing instruction number, Limited Quantity, Stowage and Handling, Segregation, Properties and Observations are available. These records are retrievable by UN Number and variant. 'Next' and 'Previous' UN Numbers are also locatable. You can also retrieve the general stowage requirements from IMDG part 7 for a substance, according to its class, subhazards, packing group, etc.

You can also check the **stowage and segregation requirements** for several substances in a mixed load; whether it is **prohibited**, allowed **on deck only**, or on or under deck, on a **passenger** or **cargo** sailing; and whether the mix of substances need to be segregated from one another. Segregation is according to the classes and subsidiary hazards of section 7.2.4 and 7.2.7, the segregation groups of chapter 3.1, special requirements from the Dangerous Goods List such as the SG codes indicating 'segregation as for class xxx' or 'separated from xxx', and other rules and exceptions of chapter 7.2. The degree of **segregation required between two loads** can also be obtained.

The **substance index** data is also available if it is included in the data file.

The text of the Special Provisions of chapter 3.3, and the Packing, IBC and Tank Instructions and Provisions of chapters 4.1 and 4.2 are retrievable by name.

The 32-bit windows DLL is called ctups.dll. Following normal windows conventions, the routines use the WINAPI calling convention.

There are sample programs with source code written in C and VB. These are intended to show very simply how the routines can be called rather than to be the basis of a complete end-user system. We also supply ctups.h (for C) and tupsdecl.bas (for VB) for you to include in your own programs to define all the routines and constants. For calling from other languages you can use these as a guide to write your own equivalent. C programs should be linked with the import library ctups.lib (unless you are doing LoadLibrary and GetProcAddress calls). VB programs need to find the dll, for instance by putting it into the <system32> directory or your program's working directory.

A similar C# DLL for .NET users and a Java program (Subsdemo.java) is also available. They use different data files and include other features such as validating packaging types and sizes according to the packing instruction.

The C data file is around 210K in size, or 290K with the substance index data. It is read into memory at startup, and subsequent calls will not require disk access. In speed tests it can do a segregation check of around 35000 two-item loads per second on an Intel Core2 quad core

2.66GHz processor. The time is dominated by the two substance lookups per load.

Either IMDG Amendment 39-18 or 40-20 may be used operationally during the transition 'year' of 2021 and part of 2022. 40-20 becomes mandatory on 1 June 2022. Amendment 41-22 is due to be published in late 2022, and the year 2023 will be the next transition period during which either Amendment can be used. Amendment 41 will require another data file, DLL and a ctups.h change, but Exis always attempts to maintain upwards compatibility. There is generally a small errata or corrigenda update published about one year after the Amendment; these can be expected to need a new data file but only rarely a DLL or H file change.

**A note for users upgrading from Amendment 39-18**

To allow for longer fields being given to UN 3481, the string 'specprov' has been extended in ctups.h. Existing programs using ctups.h will need to be recompiled to use this DLL and the amdt 40 data file. The new DLL can also be used with the existing amdt 39 data file and will then calculate using the amdt 39 rules.

The old DLL should not be used with the amdt 40 data file. It may fail accessing some UN Numbers and the segregation results will not be according to amdt 40 rules.

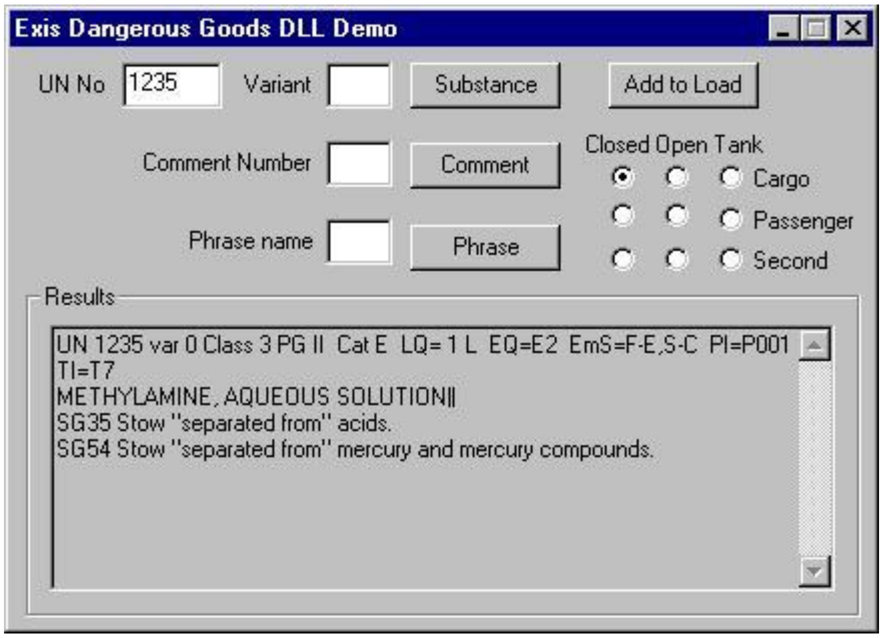
## The download package

In the zip file you have 12 files:

| | |
|---|---|
| readme.pdf | these instructions |
| ctups.dll | the DLL |
| ctups.h | the H file to include in your C program |
| ctups.lib | the import library to link with your program |
| subs.ind | the amendment 40 data file |
| | |
| subsdemo.exe | a simple demonstration program written in C |
| subsdemo.cpp | the source for subsdemo.exe |
| subsdemo.rc | the resource script for subsdemo.exe |
| makedemo.bat | the commands to compile and link subsdemo |

## Getting started with subsdemo.exe or VBexample.exe

The exe files, ctups.dll and subs.ind should be in the same directory.  At first you can only look up UN Numbers ending in 5. Run subsdemo.exe (for a C example) or vbexample.exe (for VB).



UN No: 1235

Closed / Cargo selected

```
UN 1235 var 0 Class 3 PG II  Cat E  LQ= 1 L  EQ=E2  EmS=F-E,S-C  PI=P001
TI=T7
METHYLAMINE, AQUEOUS SOLUTION||
SG35 Stow "separated from" acids.
SG54 Stow "separated from" mercury and mercury compounds.
```

To look up a substance, type 1235 in the UN Number box, and press 'Substance'. You see the primary class, packing group, stowage category, limited quantity value, emergency schedule and packing instruction for Methanol, and the stowage and segregation comments. All the other DGL fields are available, but subsdemo is only meant to show the principal of how things are done.

```
UN 1235 var 0 Class 3 PG II  Cat E  LQ= 1 L  EQ=E2  EmS=F-E,S-C  PI=P001 TI=T7
METHYLAMINE, AQUEOUS SOLUTION||
SG35 Stow "separated from" acids.
SG54 Stow "separated from" mercury and mercury compounds.
```

Type 2545 and press 'Substance'. You see variant 1 of HAFNIUM POWDER, DRY with a variation description 'Packing group I.' after the name. Type 2 into the variant box, press 'Substance' again, and see the second variant, packaging group II.

To look up a packing provision, type PP4 in the phrase name box and press 'Phrase'.

> Packagings shall meet the packing group II performance level.

To see a special provision, type 191 into the box and press 'Phrase' again.

> Receptacles with a capacity not exceeding 50 mL containing only non-toxic constituents are not subject to the provisions of this Code.

Examples of other phrases are: packing instruction (P302), IBC instruction (IBC02), tank instruction (T3), IBC provision (B6), tank provision (TP10).

To check for segregation requirements within a load, start a load by clicking the radio button for 'Closed / Passenger'. Type 3295 in the UN Number box, 1 in the variant box, and press 'Add to load'. You see return code 98 - this substance is not allowed on passenger sailings. Start another load by clicking the radio button for 'Closed / Cargo'. Press 'Add to Load', and this time it is allowed (return code 0). The label numbers (bitsfields 20 and 1) are decoded as primary class 3, and a possible marine pollutant (refer to the description of function TupsLabCom). The other numbers are stowage and segregation comments for the load so far. Type one of the numbers such as 162 into the comment number box and press Comment to see For marine pollutants, under deck stowage is preferred., a requirement from IMDG 7.1.4.2 for marine pollutants which may be stowed under deck. Add another substance by putting 3285 and 2 in the UN and variant boxes, and press 'Add to Load'. There is no clash, and class 6.1 (bitfield 800) is added to the labels. Add a few more substances (e.g. variant 0 of substances 1345, 1515, 1945), and you will start to see clashes - a set of 3 numbers means substance sequence number 'i' clashes with number 'j', and the degree of segregation required is 'k'. Start a new load by clicking Cargo/Tank and you may also see return code 97 - this substance is not allowed in tanks (e.g. UN 1945).

Having built one load you can check how it should be segregated against another one. Start the second load by clicking the radio button for 'Closed / Second', and add a few substances to that. In this case the set of 3 numbers means that substance sequence number 'i' in load one clashes with number 'j' in load two and the degree of segregation required is 'k'. For a more complete explanation refer to the technical document section "Compatibility between loads".

## Recompiling the sample programs

Subsdemo is a small, simple windows program. If you want to add more features, edit the .cpp or .rc file. makedemo.bat compiles and links the program. This works with Microsoft C++ versions 5 and 6. Slight changes may be needed for other compilers. It assumes that 'cl', 'rc' and 'link' are to be found in the path, an that environment variables INCLUDE and LIB point to the standard .h and .lib files.

VBexample is an equivalent program in VB, which is most easily worked with inside Visual Studio. If you want to run the program from inside Visual Studio, it is easiest to uncomment the 'ChDrive' and 'ChDir' calls in 'Sub Form_Load', and set the values to your own working directory.

## Specification of the routines

### Initialization

**int TupsInit(char *name)**
Function TupsInit (ByVal fname As String) As Long

This should be called first. It opens the substance file and performs various initializations.

| | |
|---|---|
| **name** | the filename, "subs.ind", with any necessary directory specification. |
| **return value** | -1 = Can't open file<br>-2 = Can't get enough memory (unlikely as subsdemo runs in only 4 Mb)<br>-28 = Unactivated copy.<br><br>+ve = OK |

With an unactivated copy and other protection failures, the system works in 'demo mode'. This only accepts UN Numbers ending in 5.

### Substance Lookups

**int TupsSub(int un, int suffix)**
Function TupsSub (ByVal un As Long, ByVal suf As Long) As Long

Finds a substance on file.

| | |
|---|---|
| **un** | UN Number |
| **suffix** | suffix |

**Note:** Where a UN Number has no variants, it will have suffix 0 (e.g. UN 2050, suffix 0). Those with variants are numbered 1 upwards (e.g. 2810 suffixes 1, 2 and 3 are for packing groups I, II and III). A search for suffix 0 or 1 will equally find either of 0 and 1. Alternatively, suffix may be -1 or -2, to find the respectively the next or previous UN Number (and suffix 0 or 1) to the one specified.

| | |
|---|---|
| **return value** | 99 if no such substance<br>0 = OK |

**int TupsRead(int field, char *buf)**

```
Sub TupsInfo (ByVal field As Integer, buffer As Any)            'used for TUPS_INFO and
                                                                'TUPS_STOW*
Sub TupsRead (ByVal field As Integer, ByVal buffer As String)   'used for other values
```

Reads data for the substance last found with TupsSub. 'buf' can be variously a pointer to a char array, an array of shorts or a 'struct tups' into which the data will be copied. Note that the user must provide a buffer of sufficient size.

**field** TUPS_NAME retrieves Proper Shipping Name, qualifying descriptive text and variation comments. A '|' character (ascii 124) separates the three items, and a NULL terminates it. The string may include embedded linefeed characters.

TUPS_STOW retrieves (as an array of shorts, terminated by a zero) the stowage and segregation comment codes from columns 16a and 16b of the Dangerous Goods List. See TupsCom for converting the codes to printable phrases.

TUPS_STOWC retrieves general stowage comment codes applicable to cargo ships, for this class, subrisks, packing group, etc., from part 7 of the IMDG Code.

TUPS_STOWP ditto for passenger ships.

TUPS_PROP retrieves as text the properties and observations from column 17 of the Dangerous Goods List.

TUPS_INFO retrieves assorted data from the Dangerous Goods List:-

```
struct tups
{       short un;                 // UN Number
        short suf;                // suffix (variant number)
        char prim[5];             // primary class
        char secn[6];             // secondary class (subrisk)
        char tert[4];             // tertiary class
        char mp;                  // Marine pollutant (Y or null)
        char ulineems;            // Underlined EmS. 1 = fire, 2 = spillage, 3 = both
        char ems [8];             // Emergency Schedule
        char classcode[5]         // ADR classification code
        char flags;               // additional user data required - see below
        char specprov[40];        // special provisions
        char lq[10];              // limited quantity
        char pi[49];              // packing instructions
        char pp[26];              //     and provisions
        char ibci[11];            // IBC instructions
        char ibcp[15];            //     and provisions
        char tankun[16];          // UN tank instructions
        char tankimo[1];          // IMO tank instructions, no longer in use
        char eq[8];               // excepted quantity
        char tankp[25];           // tank provisions
        char flash[26];           // flashpoint
        char explim[20];          // explosive limits
        char stow[15];            // stowage SW and H codes
        char seg[55];             // segregation SG codes
        char cvl[8];              // Coded Variant List value, used in EDI files
        char scat[3];             // stowage category, A to E or 01 to 05
        char pg[4];               // packing group, null, I, II or III
        char state;               // S,L,G,E,A for solid, liquid, gas, explosive
                                  //     substance or explosive article
};
```

which is declared in an H file. Note this has changed from what was used in previous amendments, in order to accommodate the changes to the lengths of the special provisions of UN 3481 .
or 'Type tups' which is declared in tupsdecl.bas
Note that there are 2 versions of this routine for VB, 'TupsRead' to return a string, and 'TupsInfo' to return an array or a 'tups' structure.

The value returned in 'flags' indicates whether any additional information is required from the user to print on a Dangerous Goods Note or manifest. It is the combination of:-

| | |
|---|---|
| 1 | Technical name required (special provision 274 and others) |
| 2 | Possible marine pollutant |
| 4 | Flashpoint required (class 3 or subrisk 3) |
| 8 | Control and emergency temperatures (items under temperature control) |
| 16 | Radionuclide, category, activity and transport index required (class 7) |
| 32 | May be carried in limited quantity |
| 64 | Net explosive content required (class 1) |
| 128 | Warn user about special provision 900 and similar (certain formulations are prohibited) |

**int TupsCom(short code, char *cbuf)**
Function TupsCom (ByVal code As Integer, ByVal buffer As String) As Long

Obtains the comment text related to a numeric comment code. The text will be null-terminated and may include linefeed (ascii 10) characters.

| | |
|---|---|
| **code** | code number (retrieved from TupsRead) to look up |
| **cbuf** | char array into which the phrase will be copied |
| **return value** | 0 = OK<br>1 = no such comment. |

**int TupsPhrase(char *code, char *cbuf)**
Function TupsPhrase (ByVal name As String, ByVal buffer As String) As Long

Obtains the text related to a text comment code. The text will be null-terminated and may include linefeed (ascii 10) characters. The codes may be packing instructions Pxxx, IBC instructions IBCxx, tank instructions Txx, packing provisions PPxx, Bx or TPxx, or special provisions, numeric. Provisions retrieved by TupsRead may include several codes (e.g. "PP25 PP26 PP31") separated by spaces. It is for the user to split these into separate null-terminated codes for this routine.

| | |
|---|---|
| **code** | phrase name to look up |
| **cbuf** | char array into which the phrase will be copied |
| **return value** | 0 = OK<br>1 = no such comment. |

Hint: If the user enters a UN number, call TupsSub for that UN number and suffix zero. For a return code of 99 tell him 'invalid UN number'. Otherwise, if it retrieves suffix zero, there are no variants and you can show him the substance details. If it retrieves variant 1, call TupsRead(TUPS_NAME... to get the description of the first variant, then TupsSub and TupsRead for variants 2, 3, etc. until a return of 99 indicates no more variations. Ask him to select one.

**void TupsDistance(short opclo, short degree, short shiptype, short ondeck, short *foraft, short *athwart, short *notes)**

Gives the separation distances required for a given segregation value, according to the ship and CTU type.

| | |
|---|---|
| **opclo** | open/closed status, 0=closed/closed, 1=closed/open, 2=open/open |
| **degree** | 1 = away from, 2 = separated from, etc. |
| **shiptype** | 1 = container ship, 2 = hatchless container ship, 3 = ro-ro |
| **ondeck** | 0 = CTUs under deck, 1 =CTUs on deck |

Sets up values in foraft and athwart for distances fore-and-aft and athwartships in metres (ro-ro) or container spaces (container ships), but value 24 means metres even on container ships, 99 means prohibited in this direction.

notes is set to 0, no note,
1 = or 1 bulkhead (and no distance restriction)
2 = and on a different deck
3 = 2 decks or 2 bulkheads
4 = this distance and 2 decks, or 2 bulkheads and no distance restriction

5 = and 1 bulkhead
6 = and 2 bulkheads
7 = and not in or above same hold

## Checking a Single Load

**void TupsNewLoad(int mode, int type, short *clashlist, int max_clash)**
Sub TupsNewLoad  (ByVal mode As Long, ByVal utype As Long, clist, ByVal les As Long)

For validation of a single load, this empties the current load, and specifies whether it is to be checked for passenger or cargo sailings, and open or closed units, or a tank.

| | |
|---|---|
| **mode** | TUPS_PSNGR  for a passenger sailing<br>TUPS_CARGO  for a cargo sailing<br>and may be or'ed with<br>TUPS_NOCHECK  to suppress validation as each substance is added. |
| **type** | TUPS_OPEN<br>TUPS_CLOSED<br>TUPS_TANK according to the unit type |
| **clashlist** | array of shorts which becomes filled with triplets describing each clash<br>a) sequence number of first substance counting from 1<br>b) sequence number of second substance<br>c) degree of clash - see values 1 to 10 for TupsAddSub<br>The list is terminated by a zero value. With a single load (a) < (b).<br>With VB, pass the first element of the array, clashlist(0) |
| **max_clash** | maximum number of clashes to report. Filling of clashlist stops when this is reached. Thus clashlist can be dimensioned to [3*max_clash +1]. |

**int TupsAddSub(int un, int suffix)**
Function TupsAddSub(ByVal un As Long, ByVal suf As Long) As Long

Adds a substance to the load, and (unless TUPS_NOCHECK applies) validates it against the load so far. Details of clashes with previous substances will appear in 'clashlist'

| | |
|---|---|
| **un** | UN number |
| **suffix** | suffix |
| **return value** | 99 if no such substance<br>98 if substance prohibited (entirely, or on this passenger sailing).<br>97 if not valid in a tank / must be in a tank, when this unit isn't / is a tank.<br>96 if this explosive is not allowed in this open unit.<br>88 if failed to expand working space (unlikely with 32-bit. 16-bit versions were limited to about 440 substances in a single load).<br>1 to 4   highest clash value if this substance clashes with others already in the load.<br>0 if no clash or if TUPS_NOCHECK is set. |

With a return value of 88 upwards, the substance is not added to the load, and so is not checked against future substances which will be added.

Return values 1 thru 4 are 'away from', etc., as in section 7.2 of the IMDG Code.

| | |
|---|---|
| 1 | "Away from" |
| 2 | "Separated from" |
| 3 | "Separated by a complete compartment or hold from" |
| 4 | "Separated longitudinally by an intervening complete compartment or hold from" |

We formerly also used numbers 5 to 10 in relation to explosives, but these provisions are removed from recent IMDG amendments

| | |
|---|---|
| 5 | If UNDER DECK – "Separated from". If ON DECK - not less than 6 metres apart. |
| 6 | As 4, and also as remote as possible from each other. |
| 7 | As 4, and also as remote as possible from each other, and not more than 50 kg total net explosive mass per ship. |
| 8 | As 4, and also as remote as possible from each other, and not more than 10 kg total net explosive mass per ship. |
| 9 | Not on same ship |
| 10 | If UNDER DECK – As 4. If ON DECK - not less than 6 metres apart. |

**Note:** In setting the highest clash value, 5 is more severe than 2, but less than 3.

Hint: Items carried in limited or excepted quantities (see IMDG chapters 3.4 and 3.5) are allowed on or under deck on both passenger and cargo sailings, need not bear hazard labels, and need not be segregated from other incompatible goods. Don't call TupsAddSub for limited or excepted quantity substances.

**int TupsAddSubGrp(int un, int suffix, char *grps)**
Function TupsAddSubGrp(ByVal un As Long, ByVal suf As Long, ByVal grps As String) As Long

As TupsAddSub but with the possibility of specifying additional segregation groups the substance belongs to

| | |
|---|---|
| **un** | UN number |
| **suffix** | suffix |
| **grps** | null-terminated list of groups. Group numbers 1 to 18 are represented by digits 1 to 9 then letters A to I (or a to i), or byte values 1 to 18. |
| **return value** | as TupsAddSub |

**int TupsEditLoad(int seq, int un, int suffix)**
Function TupsEditLoad (ByVal seq As Long, ByVal un As Long, ByVal suf As Long) As Long

Changes a substance in a load, but does not recheck the validation.

| | |
|---|---|
| **seq** | sequence number within the load, numbered from 1 upwards. |
| **un** | UN number (or zero to remove it from the load) |
| **suffix** | suffix |

**return value** 99 to 96 as with TupsAddSub
0 = OK

**int TupsReCalc()**
Function TupsReCalc () As Long

Re-validates a load, if it was originally defined as TUPS_NOCHECK, or when one or several changes have been made with TupsEditLoad.

**return value** 0 if no clash
1 to 10 highest clash value if a clash.

**void TupsLabCom(short *lab_com)**
Sub TupsLabCom (labcom As Integer)

retrieves the aggregate list of labels and stowage comment codes for the items in the load.

|  |  |
|---|---|
| **lab_com** | filled with a list of consolidated label and comment data for the current load. |
| **lab_com[0]** | bitfields for primary labels |
| **lab_com[1]** | bitfields for subrisk labels |
| **lab_com[2...]** | comment codes. Use TupsCom to convert to phrases. |

With VB, pass the first element of the array, lab_com(0)

The bitsfields for labels represent (in hex):

|  |  |
|---|---|
| 0001 | marine pollutant |
| 0002 | explosive (1) |
| 0004 | flammable gas (2.1) |
| 0008 | non-flammable gas (2.2) |
| 0010 | toxic gas (2.3) |
| 0020 | flammable liquid (3) |
| 0040 | flammable solid (4.1) |
| 0080 | spontaneous combustible (4.2) |
| 0100 | dangerous when wet (4.3) |
| 0200 | oxidizing substance (5.1) |
| 0400 | organic peroxide (5.2) |
| 0800 | toxic (6.1) |
| 1000 | infectious (6.2) |
| 2000 | radioactive (7) |
| 4000 | corrosive (8) |
| 8000 | miscellaneous (9) |

Bit '1' for a primary label indicates a marine pollutant label is definitely required. '1' for a secondary label indicates one may be required, depending on concentration or composition (at least one item was 'marine pollutant bullet', but none were definite marine pollutants)..

lab_com[2] will always be 1 (prohibited), 2 (as approved by competent authority), 3 (on-deck only) or 4 (on or under deck). Some other values later in the list will restrict the load to various parts of a ship, e.g. 32 (Clear of living quarters) or 60 (Stow in a well ventilated space).

## Compatibility between loads.

Having built one load with TupsNewLoad and TupsAddSub, TupsNewLoad can be called again with a mode of TUPS_SECOND Tups_AddSub is then called for each substance in the second load.

**Note:**

a) if the first load was known to be valid, time is saved if it is built with a TUPS_NOCHECK flag.

b) further loads can be checked against the first with more TupsNewLoad / TUPS_SECOND calls.

c) TupsEditLoad, TupsReCalc and TupsLabCom are not appropriate with a TUPS_SECOND load.

d) in 'clashlist', (a) is the sequence number of an item within the first load, (b) within the second.

e) to check ten loads against each other build load one as a first load, then check it against loads two to ten in turn. Then build load two as a first load and check it against loads three to ten in turn, and so on, i.e. you need to use a nested loop.

## Substance index lookups.

Routines exist to locate the position at which a name in the substance index would be located, and to retrieve the entry at a particular location.

**int IndexPos(char *name)**

finds the row number (from 0 to about 4600) closest to position at which 'name' would exist. Any digits, spacing or punctuation in 'name' will be ignored to match IMO's sort order.

| | |
|---|---|
| **name** | name to look up. |
| **return value** | row number, or -1 if the index data was not included in the data file |

**char *IndexLine(short num)**

returns a pointer to the text of the entry at 'num', as if it were a variable length struct of unno[5]; class[5]; mp[2]; name[];

| | |
|---|---|
| **num** | row number to look up. |
| **return value** | index data, or NULL if num was out of range |

## Notes for VB users

In the above descriptions, 'short' means 'Integer', 'int' means 'Long', and 'char*' means String. The function declarations are thus:-

```
Function    TupsInit (ByVal fname As String) As Long
Sub         TupsRead (ByVal field As Integer, ByVal buffer As String)
Sub         TupsInfo (ByVal field As Integer, buffer As tups)
Function    TupsSub (ByVal un As Long, ByVal suf As Long) As Long
Function    TupsCom (ByVal code As Integer, ByVal buffer As String) As Long
Function    TupsPhrase (ByVal name As String, ByVal buffer As String) As Long
Sub         TupsNewLoad  (ByVal p1 As Long, ByVal p2 As Long, clist, ByVal les As Long)
Function    TupsAddSub(ByVal un As Long, ByVal suf As Long) As Long
Function          TupsAddSubGrp(ByVal un As Long, ByVal suf As Long, ByVal grps As String) As
Long
Function    TupsEditLoad (ByVal seq As Long, ByVal un As Long, ByVal suf As Long) As Long
Function    TupsReCalc () As Long
Function    TupsEnd () As Long
Sub         TupsLabCom (labcom As Integer)
Function    TupsClashes (clashes As Integer, ByVal maxnum As Long) As Long
Function    Tups_Start (ByVal code As Long) As Long
Sub         TupsDistance (ByVal opclo As Integer, ByVal degree As Integer, ByVal shiptype As
            Integer, ByVal ondeck As Integer, foraft As Integer, athwart As Integer, notes As
            Integer)
```

To pass an array to the DLL, pass the first element. So for example, the call to TupsLabCom is

```
Dim labcom(40) as Integer
call TupsLabCom(labcom(0))
```

There is an extra declaration 'TupsInfo', which is the same as TupsRead, but with the second argument of type 'tups' so you can call

```
TupsInfo (TUPS_INFO, details)
```

Null-terminated strings are not handled nicely in VB, so the sample program includes a

```
Function sTrim(s As String) As String
```

which converts a null-terminated string to a VB String

## Contacting Exis Technologies Limited

If you have any questions or comments please contact Exis Technologies Limited by,

| | |
|---|---|
| email: | support@existec.com |
| telephone: | +44 (0)1325 466672 |
| fax: | +44 (0)1325 466643 |

## Sample Source Code - C and VB

This code shows data being retrieved and displayed using printf calls. A windows program would show the data in dialog boxes.

a) Open the file

```
if (TupsInit("subs.ind") == -1)
        {printf("subs.ind not found\n"); return 2; }


ChDir ("\tupslib") ' find the DLL and data file in this directory
ret = TupsInit("subs.ind")
results.Text = "TupsInit returned " + CStr(ret)
```

b) Show various details of UN 2045

```
short labc[20], clist[31];
int un, i;
struct tups info;
char buf[700];

ret = TupsSub(2045,0);
if (ret ===0)
{       TupsRead(TUPS_INFO, (char *)&info);
        TupsRead(TUPS_NAME, buf);
        printf(" UN %04d %d EmS = %s PI = %s\n Name = %s\n\nStowage:- ",
                info.un, info.suf, info.ems, info.pi, buf);
        TupsRead(TUPS_STOW, (char *)labc);    // stowage from DGL
        for (i=0; labc[i] > 0; i++)
        {       TupsCom(labc[i],buf);
                printf(" %s\n", buf);
        }
        printf("\nCargo:- ");
        TupsRead(TUPS_STOWC,(char *)labc);    // general stowage for
        for (i=0; labc[i] > 0; i++)           // this class and packing group
        {       TupsCom(labc[i],buf);
                printf(" %s\n", buf);
        }
        TupsRead(TUPS_PROP,buf);
        printf("\n Properties and Observations %s\n",buf);
}
```

```
Dim subsname As String, codes(20) As Integer, i As Integer
Dim details As tups
subsname = String$(5120, 0)
unno = txUNno.Text
suf = txSuf.Text
ret = TupsSub(2045, 0) ' look up the substance
If (ret <> 0) Then
   results.Text = "TupsSub returned " + CStr(ret)
```

```
    Else
        Call TupsInfo(TUPS_INFO, details) ' write various details
        Call TupsRead(TUPS_NAME, subsname)
        results.Text = "UN " + CStr(details.un) + " " + CStr(details.suf) + " EmS = " +
sTrim(details.ems) + " PI= " + sTrim(details.pi) + Chr(13) + Chr(10) + "Name= " +
sTrim(subsname)
        Call TupsInfo(TUPS_STOW, codes(0))
        i = 0
        While codes(i) > 0 ' stowage and segregation comments
            Call TupsCom(codes(i), subsname)
            results.Text = results.Text + Chr(13) + Chr(10) + sTrim(subsname)
            i = i + 1
        Wend
    End If
```

c) Special Provision  274

```
        ret = TupsPhrase("274",buf);
        if (ret == 0) printf(" %s\n", buf);


        Dim Comment As String
        Comment = String$(5120, 0)
        ret = TupsPhrase("274", Comment)
        If (ret = 0) Then
                results.Text = "Phrase 274 is " + sTrim(Comment)
        End If
```

d) Show variations of UN 3145

```
        un = 3145;
        ret = TupsSub(un,0);
        if (ret ==0)
        {       TupsRead(TUPS_INFO, &info);
                if (info.suf > 0)                               // have found variant 1
                {       printf("Variations exist\n");
                        i = info.suf;
                        while(ret == 0)
                        {       TupsRead(TUPS_NAME,buf);
                                p = strchr(buf, '|');  // start of descriptive text
                                p = strchr(p+1, '|');  // start of variation description
                                printf(" %d  - %s\n",i,p+1);
                                i++;
                                ret = TupsSub(un,i);   // look for next one
                        }
                }
        }
```

e) Check 3 items in a load

```
        TupsNewLoad(TUPS_CARGO, TUPS_CLOSED, clist,10);
        ret = TupsAddSub(1230,0);     // methanol
        i = TupsAddSub(1790,1);               // hydrofluoric acid, pg I
        if (i > ret) ret = i;
        i = TupsAddSub(1605,0);               // ethylene dibromide
        if (i > ret) ret = i;
        if (ret == 0)
        {       TupsLabCom(labc);
                printf("Load complies -  labels %04x %04x\n", labc[0], labc[1]);
                for (i=2; labc[i] > 0; i++)
                {       TupsCom(labc[i],buf);
                        printf(" %s\n", buf);  // stowage requirements
                }
        }
        if (ret > 0 && ret <= 10)
            printf("LOAD DOES NOT COMPLY Segregation of degree %d required\n", ret);
                // could use clist to say which ones give the problem
        if (ret > 10) printf("Load not valid");
```

f) Check a 2 item load against the previous one

```
        TupsNewLoad(TUPS_SECOND, TUPS_CLOSED, clist,10);
        ret = TupsAddSub(1065,0);     // neon
        i = TupsAddSub(1500,0);               // sodium nitrite
        if (i > ret) ret = i;
        if (ret == 0) printf("No segregation required\n");
        if (ret > 0 && ret <= 10)
            printf("Segregation of degree %d required between these 2 loads\n", ret);
                // could use clist to say which ones give the problem
        if (ret > 10) printf("Load not valid");
```

(Note that 1065 needs to be 'away from' 1230. However, since both units are 'closed', this requirement is automatically satisfied, and no clash is reported. 1500 needs to be 'separated from' both 1230 and 1790.)