

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

Author: Exis Technologies Limited, December 2007

This is a toolkit for software developers wanting to integrate IMDG Code functionality into their own in-house cargo management systems.

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 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 chapter 7.1 for a substance, according to its class, subrisks, packing group, etc.

You can also check the **stowage and segregation requirements** for several substances in a 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 risks of section 7.2.1.16 and 7.2.7, the segregation groups of chapter 3.1, special requirements from the Dangerous Goods List such as 'segregation as for class 5.1 but away from...', and other paragraphs of chapter 7.2. The degree of **segregation required between two loads** can also be obtained.

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. 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 <system> directory or your program's working directory. The C routines can also be supplied as a library for other operating systems.

A similar Java program (Subsdemo.java) is also provided. It uses the same data file and calls other .class files but does not use the DLL.

The data file is around 2 Mb in size. It is binary and portable between Windows/DOS, VMS, AS/400 and RS/6000 systems (non-8086 versions will flip 16-bit integers from low-high to high-low byte order as they read). Accessing a substance record requires just one disk 'seek'. Packaging Instructions and Consolidated Comments will take several.

The current version corresponds to IMDG Amendment 33-06, including the errata published in November 2007. This is usable during the transition year of 2007 and is mandatory from 1 January 2008. IMDG Code errata and future amendments will require changes to the data file and possibly to the routines also. Amendment 34-08 is due to be published in late 2008, and the year 2009 will be the transition period during which either amendment 33 or 34 can be used.

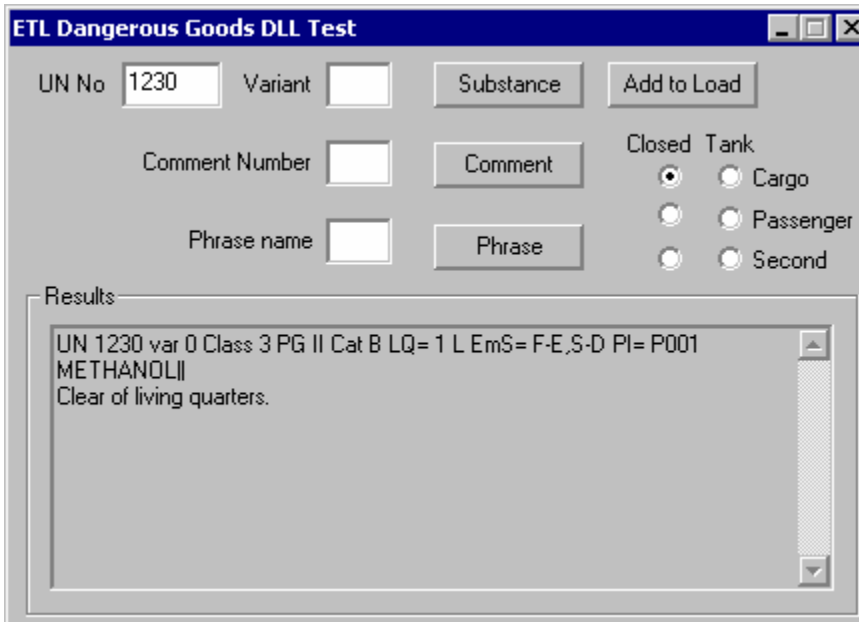
### The download package

In the zip file you have 20 files:

readme.rtf	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 data file (sample data only, only 20% of the substances)
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
VBexample.exe	a simple demonstration program written in VB
frmTups.frm	the source for vbexample.exe
tupsdecl.bas	declarations for VB
prjTupsExample.vbp	visual basic project
prjTupsExample.vbw	visual basic workspace
Subsdemo.java	a simple demonstration program written in Java swing
Subsdemo.class	)
Subsdemo\$1.class	) Java class files
DGLoad.class	)
DGSubs.class	)
Tups.class	)

### 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 0 or 5. Run subsdemo.exe (for a C example) or vbexample.exe (for VB).



To look up a substance, type 1230 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.

UN 1230 var 0 Class 3 PG II Cat B LQ= 1 L EmS= F-E,S-D PI= P001  
METHANOL||  
Clear of living quarters.

Type 2430 and press 'Substance'. You see variant 1 of ALKYLPHENOLS, SOLID, N.O.S., with qualifying descriptive text and a variation description 'Packing group I.' after the name. Type 2 into the variant box, press 'Substance' again, and see the second variant.

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

PP4 For UN 1774, 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 1790 in the UN number box, 2 in the variant box, and press 'Add to load'. You see return code 97 - 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 (bitfields 4000 and 800) are decoded as primary class 8, subrisk 6.1 (refer to the description of function TupsLabCom). Add another substance by putting 1230 and 0 in the UN and variant boxes, and press 'Add to Load'. There is no clash, and class 3 (bitfield 20) is added to the labels. Add a few more substances (e.g. substances 1400, 1515, 3250), 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'. You may also see return code 98 - this substance is only allowed in tanks (e.g. substance 3250).

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, and 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  
-2 = Can't get enough memory (unlikely, except in 16-bit versions)  
-28 = Unactivated copy.  
Other negative values = various copy protection failures.  
  
+ve = OK (the file handle of subs.ind)

With an unactivated copy and other protection failures, the system works in 'demo mode'. This only accepts UN numbers ending in 0 or 5, and some of the fields of the dangerous goods list are blank (struct tups, elements pp, ibci, ibcp, tankun, tankimo, tankp, flash and explim; specprov shows only the first provision)

## **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 be 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  
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 phrase codes from column 16 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 chapter 7.1 of the 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
    char prim[5]; // primary class
    char secn[6]; // secondary class (subrisk)
    char tert[4]; // tertiary class
    char mp;      // Marine pollutant, null, Y, S (=severe, PP) or C
                // (=conditional, bullet)
    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[28]; // special provisions
    char lq[10];   // limited quantity
    char pi[16];   // packing instructions and provisions
    char pp[26];
    char ibci[11]; // IBC instructions and provisions
    char ibcp[15];
    char tankun[7]; // tank instructions, UN, IMO and provisions
    char tankimo[9];
    char tankp[25];
    char flash[26]; // flashpoint
    char explim[28]; // explosive limits
    char scat[3]; // stowage category, A to E or 01 to 15
    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.

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 provisions 900 (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  
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  
1 = no such comment.

Hint: If the user enters a UN number, call TupsSub for that UN number and suffix zero. For an 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.

## 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 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  
TUPS\_CARGO for a cargo sailing  
and may be or'ed with  
TUPS\_NOCHECK to suppress validation as each substance is added.

**type** TUPS\_OPEN  
TUPS\_CLOSED  
TUPS\_TANK according to the unit type

**clashlist** array of shorts which becomes filled with triplets describing each clash  
a) sequence number of first substance counting from 1  
b) sequence number of second substance  
c) degree of clash - see values 1 to 10 for TupsAddSub  
The list is terminated by a zero value. With a single load (a) < (b).

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  
98 if not valid in a tank / must be in a tank, when this unit isn't / is a tank.  
97 if substance prohibited (entirely, or on this passenger sailing).  
96 if this explosive is not allowed in this open unit.  
88 if failed to expand working space (16-bit versions are limited to about 440 substances in a single load).  
1 to 10 highest clash value if this substance clashes with others already in the load.  
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 also use numbers 5 to 10 in relation to explosives:

- 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 quantities (see IMDG chapter 3.4) 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 quantity substances.

### **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 bitfields 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.

## 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 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](#)

## The Java routines

Subsdemo.java is a similar demonstration program written in Java. The Java routines it calls are thread-safe, so they could be used by a multi-threaded server application. There is one static class 'Tups' which controls the data file and is used for phrase lookups. An instance of a class 'DGSubs' is used for substance lookups, and one of class 'DGLoad' is used for load validation.

### class Tups

#### **public static int start(String filename)**

This should be the first routine called. On an initial call it opens the data file and does various initialisations. Subsequent calls to the routine (e.g. by other threads in a multi-threaded server application) are harmless and have no effect.

**filename**            the filename, "subs.ind", with any necessary directory specification.

**return value**        0 = OK  
                      1 = error, probably unable to read the file

#### **public static synchronized void unload()**

This closes the data file, for instance if it needs to be replaced by a revised version without restarting the application or taking down a server. No other routines should be used until another call to 'start' has been made to open another file. If the data file is changed and a user has existing instances of class DGSubs, the next routine called should be TupsSub (to look up a new substance) rather than one which gets information about the 'current' substance. Similarly, if there are existing instances of DGLoad, the next routine should be TupsNewLoad to start a new load rather than continue working with the substances in an existing load.

#### **public static int TupsCom(short code, StringBuffer phrase)**

Obtains the comment text related to a numeric comment code. The text may include linefeed (ascii 10) characters.

**code**                the comment number  
**phrase**              the phrase

**return value**        0 = OK  
                      1 = no such comment

#### **public static int TupsPhrase(String phrasecode, StringBuffer phrase)**

Obtains the text related to a named comment code. The text may include linefeed (ascii 10) characters. The codes may be special provisions (2 or 3 digits, as in IMDG chapter 3.3), packing instructions (Pxxx or LPxx), IBC instructions (IBCxx), tank instructions (Txx), or packing provisions (PPxx, Bx or TPxx), as in IMDG chapters 4.1 and 4.2.

**phrasecode**        the phrase name  
**phrase**              the phrase

**return value**        0 = OK  
                      1 = no such phrase

### class DGSubs

Public member variables

```

short un;           // UN number
short suf;         // suffix - see description in TupsSub
StringBuffer prim; // primary class
StringBuffer secn; // secondary class
StringBuffer tert; // tertiary class
char mp;          // null, Y, S or C for No, Yes (P),
                // Severe (PP) or possible (bullet)
byte ulineems;    // whether the EmS is underlined
                // 1=fire, 2=spillage, 3=both
StringBuffer ems; // EmS
StringBuffer specprov; // special provisions
StringBuffer lq;  // limited quantity
StringBuffer pi;  // packing instructions and provisions
StringBuffer pp;
StringBuffer ibci; // IBC instructions and provisions
StringBuffer ibcp;
StringBuffer tankun; // tank UN, IMO and provisions
StringBuffer tankimo;
StringBuffer tankp;
StringBuffer flash // flashpoint
StringBuffer explim; // explosive limits
StringBuffer scat; // stowage category
StringBuffer pg;    // IMDG packing group
char state;        // S,L,G,E,A for solid, liquid, gas,
                // explosive substance or article
int flags          // flags indicating what additional data
                // is required from the user

```

The value returned in 'flags' is the combination of:-

```

1 technical name required
2 possible marine pollutant
4 flashpoint required
8 control and emergency temperatures
16 Radionucleide, category, activity and transport index, for class 7.
32 possible limited quantity
64 net explosive content
128 warn user about special provisions, e.g. 900
256 class or subrisk of 2.1

```

### **public int TupsSub(int unno, int suffix)**

Finds a substance on file.

<b>unno</b>	UN number
<b>suffix</b>	suffix. Where a substance has no variants, it will be numbered 0. Those with variants are numbered 1 upwards. A search for suffix 0 or 1 will equally find either of 0 and 1. Alternatively, suffix may be -1 or -2, to find respectively the next or previous UN number (and suffix 0 or 1) to the one specified.
<b>return value</b>	0 = OK. The member variables are now set up. 99 = no such substance. The UN number / suffix combination does not exist, or for next / previous UN number, the specified UN number was already at or beyond the end of the range.

**public void TupsName(short wanted, StringBuffer name)**

Retrieves the Proper Shipping Name, qualifying descriptive text and variation description for the current substance, or the 'Properties and Observations' from the Dangerous Goods List.

**wanted** = Tups.NAME (3) for the Proper Shipping name.  
= Tups.PROPS (4) for Properties and Observations  
**name** the name or properties are put into this variable

**public void TupsName(short wanted, short codes[])**

Reads a set of stowage comment code numbers. These can be passed to Tups.TupsCom to be converted to phrases.

**wanted** = Tups.STOW (2) for phrases shown on the 'Stowage and Segregation' in the Dangerous Goods List  
= Tups.STOWP (1) for general stowage phrases for this class, packing group, subsidiary risk, etc from IMDG chapter 7.1, for passenger sailings  
= Tups.STOWC (0) as above, but for cargo sailings.  
**codes** array of shorts to receive the code numbers. An array of 20 elements will be sufficient. The list is terminated by a zero value. For wanted values other than 2, the first number will always be one of the following (the text can be obtained from TupsCom).

- 1 Prohibited.
- 2 As approved by the competent authorities of the countries involved in the shipment.
- 3 On deck only.
- 4 On or under deck.
- 22 On or under deck in closed CTU.
- 23 On deck only in closed CTU.
- 24 On deck in closed CTU or under deck.
- 26 On deck in closed CTU or under deck Magazine stowage type 'C'.
- 27 On deck in closed CTU or under deck Magazine stowage type 'A'.

**class DGLoad****public static variable**

**short MAXSUB** default value of 40. The maximum number of substances in a load. To allow the validation of a load with more items, increase the value before calling the DGLoad constructor. There is a memory overhead of 42 \* MAXSUB for each DGLoad object, so do not set it extravagantly large. For segregation between loads, MAXSUB needs to be at least 1 greater than the number of items in the first load.

**public void TupsNewLoad(short mode, short unittype)**

For validation of a single load:-

**mode** DGLoad.PSNGR (4) for a passenger sailing  
DGLoad.CARGO (0) for a cargo sailing  
**type** DGLoad.OPEN (1) or  
DGLoad.CLOSED (2) or  
DGLoad.TANK (3) according to the unit type

For segregation checks between this load and others, see later.

**public int TupsAddSub(int un, int suffix)****public int TupsAddSub(int un, int suffix, int nec)**

Adds a substance to the load, and validates it against the load so far.

<b>un</b>	UN number
<b>suffix</b>	Suffix
<b>nec</b>	(optional) net explosive content in grammes (ignored except for explosives of compatible groups B, C, D, E, G and N)
<b>return value</b>	99 if no such substance 98 if not valid in a tank / must be in a tank, when the unit type is / is not a tank 97 if the substance is prohibited (totally, or on this passenger sailing) 96 if the explosive is not allowed in this open unit. 95 if the total net explosive content exceeds that permitted on this passenger sailing 88 if MAXSUB items are already in the load. 1 to 10 highest clash value if segregation is required from previous items. 0 if no segregation required

With a return value of 88 upwards, the substance is not added to the load.

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

We also use numbers 5 to 10 in relation to explosives.

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.

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

#### **public int TupsEditLoad(int seq, int un, int suffix)**

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

<b>seq</b>	sequence number within the load, numbered from 1 upwards.
<b>un</b>	UN number (or zero to remove an item from the load)
<b>suffix</b>	Suffix
<b>return value</b>	99 to 96 as with TupsAddSub 0 = OK

#### **public int TupsReCalc()**

re-validates a load, when one or several changes have been made with TupsEditLoad.

<b>return value</b>	0 if no clash 1 to 10 highest clash value if a clash.
---------------------	--

#### **public void TupsLabCom(short lab\_com[])**

lab\_com becomes 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, terminated by a zero value. Use TupsCom to convert to phrases.

The bitfields for labels represent (in hex)

- 1 marine pollutant
- 2 explosive (1)
- 4 flammable gas (2.1)
- 8 non-flammable gas (2.2)
- 10 toxic gas (2.3)

20 flammable liquid (3)  
40 flammable solid (4.1)  
80 spontaneous combustible (4.2)  
100 dangerous when wet (4.3)  
200 oxidizing substance (5.1)  
400 organic peroxide (5.2)  
800 toxic (6.1)  
1000 infectious (6.2)  
2000 radioactive (7)  
4000 corrosive (8)  
8000 miscellaneous (9)

Bit '1' in prims indicates a marine pollutant label is definitely required. '1' in secs indicates one may be required, depending on concentration or composition. lab\_com[2] will always be

1 (prohibited),  
2 (as approved by competent authority),  
3 (on-deck only) or  
4 (on or under deck) - the stowage position according to IMDG.

Some other values in later elements of lab\_com 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).

#### **int TupsClashes(short list[], int maxnum)**

This returns details of the individual clashes between items in the load. The list is filled with sets of three numbers being:-

the sequence number of one item in the load  
the sequence number of the item it clashes with  
the degree of clash (1 to 10)

<b>maxnum</b>	the number of clashes stored in 'list', so 'list' can be dimensioned to 3*maxnum+1.
<b>return value</b>	the number of clashes stored in 'list'.

The list ends with a zero value for the first sequence number. For validation of a single load the first sequence number will always be smaller than the second.

#### **Compatibility between loads.**

Having built one load with TupsNewLoad and TupsAddSub, TupsNewLoad can be called again with a mode of DGLoad.SECOND (64). Tups\_AddSub is then called for each substance in the second load, and will check them for segregation from all the items in the first load. Note:-

- further loads can be checked against the first with more TupsNewLoad / TUPS\_SECOND calls.
- TupsEditLoad, TupsReCalc and TupsLabCom are not appropriate with a TUPS\_SECOND load.
- in TupsClashes, the first sequence number refers to an item within the first load, the second to an item within the second load.

If one 'first load' is being checked against all the other loads on the ship as 'second loads', a return value of 95 from TupsAddSub will indicate that the total explosive content of a passenger ship (IMDG 7.1.7.5) has been exceeded.

**Contacting Exis Technologies Limited**

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

email: [support@existec.com](mailto: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 2050

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

ret = TupsSub(2050,0);
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(2050, 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 944

```

ret = TupsPhrase("944",buf);
if (ret == 0) printf(" %s\n", buf);

Dim Comment As String
Comment = String$(5120, 0)
ret = TupsPhrase("944", Comment)
If (ret = 0) Then
    results.Text = "Phrase 944 is " + sTrim(Comment)
End If

```

#### d) Show variations of UN 2810

```

un = 2810;
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.)