

# ALERTER for DEVELOPERS: User Manual

SHUTTLE ENTERPRISE SERVER

## Introduction

ALERTER is a system that is able to inspect your Data, test for Conditions, and act thereon. ALERTER is also a Monitor or Watchdog. It can be told to Alert the necessary people when something goes wrong, when something goes right, or it can be programmed simply to perform Tasks or remind you of things you wish to be reminded of.

Simple words, but they don't convey the power of Alerter. Imagine being away from base and getting an e-mail from Alerter that the Budget on a certain Account is overspent (or about a Birthday you dare not forget!). Now imagine being advised that somebody didn't finish a certain Project deadline on time, that the next responsible Person in the hierarchy has done nothing about it either, and that there are imminent penalties. Now you get alerted, and if you don't do anything about it either, then Alerter may tell your boss. Or imagine getting your daily Progress Report automatically by e-mail, whether you're at the office or away.

These are only examples of what Alerter can do. In fact, it is capable of inspecting any programmable condition in your system, and can even act as a Fraud Inspector.

## Purpose of the Manual

The purpose of this User Manual is to introduce you to the use of ALERTER DEVELOPER know how.

The Alerter User Manuals are presented in a set of 3 :-

### Identification

|    |   |
|----|---|
| #1 | Alerter for Users                         |
| #2 | Alerter Operations                        |
| #3 | Alerter for Developers (Current Document) |

### Target Users

|                  |
|------------------|
| ALL Users        |
| Operations Staff |
| Developers       |

## Before you start ...

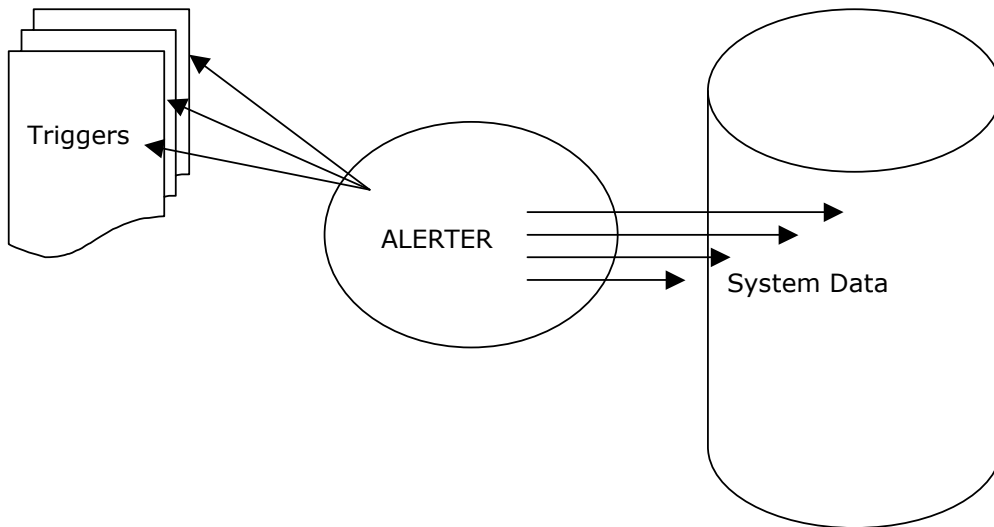
Before you start with this User Manual, be sure to have studied the 1<sup>st</sup> 2 Manuals in this set of 3, e.g. the Alerter User Manual for Users, and the Alerter User Manual for Operations.

You should have already done the SHUTTLE Developer Course (self-study or other) before attempting the steps in this Manual.

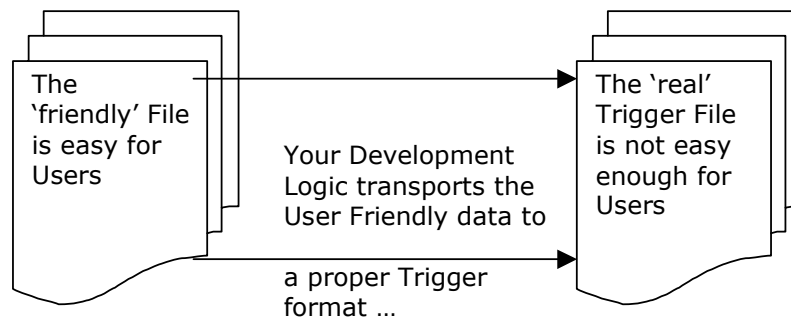
## What is it? Alerter Developer

Since you have now already studied the other 2 Alerter Manuals, you know what Alerter is, and what it is for. The crucial remaining issue is to be able to provide Users with new Alerter Trigger Types. However logical Alerter may be, it uses advanced and sophisticated concepts that are not necessarily difficult for Developers to understand, but that are definitely beyond the grasp of the average system User. For this reason, it has been designed in a way to provide you with tools to create Trigger Types that will be easy for Users to understand.

Alerter is so powerful, and there are so many things you can achieve with it, that we cannot possibly provide you with everything ready-made. You are certainly going to think up other things you want to do with Alerter, and that's why you need to understand that you have in Alerter everything you need to do so.



We can visually represent Alerter's task as looking at the Triggers that are present, and applying the Trigger logic to the system data, then, based on Criteria and Conditions, to decide what to do. (Even though this is an over simplification!)



When we look at the 'real' Trigger File, on which Alerter operates, there are 4 generic Trigger Types. You, as a Developer, need to understand each of these. Once you do, you may then construct any number of User Friendly input Screens for Users to define on the 'triggerfriendly' File, and the logical Routines that relay them to the real Trigger File, which will only accept the 4 generic types.

The 4 Generic Trigger Types are –

- 1) Alert BEFORE Date on Conditions
- 2) Alert AFTER Date on Conditions
- 3) Alert on Value
- 4) Alert by Subroutine Logic

Any Alert Trigger that cannot accomplish it's objective with any of the 1<sup>st</sup> 3 options, certainly can with the last.

Among the Alerter Operations Options, there is a function to 'Update Trigger Types'. The generic system types are numbered 996 – 999, and you should never attempt to change these, as they are the fundamental types used by all other types. In the number range 001 – 995, you may invent any types required for your Application. Below we have a look at the definition of a Trigger Type.

**DEFINE ALERT TRIGGER TYPES (UPDATE)**

A Trigger Type has a Number and a Description

Trigger Type Key: 001

Description: Daily Diary Alert

Process for New: triggerfriendly-001 Alert Trigger 001

Process for Amend: triggerfriendly-001 Alert Trigger 001

Auto Remove (Once off Triggers)

Developers Only ?

May Use ?

Help Description: This Alert Trigger will warn on a Daily basis if there are any Entries in your Diary that are 'older' than today, i.e. to have been attended to before today. The Alert is not per entry, but rather per Diary, i.e. will alert once for 1 or more Entries older than Today.

Hint: Diary Entries may be moved to future Dates as required. The Alert is intended to Alert you to the fact (if this is the case) that you have 1 or more entries in your Diary which you

Callouts: A points to the process fields, B points to the checkboxes, C points to the help description.

A: Each Trigger Type has a defined Process to call for defining a New Trigger of this Type, and for Amending an Existing Trigger of this Type. You may define your Process in such a way that the same Process may be used for both modes. In each case, the Process will be a Screen based on the 'triggerfriendly' File.

B: **Auto Remove** is used only when the Trigger is of the once-off Type, i.e. as soon as the Alert is Signed Off, the system should automatically remove the Trigger (not the Type!). Hint: This is unusual, and not often used. Even if a Trigger causes a once-off Alert Message, the Trigger itself usually stays to continue doing it's work. A typical ONCE-OFF Trigger Type would be something that acts as a once-off reminder for a User, not necessarily data related. If the system does not automatically remove such a Trigger, it would stay in the system indefinitely, without ever doing any work again, wherefore such a Trigger should be CHECKED for Automatic Removal at Sign Off.

**Developers Only** is checked for the generic system Trigger Types so that they do not appear among User choices for Trigger definition.

**May Use** is checked to allow new Triggers of this Type to be defined, and unchecked when Users are not allowed to define new Triggers of the Type, even though existing Triggers of the Type will remain unaffected.

C: The Help Description is very important, since this Text is offered to the User as a final step in deciding whether the correct Trigger Type is being selected for definition (as we shall see below when looking at Trigger Definitions).

We will now have a look at ALERT BEFORE DATE

**DEFINE NEW TRIGGER ?**

|              |                                   |
|--------------|-----------------------------------|
| Trigger Type | 996                               |
| Description  | Alert BEFORE Date on Condition(s) |

Help

This is a Generic Trigger type used mostly by Developers.  
It is used to Alert BEFORE a derived DATE, based on the specified condition(s).

Whenever a new Trigger is defined, the system offers a Description of the Type to be defined. It is important to provide Users with a Description that is useful in deciding whether it is in fact the correct Trigger Type that is required.

Once the correct Type is accepted, the Trigger Definition Screen is offered. In the example below, we are dealing with one of the generic types, i.e. 'Alert Before Date', and once we have looked at the basic definition of such a Trigger Definition, we will discuss the concepts encountered on this Screen.

**ALERT TRIGGER TYPE 996 (UPDATE)**

1 Main | 2 Notes

Trigger Key: 1307037902

Type: 996 Alert BEFORE Date on Condition(s)

Description: [Empty]

File Name: [Empty]

Opt Null Field Check: [Empty]

Calculate Date From: [Empty]

Warn Days Before: [Empty]

Notify Whom: [Empty]

Alert Message: [Empty]

Trigger Mode: PROCESS TRIGGER

Inspection Frequency: CHANGE

Data Selector: \*\*\* Item Key

Message Frequency: ONCE

Sign Off Condition: Either Auto or Manual

Postponement Rule: NO POSTPONEMENT

Status: ACTIVE

Ownership: Self

Owner: Data Manager

Buttons: Null Field, Calc Date From, Warn Before, Notify, Data Selector, Trigger Mode, Remove Trigger, Debug / Test, Quit, Update

**Type:** The Type will always be pre-formatted for the User. The generic types are in the range 996 – 999, and you may 'invent' any types in the range 001 – 995.

**Description:** Although each Trigger that is defined will always have a Description, so the User may differentiate it from other Triggers in his / her Queue, you may decide whether your User has to choose a Unique Description, or whether your Process will provide it.

**FileName:** A Trigger always operates on a primary FileName. When the Trigger is not Data Related in any way, you can use a 'dummy' FileName, as long as it exists as a valid Filename in the current DataMart.

Not all the Fields encountered in the generic example above will or should be required from the User. The point is exactly that although each Trigger must always fulfill all the requirements, you may provide (through your Process), many of these Fields, and let the User only capture what is essential, or may be different from one User Trigger to the next. To understand this more clearly, let us look at an example of a 'friendly' Trigger before continuing with the discussion of the Fields in the example above.

**ALERT TRIGGER 001 (UPDATE)**

Trigger Key

Type

Description

Owner

Status

Above we show an example of a 'friendly' Trigger, in the format it is offered to the User for Definition. This particular type will Alert the User whenever there are Diary Entries in his / her Queue older than TODAY. Only a single field has to be entered / chosen by the User, e.g. whether the Trigger is to be ACTIVE or INACTIVE at this stage. When the User chooses UPDATE, there is a subroutine call behind the Button that will resolve all other Field Requirements necessary for a proper Trigger definition, and that is the Developer's task to fulfill.

Since the Developer must decide, for each Trigger Type, which Fields the User will define, and in which way, and which Fields will be provided by the system, it is necessary for the Developer to have a good understanding of all the Fields on the generic example shown higher up, and with which we may now continue. The next field on that screen is 'Opt Null Field Check'.

**Opt Null Field Check:** This Field is optional, but it is a useful option; If you state a DataName on the primary File here, the system will check to ensure that the DataName = NULL for a Record being evaluated, and if not, it does not qualify for an Alert, and will not proceed with further evaluation on the Record;

**Calculate Date From:** This Field is used to indicate which Data to inspect to find the Date before which the system should warn or raise an alert, and is specified in the following format :

*ho dataname dataname - 10*  
 (where 'ho' is highest of dates found in listed datanames less 10 days)

*lo dataname dataname dataname + 5*  
 (where 'lo' is lowest of dates found in listed dataname(s) plus 5 days)

*'ho' and 'lo' are interchangeable; 1 or more datanames may be specified; datanames can be single or multi valued; '-' and '+' are interchangeable; no of days must be an integer*

*in the range 0 - 9999 ...*

*Note: no of days will normally be '0', but the feature is provided to allow more flexibility in adjusting dates, since Alerter encourages interpretation, while the Dates themselves are 'real' Application Data ...*

**Warn Days Before:** This Value works in conjunction with the 'Calculate Date from Data' value. That result determines a Date from the Application Data. The purpose of this Trigger though, is to Alert BEFORE that Date is reached, and you should indicate how many Days BEFORE that Date the Alert should be raised.

*State as negative, e.g. no of days before System Date reaches the 'Calculated Date', to be warned, e.g. warn 5 days before: -5*

*or to warn ON the Date, state: 0*

**Notify Whom:** Tells the system who should be Alerted.

*Please specify who should be notified when an Alert message is generated : (one of the following options)*

*User List*

*-----*

*u tt dm kl*

*(where 'tt' 'dm' and 'kl' are all Users specified on the SHUTTLE users file)*

*Subroutine Call*

*-----*

*sub subname*

*(where 'subname' will be called with the following argument list: filename,recordkey,trigkey,users and 'subname' should determine multi-valued list of 'users' and return such)*

*Derive from Record*

*-----*

*d dataname dataname*

*(where datanames enumerated will be interrogated in the relevant data record relating to the Trigger entry for*

**Alert Message:** Message to generate - Text.

Only standard text is used for the Base message, since the User will always be informed of details on the Record key and other Trigger detail as well.

*In recognition of situations where you expressly need to customise a message at Runtime, one more option is provided:*

*specify : %subname%*

*and if nothing else but the above is found on the Message field, Alerter will call the Subroutine name enclosed within '%', with the following argument list :*

*FileName,ItemKey,TriggerId,CustomMessage*

where your subroutine will return the Custom Message as a MultiValued Text List ...

*Hint(1): The 1st Line on the Message is used for the Message/Mail Title, and is repeated again in the body thereof.*

*Hint(2): If your Trigger will submit a Task or a Diary Entry instead of raising an Alert Message, you may use this same facility (calling a Subroutine)in which to raise the Task or Diary Entry, calling either of :-*

*diary.submit  
task.submit*

*from your Subroutine. Detail on how to use these Subroutines appear with How\_Do\_I. Search on: devtips diary OR devtips task*

### Trigger Mode:

We discern 2 Trigger Modes -

#### a) Process Trigger

A Process Trigger is used to evaluate a Data Record when it is submitted by a User from a Screen. Some typical examples could be :-

- i) A Personnel Record is updated, and dependent on defined conditions, this may result in an Alert to perform another action, or may result in another Task being raised, e.g. to follow through a certain action for the Employee;
- ii) A new Job Account is opened, which may trigger another Task or Alert to attend to some particular aspect;
- iii) A Project Master Record is amended to reflect a change in some Target Date, which in turn may result in some User(s) being alerted to attend to certain aspects timeously;

#### b) Monitor Trigger

Monitor Triggers are used for evaluating Data based on Data Selectors, i.e. regular inspection of certain Selections from defined Data Sources. This method is the only way to Trigger Alerts from Data that may be updated or changed programmatically, or as a result of other Data changes, i.e. Data that cannot be intercepted because of a User committing a Data Screen. Some typical examples could be -

- i) A Manager specifies a range of Ledger Accounts for which to be alerted if any of these are overspent, i.e. over the Budget; Since the evaluation of whether an Account is overspent depends on a relationship between data in a number of Files, it cannot be done by a Process Trigger, because if no change is done to the Ledger Master, no Process Triggering will take place. Instead, the Balance and/or Budget for each Account is dependent on other Processes, some of which are programmatic, and has nothing to do with a User committing a related Data Screen;
- ii) A Database Administrator specifies a Monitor Trigger to advise him / her when a certain FileName reaches a specified Record Count. Although a Process Trigger could be used to Count the

Records in the File each time a new Data Record is submitted (if indeed it is not done programmatically), that would constitute undue resource strain on the system - better to use a regular Monitor Inspection to see if the FileName has reached the specified Record Count;

Hint (1) Monitor Mode: A Monitor Trigger is a 'compound' Trigger which may do work on many Data Records in one go. For each DataKey returned by the Data Selector, the entire Trigger Evaluation is done. Therefore, it is not unusual for a Monitor Trigger to raise multiple Alerts simultaneously, i.e. for each DataKey selected where the conditions for an Alert are met;

Hint (2): This Field offers a Lookup, and can also be changed with the 'Change Mode' button.

Further help on the difference between MONITOR and PROCESS:

a) You have a File with many thousands of Records. Whenever one of these Records are updated from a Screen, you want the Alerter to check whether certain conditions are present. You would use a PROCESS Trigger. If you used a MONITOR Trigger, it would have to search the entire File each time.

b) You have another File on which you wish to check a selection of Records each day (NOT the entire File), which Records are selected according to certain rules applied by your choice of Data Selector. For this you would choose a MONITOR Trigger, because a Process Trigger will only select Inspections on Records updated by a User.

**Inspection Frequency:** CHANGE is the only setting allowed for PROCESS Alert Triggers, and this setting may not be used with MONITOR Alert Triggers.

A DAILY inspection on Monitor Triggers will usually suffice, since a Monitor Trigger always uses a Data Selector to evaluate certain Data Sets. However, for critical Monitor Triggers where Alerts need to be evaluated more frequently, you may choose CONTINUOUSLY, in which case the Monitor Trigger will be executed as often as the Alerter Monitor is working in the DataMart, possibly hourly.

Hint (1): The Alerter Monitor has it's own settings as to how much work it should do during the course of a Day, i.e. during which hours it may perform it's work.

Hint (2): Inspection Frequency should not be confused with Alert Frequency. Even if a Monitor Trigger is evaluated daily, once it creates Alerts, these are monitored continuously for possible Alert Messages.

**Data Selector:** A Data Selector is MANDATORY with a MONITOR type Trigger. For PROCESS triggers, the system will insert "\*\*\* Item Key", i.e. only the Data Key that triggers the Alert inspection constitutes the Data Selection.

You may specify a purpose-specific Data Selector for the Trigger, i.e. to select ONLY the relevant Data, but whether the Data Selector calls

other Data Selectors or not, there may be NO runtime prompts.

Hint: RunTime Prompts are the "?" convention used in Selection Criteria to pause for User Response, clearly not compatible with Trigger Processing.

Hint(2): The DataSelector will often use the same DataSource as specified at the Trigger 'Data Source' prompt, but this is not necessarily so.

Developer Tip: If your Data Selection need is to select the UserCode of the Owner of the Trigger being processed, use the system Data Selector "trifriend001d" ;

**Message Frequency:** When an Alert Trigger is raised, this may result in

- a) an Alert Message
- b) a Task being submitted
- c) a Diary entry being submitted
- d) a Programmatic Task being executed

If an Alert Message is being raised, you may specify how often the targeted Users will be notified, i.e. 'once' or whether they should be 'nagged' until the condition that raises the Alert is resolved, i.e. by sending the Alert Message repeatedly at specified intervals.

*Please specify message frequency, as per one of the following examples :-*

*Once Only*

-----

*once*

*Until Signoff with hourly intervals of (always between stated message timeslots*

-----

*h 3*

*(every 3 hours); value must be integer in range 1 - 24;*

*Hint: For 'daily', specify 'h 24'*

*For Triggers that do not raise Alert Messages, you may choose 'NEVER'. This may result in an Alert Entry being raised, which remains in the User Queue, and while not raising a Message or necessarily any other action, it is there when the User views his / her Alerts, or at least until it is Signed Off according to the SignOff Rule.*

*Hint: For actions other than an Alert Message, i.e. Task, Diary Entry, etc. you will also choose NEVER for this Rule. If you combine this setting with SignOff = IMMEDIATE, the Alerter Queue entry will be removed without ever raising an Alert Message (although it may have raised a Diary Entry or Task from your Subroutine(s) that were called. If you combine NEVER without using IMMEDIATE SignOff, then the Alert Entry simply remains present until SignOff conditions have been met, i.e. being inspected daily to ascertain this.*

*A note about NAGGING: Nagging is the practice of Alerting a User at specified intervals until the Condition has been complied with. This is commonly used to ensure that Users will execute specified Tasks timeously, i.e. to meet deadlines and so on. It can go further than that also. You may also specify an additional Trigger that will start informing*

*the User's Superior (Boss?) if the User does not respond to the Alert after some time, and Triggers may in fact be used to escalate this all the way up to the CEO, if necessary. Hint: Your tiered Triggers should simply act on the same conditions, excepting that it introduces some further delay beyond what the 1st tier Trigger does, e.g. if the User does not respond to his / her Alert after 2 days, 3 days, etc. then the next tier Trigger kicks in, informing the next level of User in the hierarchy.*

**Sign Off Condition:** Choose one of the following :-

*AUTOMATIC: (signed off by System only when criteria are met, i.e. the condition that resulted in the Alert no longer exists; User cannot SignOff)*

*MANUAL BY USER: (signed off manually by the User only, i.e. even if the original condition no longer exists, the system will not sign it off;)*

*EITHER: (either of the above allowed, whichever happens first; if signed off by User while original Alert condition still exists, there is a possibility that a new Alert may be raised, depending on Trigger Type. For example, if it is a Process Trigger, another Update on the Record may result in a new Alert being raised, while for Monitor Triggers it may be selected again and found valid to raise a new Alert;)*

*IMMEDIATE (signed off by the System immediately, i.e. after Alert Message or Alert Action has been taken)*

*Note: For all settings other than IMMEDIATE, an outstanding Alert remains in the Alert Queue, until the Alert is signed off per the condition that has been set. In such cases, if the Message Frequency states repeat Messages, then these will be raised at the stated interval until the Alert has been Signed Off. In the case of IMMEDIATE, the Message Repeat Frequency is irrelevant, since the Alert is signed off after the 1st Action (Alert Message or other).*

*Note(2): Careful consideration of the SignOff Mode is necessary with WARN BEFORE. If the system is allowed to SignOff, i.e. with AUTO or EITHER, then SignOff will occur when the BEFORE Date is reached, because it can no longer ALERT BEFORE, and will remove the Alert Entry.*

**Postponement Rule:** The Postponement Rule determines whether the Alerted User(s) may choose not to be 'nagged' by constant Messages until a specified future date.

If the urgency of the situation calls for the User to be 'nagged', then Postponement should not be allowed. Then again, if it is a matter of discretion, i.e. the User may know that the matter needs attention, but may also have other information that gives rise to postponing the situation for some valid reason for a few days, when the Alerter should kick in with Alert Messages again.

The DATE option is used for longer Postponements, when the User will select the DATE until when no further Messages should be raised.

Note(1) When a Postponement Rule of 5 (days) is allowed, the User may continue to postpone for 5 days at a time, i.e. the Postponement allowance may be used indefinitely.

Note(2) Postponement does not mean that the Alert is Signed Off. It stays in the Alerter Queue as an Active Alert that simply is not allowed to raise further Messages until the Postponement Date is reached.

**Status:** When you wish the Trigger not to function for some time, change it to INACTIVE, in which case it will not raise any actions. Once you want to use it again, change it back to ACTIVE.

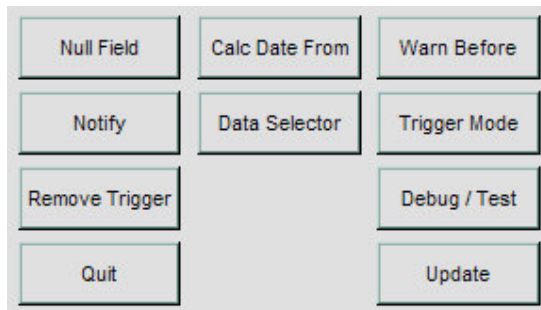
*Hint: Any Alerts that are active in the Queue already are UNaffected by changing this status from ACTIVE to INACTIVE.*

**Ownership:** For Private Alert Triggers, please choose SELF. Such Triggers can only be amended by the Owner, i.e. the User, and are automatically removed by the system when the UserCode Profile in the DataMart is removed.

For Alert Triggers that may be updated by any Developer User, please choose SYSTEM.

Hint: When a Developer User defines a Trigger on behalf of another User, please also choose SYSTEM.

And a brief look at the Function Buttons –



Null Field, Calc Date From, Warn Before, Notify, Data Selector & Trigger Mode will all call Wizards to assist you in the definition of these Fields. Whenever you need to include any of these Fields 'as is' on one of your Trigger Types, you may use the same SHUTTLE calls (as found on the Screen Definition in this example) to provide Functions to assist the User.

REMOVE TRIGGER allows the User to remove a Trigger (not the Type!), and DEBUG / TEST permits testing of the Trigger.

We still need to have a look at the other 3 generic Trigger Types, which use many of the same Fields, albeit with some differences (which we will look at).

The next generic type (997), is ALERT AFTER DATE –

**ALERT TRIGGER TYPE 997 (UPDATE)**

1 Main | 2 Notes

Trigger Key: 1307048861

Type: 997 Alert AFTER Date on Condition(s)

Description: [Empty]

File Name: [Empty]

Opt Null Field Check: [Empty]

Calculate Date From: [Empty]

Warn After Days: [Empty]

Notify Whom: [Empty]

Alert Message: [Empty]

Trigger Mode: PROCESS TRIGGER

Inspection Frequency: CHANGE

Data Selector: \*\*\* Item Key

Message Frequency: ONCE

Sign Off Condition: Either Auto or Manual

Postponement Rule: NO POSTPONEMENT

Status: ACTIVE

Ownership: Self

Owner: Data Manager

Buttons: Null Field, Calc Date From, Warn After, Notify, Data Selector, Trigger Mode, Remove Trigger, Debug / Test, Quit, Update

The only difference between this type and the previous is that this one uses 'warn after'

Type 998 is to ALERT ON VALUE, i.e. derive 2 primary Values, and compare them.

**ALERT TRIGGER TYPE 998 (UPDATE)**

1 Main | 2 Notes

Trigger Key: 1307048997

Type: 998 Alert on VALUE

Description: [Empty]

File Name: [Empty]

Opt Null Field Check: [Empty]

Calc Amt From: [Empty]

Measure: = (Equal To)

Compare With Amt: [Empty]

Notify Whom: [Empty]

Alert Message: [Empty]

Trigger Mode: PROCESS TRIGGER

Inspection Frequency: CHANGE

Data Selector: \*\*\* Item Key

Message Frequency: ONCE

Sign Off Condition: Either Auto or Manual

Postponement Rule: NO POSTPONEMENT

Status: ACTIVE

Ownership: Self

Owner: Data Manager

Buttons: Null Field, Calc Amt, Compare Amt, Notify, Data Selector, Trigger Mode, Remove Trigger, Debug / Test, Quit, Update

3 Different Fields appear with this Type.

**Calc Amt From** A base value is derived from the data -  
Please specify as per the following example :-

dataname + dataname - dataname + "v10000" + "%10.5"

where: any number of datanames (single or multi) may be listed; '+' and '-' are interchangeable; amounts must be specified in quotes starting with 'v'; percentages must be specified in quotes starting with '%'; all values are treated as if no conversion applies ...

**Measure** This Value is used in conjunction with 'Calculate Amount' and 'Compare with Amount' to determine whether we have an Alert or not.

Choose one of the following :-

- > (greater than)
- < (less than)
- >= (greater or equal)
- <= (less than or equal)
- # (not equal)
- = (equal)

**Compare with Amt** Please specify as per the following example :-

dataname + dataname - dataname + "v10000" + "%10.5"

where: any number of datanames (single or multi) may be listed; '+' and '-' are interchangeable; amounts must be specified in quotes starting with 'v'; percentages must be specified in quotes starting with '%'; all values are treated as if no conversion applies; NULL value is interpreted as zero ...

Note: values are processed in sequence, with percentage always being applied (whether - or +) to the running total at that point in time, therefore NEVER have a percentage as the first element;

example: add Contract.Sum & Contingency.Sum then add 5%  
specify: contract.sum + contingency.sum + "%5"

And we note that there are 2 new Function Buttons to assist in the definition of 'Calc Amt' and 'Compare Amt'.

The final type, ALERT BY SUBROUTINE (type 999), allows you to call a Subroutine to determine whether an Alert is being raised or not.

**ALERT TRIGGER TYPE 999 (UPDATE)**

1 Main | 2 Notes

Trigger Key: 1307049511

Type: 999 Alert by Customized Subroutine(s)

Description: [Empty]

File Name: [Empty]

Opt Null Field Check: [Empty]

Alert Sub: [Empty]

Notify Whom: [Empty]

Trigger Mode: PROCESS TRIGGER

Inspection Frequency: CHANGE

Data Selector: \*\*\* Item Key

Message Frequency: ONCE

Sign Off Condition: Either Auto or Manual

Postponement Rule: NO POSTPONEMENT

Status: ACTIVE

Ownership: Self

Owner: Data Manager

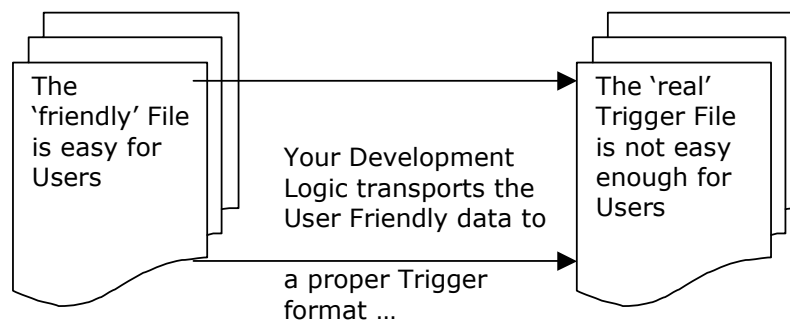
Buttons: Null Field, Proceed Sub, Alert Sub, Notify, Data Selector, Trigger Mode, Remove Trigger, Debug / Test, Quit, Update

**Alert Sub:** A Subroutine Call is specified to process the Alert condition. Otherwise, the Fields are the same as for the other types.

**Alert Sub:** Your subroutine will always be called with the following arguments: (filename,datakey,itemcontent,alert)

'alert' should return 'y' or 'n' to indicate whether an alert condition exists or still exists; once an Alert is generated (whether with or without an Alert Message, SignOff (if Automatic is allowed)) will occur when the Subroutine is called on regular inspection and returns 'n';

The intent with a new Trigger Type is to design a Screen Process, with 'triggerfriendly' as the primary File, that presents a Screen that is easy for the User to define and understand, and which translates the User choice(s) from the 'friendly' file to the 'real' trigger File (as shown previously and shown again below).



Below, we show an example of such a Screen, in this case for Trigger Type 001.

**ALERT TRIGGER 001 (UPDATE)**

Trigger Key: 1307039104

Type: 001 Daily Diary Alert  
 Description: Daily Diary Alert  
 Owner: Data Manager

Status: ACTIVE

Buttons: Debug / Test, Remove Trigger, Quit, Update

When the User chooses UPDATE, our (or your) Subroutine performs the Logic that translates the 'User Friendly' Trigger Data to a 'real' Trigger.

So we repeat: The Function of a User Friendly Trigger is a mechanism that allows the User to easily define a Trigger, which then creates a 'real' Trigger as if the User had defined a Trigger from one of the generic Trigger Types, directly.

As we have already seen, you define a new Trigger Type, which includes a Process for 'new' and a Process for 'amend', so that when the User wishes to define a New Trigger of a selected type, or amends an existing trigger in his / her Queue, then the system knows which Process to offer to achieve the objective.

Your Processes are always defined as Screen Processes, they always operate on file 'triggerfriendly' as the primary File, and they are always UPDATE mode, but the system does not allow the User to be able to use the standard SAVE function. Instead, you always provide 1 or more Function Buttons, which may be called SAVE or UPDATE or any other appropriate term, and such function(s) will call your defined Subroutine, which will use the current Screen Data to either submit a NEW Trigger or UPDATE an existing Trigger.

If we have another look at the 'friendly' type 001 shown above, we can see on the Screen that the type is '001'. If we use the TRIGGER TYPE function from the Alerter Wizard, and we retrieve '001', we see the following -

**DEFINE ALERT TRIGGER TYPES (UPDATE)**

Trigger Type Key: 001  
 Description: Daily Diary Alert

Process for New: triggerfriendly-001 Alert Trigger 001  
 Process for Amend: triggerfriendly-001 Alert Trigger 001

Auto Remove (Once off Triggers)  
 Developers Only ?  
 May Use ?

Help Description: This Alert Trigger will warn on a Daily basis if there are any Entries in your Diary that are 'older' than today, i.e. to have been attended to before today. The Alert is not per entry, but rather per Diary, i.e. will alert once for 1 or more Entries older than Today.  
 Hint: Diary Entries may be moved to future Dates as required. The Alert is intended to Alert you to the fact (if this is the case) that you have 1 or more entries in your Diary which you

This type uses a single Process for 'new' and 'amend' of this particular Trigger Type

The system will display this Text when the User selects a Trigger Type 001, so the User can decide whether the description fits what he / she wishes to do ...

If we go to the Menu Objects, and retrieve the Menu Process called 'triggerfriendly-001', we find the following -

**PROCESS LIBRARY (UPDATE)**

1 Main | 2 Advanced | 3 Usage

Process Key: triggerfriendly-001

\*Menu Description: Alert Trigger 001

Logical Driver: Screen  
 Object To Drive: triggerfriendly-001 :Daily Diary Alert  
 Web Screen: :

We can see that the Process is a Screen, and the Screen itself has the same Key, e.g. 'triggerfriendly-001'. On file 'triggerfriendly' we already have all the Fields that are used for definition of the generic types, but we can also add any required Fields for use in presenting 'friendly' Screens for our Users. If we retrieve the

Screen Definition for 'triggerfriendly-001', we can ascertain the program that is called when the User chooses UPDATE.

|   | Button Caption | Form # | X-Pos | Y-Pos | Width | Height | Sub / Process      |
|---|----------------|--------|-------|-------|-------|--------|--------------------|
| 1 | Quit           | 1      | 68    | 12    | 10    | 2      | unload.form        |
| 2 | Update         | 1      | 68    | 14    | 10    | 2      | next/trifriend001u |
| 3 | Remove Trigger | 1      | 68    | 7     | 10    | 2      | adef.assist.remove |
| 4 | Debug / Test   | 1      | 68    | 5     | 10    | 2      | trigger.debug      |

Associated with UPDATE, we see the Process call is 'next/trifriend001u'.

*Hint: 'next/' ensures that the last User action is validated and processed before the 'trifriend001u' Subroutine is called.*

And the 'trifriend001u' program looks like this -

```

subroutine trifriend001u
*
equate am to char(254), vm to char(253), svm to char(252)
*
*
*** *** *
* use this program as the basis for designing your own Custom
* Trigger Update Programs
*
* The SAVE Function on the TRIGGERFRIENDLY File is inhibited;
* the User must always use your UPDATE button which calls your Program
*
* Hint: You also want your data saved in the 'triggerfriendly' file
* since you may have additional paramaters to what will
* be saved in the 'alertt' Trigger Master File;
*
*
*** *** *

*** *** * get the Key and Record Contents, as well as the Menu Process
* Key in case it's needed

param = '8'
dat = ""
call s.peek(param,dat)
ProcessId = dat<1,1>

param = '10'
dat = ""
call s.peek(param,dat)
RecordKey = dat<1,1>

param = '11'
dat = ""
call s.peek(param,dat)
RecordData = dat

```

```
*** *** *
```

```
* now use your data to build TriggerData before calling the Trigger  
* Submit program
```

```
*** *** *
```

```
* there are 24 attributes required for TriggerData, not all of which  
* will apply for all Trigger Types, but if we keep the Program Design  
* simple and clear, it is easy to change it for each next TriggerType  
* we may do ...
```

```
TriggerData = "
```

```
gosub td01 ; * Key  
gosub td02 ; * Our Trigger Type  
gosub td03 ; * The system Generic Trigger Type  
gosub td04 ; * Description / Title  
gosub td05 ; * Primary FileName  
gosub td06 ; * Active Status  
gosub td07 ; * Mode  
gosub td08 ; * Data Selector  
gosub td09 ; * Inspection Frequency  
gosub td10 ; * Message Frequency  
gosub td11 ; * Notification  
gosub td12 ; * SignOff  
gosub td13 ; * Alert Message  
gosub td14 ; * Postponement  
gosub td15 ; * Owner  
gosub td16 ; * Notes  
gosub td17 ; * Generic 998 Only: Calc Amt  
gosub td18 ; * Generic Types 996/7: Calc Date  
gosub td19 ; * Generic Type 998: Measure  
gosub td20 ; * Generic Type 996: Warn Before  
gosub td21 ; * Generic Type 997: Warn After  
gosub td22 ; * Generic Type 998: Compare Amt  
gosub td23 ; * Null Field Check: Optional with all Types  
gosub td24 ; * Generic Type 999: Alert Subroutine Call
```

```
*** *** *
```

```
* In this program example, we will fill all the Fields, as if coming  
* from the standard Generic Type Attributes, and as if we are doing  
* all the Generic Types at once (which we cannot, of course);  
* No doubt in your Custom Programs you will apply some logic and derive  
* some of the Values from Custom Fields that you define on File  
* TRIGGERFRIENDLY ...  
*
```

```
* Note the standard Dict positions as specified for TriggerFriendly,  
* other than new ones that you define, in the Dict Manager;
```

```
*** *** *
```

```
* when your data is ready, call the Trigger Submit
```

```
Result = "
```

```
call submit.alert.trigger(TriggerData,Result)
```

```

*** *** * was it accepted?
if Result = " then
  * wow, it worked!
end else
  mes = 'Trigger not Accepted, because :'
  mes = mes : char(13) : char(13) : Result
  param = '8'
  call s.push(param,mes)
  * return to Screen
  go mainexit
end

*** *** *

* save the data on TRIGGERFRIENDLY also
file triggerfriendly
triggerfriendly = RecordData
write triggerfriendly on RecordKey
* we use the same Key as for our Trigger!

*
go mainexit
*** subs start
*
td01: *
  * Key
  TriggerData<1> = RecordKey
  * Note that we want the Trigger to have the same Key as our Recordkey, with
  * which Key we will update 'triggerfriendly'; That way, it is the same Key that
  * will be used when the User selects the Trigger for Editing
return
*
td02: *
  * Our Trigger Type we will probably specify on the Screen Spec
  * as a Default, and not allow the User to access it
  TriggerData<2> = RecordData<4>
return
*
td03: *
  * The system Generic Trigger Type
  TriggerData<3> = '999'
return
*
td04: *
  * Description / Title
  TriggerData<4> = RecordData<1>
return
*
td05: *
  * Primary FileName
  TriggerData<5> = 'ilusers'
return
*
td06: *
  * Active Status
  TriggerData<6> = RecordData<28>
return

```

```

*
td07: *
  * Mode
  TriggerData<7> = 'm'
return
*
td08: *
  * Data Selector
  TriggerData<8> = 'trifriend001d'
return
*
td09: *
  * Inspection Frequency
  TriggerData<9> = 'daily'
return
*
td10: *
  * Message Frequency
  TriggerData<10> = 'h 24'
return
*
td11: *
  * Notification
  TriggerData<11> = 'sub trifriend001n'
return
*
td12: *
  * SignOff
  TriggerData<12> = 'e'
return
*
td13: *
  * Alert Message
  TriggerData<13> = 'Diary Entries older than TODAY!'
  TriggerData<13,-1> = ' '
  TriggerData<13,-1> = 'You have 1 or more Diary Entries that'
  TriggerData<13,-1> = 'need to be attended, e.g. older than'
  TriggerData<13,-1> = 'TODAY!'
return
*
td14: *
  * Postponement
  TriggerData<14> = 'no'
return
*
td15: *
  * Owner
  TriggerData<15> = RecordData<27>
return
*
td16: *
  * Notes
  TriggerData<16> = RecordData<2>
return
*
td17: *
  * Generic 998 Only: Calc Amt

```

```

return
*
td18: *
    * Generic Types 996/7: Calc Date
return
*
td19: *
    * Generic Type 998: Measure
return
*
td20: *
    * Generic Type 996: Warn Before
return
*
td21: *
    * Generic Type 997: Warn After
return
*
td22: *
    * Generic Type 998: Compare Amt
return
*
td23: *
    * Null Field Check: Optional with all Types
return
*
td24: *
    * Generic Type 999: Alert Subroutine Call
    TriggerData<24> = 'trifriend001a'
return
*

**** subs end
*
mainexit: *
*
return

```

When you look at the code in this program, bear in mind that as the Developer, you need to decide for the Trigger Type in question, which Data you need from the User and which you can supply automatically (by making these decisions on behalf of the User), in order to fulfill the requirements of Trigger Submission, which is done with the system call: `call submit.alert.trigger(TriggerData,Result)`

The system help for this call is as follows :-

#### PROGRAMMATIC ALERT TRIGGER SUBMISSION

Alert Trigger Submission for types other than the generic 996-999 range are done from the TRIGGERFRIENDLY File Interface, using a Command Button (Proceed, Update, etc.) that calls your Subroutine, that in turn puts together the Data in the required format, and calls the Shuttle Routine 'submit.alert.trigger(dynam,result)'; This call can be made either for a new Trigger or to Update the details of an existing Trigger spec. If 'result' returns NULL then the Update was accepted, otherwise 'result' = Reason Rejected.

DYNAM, the single Argument for the call, should be a dynamic array with the following Parameters :-

Attribute # 1: The Record Key (mandatory, unique unless pre-existing)  
Note: You may derive your Unique Key for new Triggers by calling 'get.unq.11(key)' - this is a Shuttle Routine that will provide you with a Unique Key. However, when using 'triggerfriendly' you also want the same key for the Trigger as for the Record Update that you write to 'triggerfriendly', so whether you use a new Unique Key or use the unique Key that the system has provided as Record Key for the Screen being executed on 'triggerfriendly' while defining the new Trigger, you should ensure that the same Key is used for the Trigger and for the Record Update to 'triggerfriendly'. That will ensure that when the User subsequently selects the Trigger for amendment, that the correct data on 'triggerfriendly' is found.

Hint: Best practice is to determine what the current Screen Key is, as in the program example shown above, and use that for the Trigger Key also. This way, you can use the same screen for Defining new Triggers and Editing them, since the Key will Automatically be the same on the 'real' Trigger File and on 'triggerfriendly'.

Attribute # 2: Trigger Type (mandatory, as defined on FileName 'atriggertypes');

Attribute # 3: Generic Trigger Type (mandatory, in range 996-999). All Trigger Types that are defined must employ one of the 4 generic types for final execution;

Attribute # 4: Short Description or Trigger Title (mandatory, for identification);

Attribute # 5: Primary FileName (mandatory, must exist on FileManager in the current DataMart);

Attribute # 6: Active Status (mandatory, 'a'ctive or 'i'nactive);

Attribute # 7: Mode (mandatory, 'P'rocess or 'M'onitor);

Attribute # 8: Data Selector (if (7) above is 'M', then this is mandatory, otherwise it is ignored; When specified, it should be a 'no-pause' Data Selector, i.e. without Runtime Prompts using "?" ...);

Attribute # 9: Inspection Frequency (for 'P' in (7) above, this is always 'C'hange, while for 'M' in (7) above, it is either 'C'ontinuously or 'D'aily);

Attribute #10: Message Frequency (mandatory, specify as 'once', 'never', 'h 1', 'h 2', 'h 3', 'h 4', 'h 8' or 'h 24');

Attribute #11: Notification (mandatory, even if (10) above is 'never';  
specify as: u usercode usercode ...  
or: d dname dname ...  
or: sub subname where subname will be called

with (fname,datkey,trigkey,users) and 'users' will return UserCodes to Notify;

Attribute #12: SignOff (mandatory, 'A'uto, 'M'annual, 'E'ither or 'I'mmediate);

Attribute #13: Alert Message (mandatory, either multi-valued Text or %subname% where 'subname' will be called with subname(fname,datkey,trigkey,mesg) and 'mesg' will return multi-valued Text to use);

Attribute #14: Postponement Rule (mandatory, indicate as 'no' for not at all, 'date' where User may choose Date, or number of days as 1, 2, 5, 10 or 30);

Attribute #15: Owner (mandatory, for System Owned, specify "phant" else the UserCode who owns the Trigger);

Attribute #16: Notes (optional, multi-valued Text);

Attribute #17: Calculate Amount (mandatory only for generic type 998; specify as explained on generic type 998);

Attribute #18: Calculate Date (mandatory for generic types 996 & 997; specify as explained for these types);

Attribute #19: Measure (mandatory for generic type 998; specify as "=", "#", ">", "<", ">=" or "<=");

Attribute #20: Warn Before Days (mandatory for generic type 996; specify in range '0' to '-999');

Attribute #21: Warn After Days (mandatory for generic type 997; specify in range '0' to '999');

Attribute #22: Compare with Amount (mandatory for generic type 998; specify as explained on generic type 998);

Attribute #23: Null Field Check (optional with all generic types; specify DataName on Primary FileName for which Value must = NULL as a pre-condition for an Alert);

Attribute #24: Alert Subcall (mandatory for generic type 999; specify 'subname' to call; subname will be called like subname(fname,datkey,recordcontent,alert) and alert should return 'y' or 'n' for Alert Condition);

### **Summary of steps to define a new Trigger Type**

- 1) Decide what the Trigger has to do, and following from that, decide which Fields (from all the Fields required to submit a Trigger with "submit.alert.trigger") can be provided by your logic, and which must be captured by the User.
- 2) If necessary, you may define new required Fields on file 'triggerfriendly'.
- 3) Now design an Update Screen on file 'triggerfriendly' for definition and editing of this Trigger Type.

- 4) Ensure that the Key for the Screen is specified as “\*var36”, and that the User may not access the Key Field. Whether the Key Field is visible to the User is up to you. Ensure also that the Screen is not set for ‘repeat’.
- 5) Provide an ‘UPDATE’ or similar button which calls the Subroutine that you have created for formatting the Data before calling ‘submit.alert.trigger’. (See our example higher up.)
- 6) Create a Menu Process for the new Screen, and ensure the process is not allowed on Menus (see ‘advanced’ Tab for Menu Processes while defining the Menu Process).
- 7) Define the new Trigger Type with the “Trigger Types” option, and specify the Screen Process (in 6 above) for ‘new’ and ‘edit’, plus the Help Description for the Users. The new Trigger Type may now be tested and / or used.

We understand that the creation of your 1<sup>st</sup> Trigger Type will be a mission, but it gets easier with each one you do. If you have further questions regarding the process, please feel free to mail us at [support@infolab.cc](mailto:support@infolab.cc)

© Infolab, 2003.

This Documentation is copyrighted by Infolab (Pty) Ltd. [ [www.infolab.cc](http://www.infolab.cc) ] All rights are reserved. Licensed Shuttle™ Users are granted permission, for internal use ONLY, to reproduce the Documentation, and to include amendments dealing with specific instructions local to your installation.