CHAPTER 4 Using the API: Application Program Interface

API Data Types

Real numbers passed to WIT are of the type `float`. Integer numbers are typically of the type `int`. The function definition provides the correct type. Many WIT functions pass or return vectors. These vectors always have length equal to nPeriods or the vector length is a parameter. Data types specific to WIT are defined in the file `wit.h`. They are:

- `WitRun`

  A structure which defines the WIT problem. A pointer to this structure is obtained by using the function `witNewRun` and is the first parameter of each WIT API function.
- `witBoolean`

  The constants `WitTRUE` and `WitFALSE` are parameters to several WIT functions having the type `witBoolean`. `WitTRUE` and `WitFALSE` are defined in the file `wit.h`.
- `witAttr`

  This type is used to define several constants passed to WIT functions. These constants are defined in `wit.h`.
- `witReturnCode`

  WIT function return codes are of the type `witReturnCode` and are either:

  ```
  WitINFORMATIONAL_RC
  WitWARNING_RC
  WitSEVERE_RC, or
  WitFATAL_RC.
  ```

  The return code represents the highest severity message condition which occurred during the function invocation. The definitions `WitINFORMATIONAL_RC`, `WitWARNING_RC`, `WitSEVERE_RC`, and `WitFATAL_RC` are in the file `wit.h`.

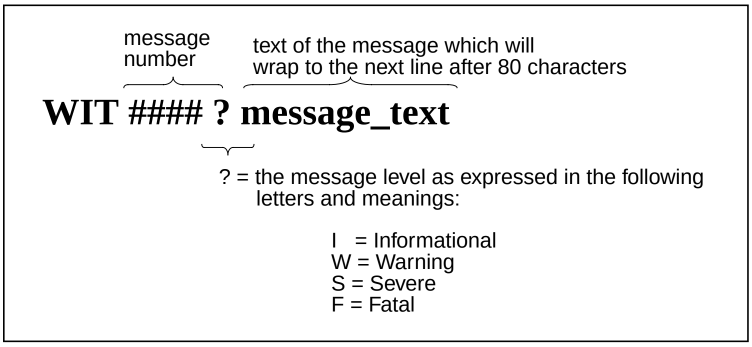
  Since the relation

```
WitINFORMATIONAL_RC< WitWARNING_RC< WitSEVERE_RC< WitFATAL_RC
```

is true, an application program can check to see if the return code is greater or equal to `WitSEVERE_RC` to check for a severe or fatal return code.

API Message Attributes

WIT messages are of the form:


message number
text of the message which will wrap to the next line after 80 characters
WIT #### ? message_text
? = the message level as expressed in the following letters and meanings:
I = Informational
W = Warning
S = Severe
F = Fatal

Message levels including the following detailed meanings:

- **I** is informational. These messages provide information on what WIT is doing.
- **W** is warning. These messages indicate that WIT has recognized a situation which may not be the user's intention.
- **S** is severe. Severe messages indicate that something has occurred that prevents WIT from continuing. The default action has WIT terminating the run of the application immediately after issuing a severe message.

  However, severe messages that result from setting part, demand, BOM entry, or substitute BOM entry attributes to out-of-range values do not cause WIT to terminate running the application. Under these circumstances the application is terminated when `witPreprocess` is invoked. This allows the application to identify all out-of-range values with a single run.

  The default action of terminating the application after issuing a severe message can be altered by changing the `mesgStopRunning` attribute. If the `mesgStopRunning` attribute is false, then the WIT routine issuing the message immediately returns to the application with a return code of `WitSEVERE_RC`.
- **F** is fatal. Fatal messages indicate that WIT has recognized a condition which probably represents an internal programming error.

mesgFileAccessMode

`char *`

Default value: "a"

This is the file access mode WIT uses when opening files with the C function `fopen`. For more information, see the ANSI C `fopen` function.

mesgFile

FILE *

This file is used to write WIT messages.

mesgFileName

char *

Default value: WitSTDOUT

Name of file where WIT writes messages. The value WitSTDOUT can be used to indicate that messages are to be written to stdout.

The acceptable values depends on the platform. Since the C function fopen is used to open the file, see fopen documentation for the platform being used.

mesgPrintNumber

witBoolean

Default value: WitTRUE

Associated with individual messages. If set to WitTRUE, the message is displayed the with WIT####? message number.

mesgStopRunning

witBoolean

Default value: WitTRUE

This attribute is associated with individual messages. It can be set and retrieved for any message, but it only applies if the message is of level “severe” or “fatal”. If set to WitTRUE, (which is the default for these messages), WIT will cause the application program to stop running after issuing the message. If the application program is to regain control after a severe or fatal message is issued, then this attribute must be set to WitFALSE. After WIT has issued a severe or fatal message, WIT’s internal data structures are no longer in a valid state, and no further WIT functions should be called (even with a different WitRun).

mesgTimesPrint

0 ≤ Integer ≤ UCHAR_MAX

Default value depends on the message.

This attribute is associated with individual messages. It indicates how many times a message is printed. UCHAR_MAX defined in the file limits.h indicates that the message will always be displayed. Zero indicates that the message will never be displayed.

API Bound Set Definition

The term bound set describes three ordered float vectors which describe a boundary condition. When using the API each vector has length equal to the number of time periods. The vectors are:

- Hard lower bounds
- Soft lower bounds
- Hard upper bounds

When a bound set is passed to a WIT function, these 3 ordered vectors are passed. If one of the vectors is `NULL` then that vector is unchanged. For more information see “Bound Set Attributes” on page 128.

The State of a WitRun

At any time, a given WitRun is considered to be in some “state”. The state of a WitRun determines which of the WitRun’s internal data structures are currently valid. It is determined by the sequence of API calls that have been previously made for the WitRun, and in some cases, it influences the effect that the next API call will have. The state of a WitRun is characterized by the following two boolean attributes:

accelerated

True if and only if an optimizing implosion has been performed while accAfterOptImp was True and all subsequent actions were compatible with an accelerated state. If this attribute is True, then the WitRun is considered to be in an accelerated state. If it is False, the Witrun is considered to be in an unaccelerated state. When a WitRun is in an accelerated state, the data structures that are necessary in order to perform an accelerated optimizing implosion exist and are in a valid state. When an application calls `witOptImplode`, the resulting optimizing implosion will be an accelerated optimizing implosion if and only if the WitRun is in an accelerated state and the optInitMethod attribute = “accelerated”. The only way to put a WitRun into an accelerated state is to call `witOptImplode` when the accAfterOptImp attribute is True. Various functions will put the WitRun into an unaccelerated state; see Table 4 on page 135.

postprocessed

True if and only if postprocessing has been performed and no subsequent action has altered the input data or the production and shipment schedules. If this attribute is True, then the WitRun is considered to be in a postprocessed state. If it is False, the WitRun is considered to be in an unpostprocessed state. Postprocessing is automatically performed at the end of the implosion routines. It computes the following data attributes:

- feasible
- stockVol
- scrapVol
- excessVol

When a WitRun is in a postprocessed state, these attributes are in a valid state in the sense that they correspond to the current input data and current production and shipment schedules. In particular, it is an error to call `witGetFocusShortageVol` or `witGetPartFocusShortageVol` when the WitRun is in an unpostprocessed state, because these attributes must be valid in order for WIT to compute a focussed shortage schedule; see “Focussed Shortage Schedule” on page 39. The following functions put the WitRun into a postprocessed state:

- `witHeurImplode`
- `witOptImplode`
- `witPostprocess`

Any function that changes the definition of the implosion problem or changes the production and shipment schedules will put the WitRun into an unpostprocessed state; see Chapter 5, “API Function Library”.

General Comments about State Attributes

The state attributes, accelerated, and postprocessed, are considered to be global data attributes of WIT (See “Global (WIT Problem) Attributes” on page 78.) and their values can be obtained by the appropriate API “get” routine. (See “witGetAttribute” on page 141.) Also, when the value of any state attribute changes, a message is displayed.

To determine which attributes can be changed while preserving an accelerated state, see Table 4 on page 135. The attributes corresponding to a “No” in the right-hand cannot be changed while preserving an accelerated state. For example, if you call the function `witSetOperationYieid` on a WitRun in an accelerated state, the WitRun will be put into an unaccelerated state. But if you call the function `witSetPartSupplyVol` on a WitRun in an accelerated state, the WitRun will remain in an accelerated state.

TABLE 4 **Which Attributes Can Be Changed While Preserving an Accelerated State**

Attribute (Input attributes only)	Object Type	Can this attribute be changed while preserving an accelerated state?
accAfterSoftLB	Global	No
accAfterOptImp	Global	Setting it to True preserves an accelerated state. Setting it to False puts the WitRun in an unaccelerated state.

TABLE 4 Which Attributes Can Be Changed While Preserving an Accelerated State

Attribute (Input attributes only)	Object Type	Can this attribute be changed while preserving an accelerated state?
appData	Any	yes
autoPriority	Global	Yes
buildAheadUB	Part	Yes
buildAsap	Part	Yes
buildNstn	Part	Yes
capCost	Global	Yes
compPrices	Global	Yes
computeCriticalList	Global	Yes
consRate	BOM Entry Substitute BOM Entry	No
cumShipBounds	Demand	See “Bound Sets and Accelerated Optimizing Implosion” on page 50.
demandVol	Demand	Yes
earliestPeriod	BOM Entry Substitute BOM Entry BOP Entry	No
equitability	Global	Yes
execBounds	Operation	See “Bound Sets and Accelerated Optimizing Implosion” on page 50
execEmptyBom	Global	No
execPenalty	Operation BOM Entry Substitute BOM Entry	Yes
execVol	Operation	Yes
expAllowed	Substitute BOM Entry	Yes
expAllowed	BOP Entry	No
expAversion	BOP Entry	No
expCutoff	Global	No
expNetAversion	Substitute BOM Entry	Yes
falloutRate	BOM Entry Substitute BOM Entry	No
focusHorizon	Demand	Yes
forcedMultiEq	Global	Yes
grossRev	Demand	Yes
hashTableSize	Global	Yes
highPrecisionWD	Global	Yes
incLotSize	Operation	No
incLotSize2	Operation	No
independentOffsets	Global	No

TABLE 4 Which Attributes Can Be Changed While Preserving an Accelerated State

Attribute (Input attributes only)	Object Type	Can this attribute be changed while preserving an accelerated state?
invCost	Global	Yes
latestPeriodt	BOM Entry Substitute BOM Entry BOP Entry	No
lotSize2Thresh	Operation	No
lotSizeTol	Global	Yes
mandEC	BOM Entry Substitute BOM Entry	No
minLotSize	Operation	No
minLotSize2	Operation	No
mrpNetAllowed	Substitute BOM Entry	Yes
multiExec	Global	No
multiRoute	Global	No
netAllowed	Substitute BOM Entry	Yes
nPeriods	Global	No
objChoice	Global	No
obj1CumShipReward	Demand	Yes
obj1ExecCost	Operation	Yes
obj1ScrapCost	Part	Yes
obj1ShipReward	Demand	Yes
obj1StockCost	Part	Yes
obj1SubCost	Substitute BOM Entry	Yes
obj2AuxCost	Operation	Yes
obj2SubPenalty	Substitute BOM Entry	Yes
obj2Winv	Global	Yes
obj2Wrev	Global	Yes
obj2Wserv	Global	Yes
obj2Wsub	Global	Yes
offset	BOM Entry BOP Entry Substitute BOM Entry	No
optWithLotSizes	Global	No
oslMesgFileName	Global	Yes
perfPegging	Global	Yes
penExec	Global	Yes
periodsPerYear	Global	Yes
pipSeqFromHeur	Global	Yes
pipShare	BOP Entry	Yes

TABLE 4 Which Attributes Can Be Changed While Preserving an Accelerated State

Attribute (Input attributes only)	Object Type	Can this attribute be changed while preserving an accelerated state?
prefHighStockSLBs	Global	Yes
priority	Demand	Yes
productRate	BOP Entry	No
propRouting	Part BOM Entry	Yes
respectStockSLBs	Global	Yes
roundReqVols	Global	Yes
routingShare	BOM Entry Substitute BOM Entry BOP Entry	Yes
selForDel	Any	Yes
selSplit	Global	Yes
shipLateUB	Demand	Yes
shipVol	Demand	Yes
singleSource	Part	Yes
singleSource	BOM Entry	Yes
skipFailures	Global	Yes
stockBounds	Part	See “Bound Sets and Accelerated Optimizing Implosion” on page 50
stockReallocation	Global	Yes
subVol	Substitute BOM Entry	Yes
supplyVol	Part	Yes
tieBreakPropRt	Global	Yes
title	Global	Yes
truncOffsets	Global	No
twoLevelLotSizes	Operation	No
twoWayMultiExec	Global	No
unitCost	Part	Yes
useFocusHorizons	Global	Yes
userHeurStart	Global	Yes
wbounds	Global	Yes
yieldRate	Operation	No