# Importing TAB and XML Data

## About TAB and XML Imports

The CoPilot Health Management System program provides functions to import and export data in the following two formats:

1. TAB – Tab Delimited Text

2. XML – Extensible Markup Language

These formats are used within the program to support archiving but they may also be used to import data from external sources that would not otherwise be supported by the CoPilot program.

## Import Capabilities

The import functions of the CoPilot program are limited to the following event data types:

Exercise Data
Glucose Data
Basal Insulin Data
Bolus Insulin Data
Lab Results
Meal Information
Medical Exam Results
Medication
Notes
State of Health
Ketone Data
Alarm Events
Generic

## Required Skills

The user will be required to have sufficient technical skills to be able to manipulate or create data files as outlined in this document. File imports of this sort should not be undertaken by users who do not have a full understanding of the formatting requirements.

**Caution –** The user must check the data for appropriate formats before importing the data into CoPilot.

## File Formats

For the purposes of simplifying the documentation and basic understanding of the import functions, this document concentrates on how to format the data for TAB file imports. There are many similarities between the TAB and XML formatting requirements. The specific XML requirement differences are included at the end of this document under **XML Standards** section.

There are many methods available to create acceptable TAB import files for CoPilot. Most of the example information in this document assumes that users will develop the import files in Excel.

The following are the two limitations associated with Excel:

1. Under some circumstances, Microsoft Excel may add illegal quotation marks around field entries when a file is saved. This occurs when commas or other common delimiter characters are included as part of the data. The user must remove these quotation marks prior to importing a file into CoPilot.
2. In order to avoid differences in values and times, when converting back and forth between Microsoft Excel and CoPilot, it is required that all values in Microsoft Excel shall be handled with full double-precision accuracy. This can be accomplished in Microsoft Excel by setting the cell formatting of these numbers to at least ten decimal places.

## File Structure

A CoPilot TAB import file is a flat, ASCII text file composed of records arranged physically in rows. Each row constitutes the complete entry for a single data event.

Each row is made up of several defined data fields with each field separated by a TAB character (ASCII 09). Each row is terminated by an ASCII carriage return and/or line feed.

## File Type

A TAB import file must have the filename extension **.TAB**. An XML import file must have the filename extension **.XML**.

## Header Row

The import file must have a header row, which is the first row in the file. The header row must include all of the following field labels, exactly as presented below, and separated by TAB characters:

DATEEVENT
TIMESLOT
EVENTTYPE
DEVICE_MODEL
DEVICE_ID
VENDOR_EVENT_TYPE_ID
VENDOR_EVENT_ID
KEY0
KEY1
KEY2
I0
I1
I2
I3
I4
I5
I6
I7
I8
I9
D0
D1
D2
D3
D4
C0
C1
C2
C3
C4
C5
C6
C7
C8
C9
ISMANUAL
COMMENT
DELETED

## Data Rows

Each row following the Header row must contain data corresponding to the header fields. In some cases, fields may be empty but TAB characters must still be inserted to separate the field locations.

## Data Fields

There are two different types of fields in each data row: Constant Data Fields and Event Type Dependent Data Fields. All field values are ASCII character text, but some fields must be numeric (made up of numerals that can be interpreted as an integer or floating point numbers). CoPilot handles formatting and wrapping of text, so it is not necessary (or advised) to include carriage returns or line feeds within text fields. Unused "C" fields (C1, C2, C3, etc.) may be left empty. Unused "I" fields or "D" fields (I0, i1, D2, etc.) must be set to the numeral zero.

### Constant Data Fields

The following fields are associated with every data record and their format is consistent, regardless of event type.

#### *DATEEVENT*

The DATEEVENT field contains the date and time of the event being recorded. The field is a double-precision floating point number that can be converted to a Microsoft Windows compatible date.

By convention, the integer portion of the number represents the number of days since the seed, or starting date, of the set. In this case, the seed date, having a value of zero, is December 30, 1899 12:00 AM. The fractional part of the date value is the portion of the day that has elapsed.

**Example:** 0.5 represents December 30, 1899 12:00:00 PM.

#### *TIMESLOT*

Most data entered into CoPilot is associated with a time period or time slot. The values entered in this field must be integers, according to the following scheme:

Pre-Breakfast = 0

Post-Breakfast = 1

Pre-Lunch = 2

Post-Lunch = 3

Pre-Dinner = 4

Post-Dinner = 5

Bed Time = 6

Sleep = 7

To have CoPilot enter a time slot based on the user's time period preferences for the given event time, set the TIMESLOT value to -1. Refer to ***Meal*** *EVENTTYPE* section for additional information on assignment of TIMESLOT values.

### *EVENTTYPE*

Event types are entered as integers according to the following scheme:

Exercise = 0

Glucose = 1

Basal Insulin = 2

Bolus Insulin = 3

Lab Results = 4

Meal = 5

Medical Exams = 6

Medications = 7

Notes = 8

State of Health = 9

Ketone = 10

Alarms = 15

Generic = 16

**Note –** Event types = 11, 12, 13 & 14 are reserved.

### *DEVICE_MODEL*

The DEVICE_MODEL field is a string data type with a maximum length of 64 characters. This field must be populated with the name (description) of the device from which the imported data was generated. If an import file is prepared containing data from a source that may also be directly uploaded or imported into CoPilot, the description must exactly match the description used in CoPilot. Otherwise, CoPilot's ability to filter using this data will be limited. When multiple instances of import files are developed that contain data from the same device, precautions must be taken that the DEVICE_MODEL is identical in each instance. Filtering based on DEVICE_MODEL is case sensitive. For all data to be imported correctly into CoPilot, this field must be populated.

### *DEVICE_ID*

The DEVICE_ID field is a string data type with a maximum length of 64 characters. This field must be populated with the specific unique identification of the device from which the imported data was generated (usually the serial number). If an import file is prepared containing data from a source that may also be directly uploaded or imported into CoPilot, the DEVICE_ID must exactly match the ID in CoPilot. Otherwise, CoPilot ability to filter using this data will be limited. When multiple instances of import files are developed that contain data from the same device, precautions must be taken that the DEVICE_ID is identical in each instance. Filtering based on DEVICE_ID is case sensitive. For all data to be imported correctly into CoPilot, the field must be populated.

## VENDOR_EVENT_TYPE_ID

The VENDOR_EVENT_TYPE_ID is an integer field that is used internally to differentiate among different classes of event identifiers. For all CoPilot events described in this document, this field must be populated with the numeral zero.

## VENDOR_EVENT_ID

The VENDOR_EVENT _ID field is a string data type with a maximum length of 64 characters. If this field is not used then it must be left empty. The VENDOR_EVENT_ID field assists CoPilot in identifying and handling duplicate event records. If the device or source from which data is being imported contains a unique, non-repeating, record identifier for each data record, this field can be used to avoid importing multiple instances of the same event record from a device. To test for and define duplicated records, CoPilot combines the VENDOR_EVENT_ID field with its unique User Identification, the DEVICE_MODEL field, and the DEVICE_ID field.

## *KEY0, KEY1, KEY2*

The KEY0, KEY1, and KEY2 fields check for and reject duplicate events being saved by CoPilot. Each field is an integer; the comparison for duplicate checking purposes is based on the combination of User ID, DATEEVENT, EVENTTYPE and the AND of the three KEY fields. These fields generally describe the "value" of the event record. For an event to be saved in the CoPilot database, the KEY value must be populated with a value that is unique to the database for that EVENTTYPE. For CoPilot to correctly check for and reject duplicate events, these fields must be populated.

**Note:** The above logic is implemented strictly for the evaluation of duplicates. If you do not populate the KEY fields (set them to zero), then all subsequent event records, which have an identical User, DATEEVENT, and EVENTTYPE, will be rejected as duplicates. On the other hand, if you populate the three KEY fields with large random integers (unlikely to be duplicated) all records will be saved, regardless of their similarity with other records in the database.

## *ISMANUAL*

The ISMANUAL field indicates whether the original source of the data was from a device or from manual entry. This field in CoPilot determines whether time, date, or value information can be edited by the CoPilot user.

**ISMANUAL = 1,** indicates that the data was manually entered (editing should be allowed).

**ISMANUAL = 0,** indicates that the data was uploaded from a device (editing should not be allowed).

**Note -** CoPilot only allows this kind of editing for manually entered data.

## *COMMENT*

The COMMENT field is optional free text, with a maximum 250 characters.

### *DELETED*

DELETED = 0 means event record in CoPilot is included in statistical calculations or report views.

DELETED = 1 means event record is hidden in CoPilot and not included in statistical calculations or report views.

## Event Type Dependent Data Fields

This section describes how to populate fields I0-I9, D0-D4, and C0-C9 for each EVENTTYPE. Each EVENTTYPE description includes the required data for each field. Blank fields in the table must be filled with zeros in the case of "I" and "D" fields (numeric types) and they may be left empty in the case of "C" fields. "I" fields are numeral values that can be mapped as integers; "D" fields are numeral values that can be mapped as double-precision floating point values; and "C" fields values are printable character strings with a maximum length of 250 characters.

This section also describes the KEY0, KEY1, and KEY2 fields that must be populated in the CoPilot program for data input. In some cases, these KEY values can be created by hashing string values to integers using the Elf Hash algorithm. Refer to the last section under **ElfHash** of this document for a Pascal code version of this algorithm.

**Note -** To import data files, it is not mandatory to follow the same duplicate KEY conventions employed by CoPilot, as long as the employed convention provides ample detection and rejection of duplicate entries.

## *Exercise {EVENTTYPE = 0}:*

The Exercise EVENTTYPE is intended for use with exercise data as follows:

| Field | Data Content | Implementation Notes |
|---|---|---|
| KEY0 | ElfHash ( Duration ) | |
| KEY1 | ElfHash( ExcerciseType ) | |
| KEY2 | ElfHash( Intensity ) | |
| I0 | | |
| I1 | | |
| I2 | | |
| I3 | | |
| I4 | | |
| I5 | | |
| I6 | | |
| I7 | | |
| I8 | | |
| I9 | | |
| D0 | | |
| D1 | | |
| D2 | | |
| D3 | | |
| D4 | | |
| C0 | Exercise Type | Free text describing the kind of exercise being documented. |
| C1 | Duration | Free text describing the duration of the exercise, typically in hours. |
| C2 | Intensity | Free text describing the intensity of the exercise, typically Low, Medium, or High. |
| C3 | | |
| C4 | | |
| C5 | | |
| C6 | | |
| C7 | | |
| C8 | | |
| C9 | | |

## *Glucose {EVENTTYPE = 1}:*

The Glucose EVENTTYPE is intended for use with glucose data as follows:

| Field | Data Content | Implementation Notes |
|---|---|---|
| KEY0 | Glucose value | Glucose in mg/dL. |
| KEY1 | | |
| KEY2 | | |
| I0 | Control Flag | 0 = Not a control value<br>1 = Control value. |
| I1 | Glucose Value | Expressed in mg/dL. |
| I2 | Out of Range Flag | -1 = Low (Negative 1)<br>0 = InRange<br>1 = Hi |
| I3 | Hours Since Last Meal | Value = Hours:Min<br>0 = 0:00<br>1 = 0:15<br>2 = 0:30<br>3 = 0:45<br>4 = 1:00<br>5 = 1:30<br>6 = 2:00<br>7 = 2:30<br>8 = 3:00<br>9 = 3:00<br>10 = 4:00<br>11 => 4:00. (Optional) |
| I4 | Temperature Flag | 0 = Not out of temperature range<br>1 = Out of temperature range<br>Note: If this flag is set, the glucose event will still appear in the CoPilot Diary List Report, but will not be displayed in other CoPilot reports, and statistical calculations. |
| I5 | CM Flag | 0 = discrete measured glucose values<br>1 = glucose data from a continuous monitor source, such as FreeStyle Navigator. |

| | | |
|---|---|---|
| I6 | Trend | Indicates the trend of the continuous glucose data as of the current measurement as follows:<br>0 = No Symbol<br>1 = Up Arrow Symbol<br>2 = Up Angle Arrow Symbol<br>3 = Level Arrow Symbol<br>4 = Down Angle Arrow Symbol<br>5 = Down Arrow Symbol<br>**Note -** This field is ignored unless I5 = 1 |
| I7 | | |
| I8 | | |
| I9 | | |
| D0 | | |
| D1 | | |
| D2 | | |
| D3 | | |
| D4 | | |
| C0 | Sample Site | Free Text. (Optional) |
| C1 | Cal Code | Free Text. (Optional) |
| C2 | Device Status | Free Text. (Optional) |
| C3 | | |
| C4 | | |
| C5 | | |
| C6 | | |
| C7 | | |
| C8 | | |
| C9 | | |

## *Basal Insulin {EVENTTYPE = 2}:*

The Basal Insulin EVENTTYPE is intended for use with pumped basal insulin only. To be recognized correctly by the program, basal insulin events on a given day must also have a basal insulin record that indicates the total pumped basal insulin for the day. (I0 = 2, D1 = total units of basal insulin pumped during the 24-hour day.)

Basal insulin profiles are created by a series of Basal Insulin events, each describing a new delivery rate that is initiated at a specified time. Insulin delivery is terminated by an event with a basal rate value (D0) of zero.

| Field | Data Content | Implementation Notes |
|---|---|---|
| KEY0 | BasalOrder | See I1. |
| KEY1 | Trunc(Rate*1000) | The basal rate or "value" field.<br>The value is truncated after Rate is multiplied by 1000. |

| | | |
|---|---|---|
| KEY2 | BasalKind | See I0. |
| I0 | BasalKind | 0 = Programmed<br>1 = Temporary<br>2 = Total Pumped Today. |
| I1 | BasalOrder | Where there are multiple basal events being posted in the same minute, this indicates the appropriate sequence order for the events. This can be critical, for example, in cases where basal segments are stopped and started in the same minute, and it is significant which event came first. The BasalOrder field is an integer that indicates sort-order of the events within that minute, with the lowest value sorted first. |
| I2 | | |
| I3 | | |
| I4 | | |
| I5 | | |
| I6 | | |
| I7 | | |
| I8 | | |
| I9 | | |
| D0 | Rate | Units per hour.<br>**Note –** The basal insulin events are entered as rates. |
| D1 | Total Pumped | Total Pumped Today – Units<br>(Ignored unless I0 = 2). |
| D2 | | |
| D3 | | |
| D4 | | |
| C0 | ProfileName | Not currently supported, should be empty. |
| C1 | Description | Free text. |
| C2 | | |
| C3 | | |
| C4 | | |
| C5 | | |
| C6 | | |
| C7 | | |
| C8 | | |
| C9 | | |

### *Bolus Insulin {EVENTTYPE = 3}:*

The Bolus Insulin EVENTTYPE is intended for use with pumped bolus insulin only. To be recognized correctly by the program, pumped bolus insulin events on a given day must also have a corresponding bolus insulin record that indicates the total pumped bolus insulin of that type (Correction or Meal) for the day. (I0 = 3 or 4, D1 = total units of bolus insulin of that type pumped during the 24-hour day.)

| Field | Data Content | Implementation Notes |
|---|---|---|
| KEY0 | BolusKind | |
| KEY1 | Trunc(Dosage*1000) | The value is truncated after Dosage is multiplied by 1000. |
| KEY2 | | |
| I0 | BolusKind | 0 = Meal<br>1 = Correction<br>2 = General<br>3 = Correction Total Pumped Today<br>4 = Meal Total Pumped Today<br>**Note -** Individual pump bolus events should always use BolusKind = 0 or 1.<br>Manual bolus events should always use<br>BolusKind = 0 or 1 or 2. |
| I1 | | |
| I2 | | |
| I3 | | |
| I4 | | |
| I5 | | |
| I6 | | |
| I7 | | |
| I8 | | |
| I9 | | |
| D0 | Dosage | Insulin bolus delivered, in Units of Insulin. |
| D1 | TotalPumped | Total Bolus insulin pumped today - in Units of Insulin.<br>Ignored unless I0 = 3 or 4. |
| D2 | | |
| D3 | | |
| D4 | | |
| C0 | Insulin Name | Free text. |
| C1 | Bolus Type | Free Text.<br>(Characterizations, such as Square Wave, Combination, Injection, etc.) (Optional) |
| C2 | Description | Free text.<br>(General information about the insulin delivery.) (Optional) |

| | | |
|---|---|---|
| C3 | | |
| C4 | | |
| C5 | | |
| C6 | | |
| C7 | | |
| C8 | | |
| C9 | | |

### *LabResults {EVENTTYPE = 4}:*

The LabResults EVENTTYPE is intended for reporting laboratory test results. Use consistent test names since the report function will collate and sort lab results by the TestType.

| Field | Data Content | Implementation Notes |
|---|---|---|
| KEY0 | ElfHash( ResultValue ) | |
| KEY1 | ElfHash( testType ) | |
| KEY2 | ElfHash( ReferenceRange + Units ) | |
| I0 | | |
| I1 | | |
| I2 | | |
| I3 | | |
| I4 | | |
| I5 | | |
| I6 | | |
| I7 | | |
| I8 | | |
| I9 | | |
| D0 | | |
| D1 | | |
| D2 | | |
| D3 | | |
| D4 | | |
| C0 | Reference Range | Laboratory assigned reference range for this result. (Optional) |
| C1 | ResultValue | The reported result for this test. |
| C2 | TestType | Name or description of the test. |
| C3 | Units | Units of measure for the Result. |
| C4 | | |
| C5 | | |
| C6 | | |
| C7 | | |
| C8 | | |
| C9 | | |

### *Meal {EVENTTYPE = 5}:*

The Meal EVENTTYPE is intended for reporting meal data. The Meal EVENTTYPE uses the TIMESLOT field differently than other events. The TIMESLOT field must be populated according to the meal as follows: 0=Breakfast, 1=Lunch, 2=Dinner, 3=Snack. If the TIMESLOT field is set to -1, CoPilot will calculate the meal category based on the Users Time Period Preferences.

| Field | Data Content | Implementation Notes |
|---|---|---|
| KEY0 | Trunc( Carbohydrates*1000 ) | The value is truncated after Carbohydrate is multiplied by 1000. |
| KEY1 | ElfHash(FloatToStr(Calories) + ‘ ‘ + FloatToStr(Protein) + ‘ ‘ + FloatToStr(Fat) ) | |
| KEY2 | ElfHash(Description0+ Description1 + Description2 + … + Description7) | |
| I0 | | |
| I1 | | |
| I2 | | |
| I3 | | |
| I4 | | |
| I5 | | |
| I6 | | |
| I7 | | |
| I8 | | |
| I9 | | |
| D0 | Calories | Unused. |
| D1 | Carbohydrates | Total carbohydrates for the meal, in grams. |
| D2 | Fat | Unused. |
| D3 | Protein | Unused. |
| D4 | | |
| C0 | Description0 | See note below. |
| C1 | Description1 | |
| C2 | Description2 | |
| C3 | Description3 | |
| C4 | Description4 | |
| C5 | Description5 | |
| C6 | Description6 | |
| C7 | Description7 | |
| C8 | Meal Category | This field is used internally by CoPilot. Leave empty. |
| C9 | OtherInfo | Free text. |

**Note:** The Description fields store string descriptions of the food related data in the following format:

X|Y|Z where:

X= number of servings
Y= number of grams carbohydrate per serving
Z= description of the food item (N/A if not available)

If multiple food items are included in the meal, the strings above are concatenated and separated by a caret (ASCII 94) character:

X|Y|Z^X|Y|Z^X|Y|Z^X|Y|Z

Note that the separators "|" and "^" (pipe and caret) are reserved strings and cannot be found in the data.

In the simplest case, where food item descriptions are not available, a 15 gram meal is represented as:

1|15|N/A

(This is also a special case in that the resulting record, when displayed in the CoPilot Diary List Report, will be blank in the Description Column – only reporting the total Carbohydrates in the Value Column.)

The total length of all the Description string must not exceed 2000 characters. The description string must be parsed into pieces such that each string does not exceed 250 characters. The first 250 characters are stored in the field Description0, the second 250 characters are stored in the field Description1, and so forth through Description7.

### *Medical Exam {EVENTTYPE = 6}:*

The Medical Exam EVENTTYPE is intended for reporting medical exam data.

| Field | Data Content | Implementation Notes |
|---|---|---|
| KEY0 | ElfHash(ExamType) | |
| KEY1 | ElfHash(ExaminedBy) | |
| KEY2 | | |
| I0 | | |
| I1 | | |
| I2 | | |
| I3 | | |
| I4 | | |
| I5 | | |
| I6 | | |
| I7 | | |
| I8 | | |
| I9 | | |
| D0 | | |
| D1 | | |
| D2 | | |
| D3 | | |
| D4 | | |
| C0 | ExaminedBy | Free text, typically the name of the Health Care Professional performing the examination. |

| | | |
|---|---|---|
| C1 | ExamType | Free text description of the exam. |
| C2 | | |
| C3 | | |
| C4 | | |
| C5 | | |
| C6 | | |
| C7 | | |
| C8 | | |
| C9 | | |

### *Medication {EVENTTYPE = 7}:*

The Medication EVENTTYPE is used for recording the administration of any medication or therapy. It is primarily intended to cover oral diabetes medications. It should not be used to describe insulin administrations, as these are described using other event types.

| Field | Data Content | Implementation Notes |
|---|---|---|
| KEY0 | ElfHash(Dosage) | |
| KEY1 | ElfHash(MedicationName) | |
| KEY2 | ElfHash(NumberOfPills) | |
| I0 | | |
| I1 | | |
| I2 | | |
| I3 | | |
| I4 | | |
| I5 | | |
| I6 | | |
| I7 | | |
| I8 | | |
| I9 | | |
| D0 | | |
| D1 | | |
| D2 | | |
| D3 | | |
| D4 | | |
| C0 | Dosage | Free text, usually indicating the strength or concentration of each unit of medicine. |
| C1 | MedicationName | Free text, usually the generic or brand name of the medication. |
| C2 | NumberOfPills | Free text, usually the number of units of medication taken at this time. |
| C3 | | |
| C4 | | |
| C5 | | |
| C6 | | |

| | | |
|---|---|---|
| C7 | | |
| C8 | | |
| C9 | | |

### *Notes {EVENTTYPE = 8}:*

The Notes EVENTTYPE uses the COMMENT field as the primary data field, but two other fields are provided, as needed.

| Field | Data Content | Implementation Notes |
|---|---|---|
| KEY0 | ElfHash(MiscValue) | |
| KEY1 | ElfHash(MiscName) | |
| KEY2 | | |
| I0 | | |
| I1 | | |
| I2 | | |
| I3 | | |
| I4 | | |
| I5 | | |
| I6 | | |
| I7 | | |
| I8 | | |
| I9 | | |
| D0 | | |
| D1 | | |
| D2 | | |
| D3 | | |
| D4 | | |
| C0 | MiscName | Free text, used for classes or types of notes or comments. (Optional) |
| C1 | MiscValue | Free text, usually not used. It can be used to represent some "value" type information associated with the Note. (Optional) |
| C2 | | |
| C3 | | |
| C4 | | |
| C5 | | |
| C6 | | |
| C7 | | |
| C8 | | |
| C9 | | |

### *State Of Health {EVENTTYPE = 9}:*

The State Of Health EVENTTYPE is used for recording current health related information.

| Field | Data Content | Implementation Notes |
|---|---|---|
| KEY0 | ElfHash(StateOfHealth) | |
| KEY1 | | |
| KEY2 | | |
| I0 | | |
| I1 | | |
| I2 | | |
| I3 | | |
| I4 | | |
| I5 | | |
| I6 | | |
| I7 | | |
| I8 | | |
| I9 | | |
| D0 | | |
| D1 | | |
| D2 | | |
| D3 | | |
| D4 | | |
| C0 | StateOfHealth | Free text, a word or phrase describing the current state of health. |
| C1 | | |
| C2 | | |
| C3 | | |
| C4 | | |
| C5 | | |
| C6 | | |
| C7 | | |
| C8 | | |
| C9 | | |

### *Ketone {EVENTTYPE = 10}:*

The Ketone EVENTTYPE is intended for use with Ketone result.

| Field | Data Content | Implementation Notes |
|---|---|---|
| KEY0 | Trunc(Reading * 1000) | |
| KEY1 | | |
| KEY2 | | |
| I0 | ControlFlag | 0 = Not a control value<br>1 = Control value. |

| | | |
|---|---|---|
| I1 | OutOfRangeFlag | -1 = Low (Negative 1)<br>0 = InRange<br>1 = Hi. |
| I2 | TemperatureFlag | 0 = not out of range<br>1 = out of temperature range |
| I3 | | |
| I4 | | |
| I5 | | |
| I6 | | |
| I7 | | |
| I8 | | |
| I9 | | |
| D0 | Reading | The ketone value, in mmol/L. |
| D1 | | |
| D2 | | |
| D3 | | |
| D4 | | |
| C0 | SampleSite | Free Text. (Optional) |
| C1 | CalCode | Free Text. (Optional) |
| C2 | DeviceStatus | Free Text. (Optional) |
| C3 | | |
| C4 | | |
| C5 | | |
| C6 | | |
| C7 | | |
| C8 | | |
| C9 | | |

### *Alarm {EVENTTYPE = 15}*

The Alarm EVENTTYPE was created specifically to support alarm notifications from the FreeStyle Navigator Continuous Glucose Monitor; however, it can be used to support any reported glucose alarms.

| Field | Data Content | Implementation Notes |
|---|---|---|
| KEY0 | AlarmType | |
| KEY1 | | |
| KEY2 | | |
| I0 | AlarmType | 0 = Hypo<br>1 = Hyper<br>2 = Impending Hypo<br>3 = Impending Hyper. |
| I1 | GlucoseValue | In mg/dL. |

| | | |
|---|---|---|
| I2 | AlarmImpendingness | Expressed in minutes (this is the "sensitivity" of the alarm, indicating how much notice the user has of an impending alarm event.)<br>**Note -**<br>This is ignored unless AlarmType = 2 or 3. |
| I3 | | |
| I4 | | |
| I5 | | |
| I6 | | |
| I7 | | |
| I8 | | |
| I9 | | |
| D0 | | |
| D1 | | |
| D2 | | |
| D3 | | |
| D4 | | |
| C0 | | |
| C1 | | |
| C2 | | |
| C3 | | |
| C4 | | |
| C5 | | |
| C6 | | |
| C7 | | |
| C8 | | |
| C9 | | |

### *Generic {EVENTTYPE = 16}*

The Generic EVENTTYPE was created specifically to support generic events generated from the FreeStyle Navigator Continuous Glucose Monitor; however, it can be used to support generic events that may be classified as an integer value.

Note: FreeStyle Navigator uses generic event types 0 through 7.

| Field | Data Content | Implementation Notes |
|---|---|---|
| KEY0 | GenericType | |
| KEY1 | | |
| KEY2 | | |
| I0 | GenericType | Any valid integer. |
| I1 | | |
| I2 | | |
| I3 | | |
| I4 | | |
| I5 | | |

| | | |
|---|---|---|
| I6 | | |
| I7 | | |
| I8 | | |
| I9 | | |
| D0 | | |
| D1 | | |
| D2 | | |
| D3 | | |
| D4 | | |
| C0 | | |
| C1 | | |
| C2 | | |
| C3 | | |
| C4 | | |
| C5 | | |
| C6 | | |
| C7 | | |
| C8 | | |
| C9 | | |

## XML Standards

The CoPilot XML import format follows conventions of XML version 1.0; however, complete compatibility with the XML standard is not guaranteed. For successful implementation of the XML import, employ the following example formats, exactly as described.

## Data Fields

In general, the XML format imports the same data fields and data records as described for TAB imports. The following rules apply to XML imports:

- All data entries must have leading and trailing quote marks.
- Empty string text type fields ("C" fields) must be explicitly entered with double quotes ("").
- Unused numeric fields ("I" and "D" fields) must be entered with zeros ("0").

## Format Structure

The following is an example of a coded XML file. This file contains two records. The first record, for example purposes, contains a meal event. The second record, for example purposes, contains a glucose event.

```
<?xml version="1.0" encoding="windows-1252" standalone="yes>
<RECORDS>
 <RECORD>
  <ROW
   DATEEVENT="38767.5347222222"
   TIMESLOT="1"
   EVENTTYPE="5"
   DEVICE_MODEL="Navigator"
   DEVICE_ID="BAAF300-80356U"
   VENDOR_EVENT_TYPE_ID="0"
   VENDOR_EVENT_ID="1073741901-12"
   KEY0="10000"
   KEY1="3289648"
   KEY2="0"
   I0="0"
   I1="0"
   I2="0"
   I3="0"
   I4="0"
   I5="0"
   I6="0"
   I7="0"
   I8="0"
   I9="0"
   D0="0"
   D1="10"
   D2="0"
   D3="0"
   D4="0"
   C0=""
   C1=""
   C2=""
   C3=""
```

```
    C4=""
    C5=""
    C6=""
    C7=""
    C8="Lunch"
    C9=""
    ISMANUAL="1"
    COMMENT=""
    DELETED="0"
   />
</RECORD>
<RECORD>
  <ROW
    DATEEVENT="38767.00625"
    TIMESLOT="7"
    EVENTTYPE="1"
    DEVICE_MODEL="Navigator"
    DEVICE_ID="BAAF300-80356U"
    VENDOR_EVENT_TYPE_ID="0"
    VENDOR_EVENT_ID="7508-12"
    KEY0="145"
    KEY1="0"
    KEY2="0"
    I0="0"
    I1="145"
    I2="0"
    I3="0"
    I4="0"
    I5="1"
    I6="3"
    I7="0"
    I8="0"
    I9="0"
    D0="0"
    D1="0"
    D2="0"
    D3="0"
```

```
  D4="0"
  C0=""
  C1=""
  C2="1"
  C3=""
  C4=""
  C5=""
  C6=""
  C7=""
  C8=""
  C9=""
  ISMANUAL="0"
  COMMENT=""
  DELETED="0"
 />
</RECORD>
</RECORDS>
```

## ElfHash

This section describes the coding implementation for the ElfHash algorithm, which is used to develop hash integer values from character strings, for use in fields KEY0, KEY1, KEY2. This implementation is in Pascal.

```
function ElfHash( const Value : string ) : Integer;
var
 i, x                : Integer;
begin
 Result := 0;
 for i := 1 to Length( Value ) do
 begin
  Result := ( Result shl 4 ) + Ord( Value[ i ] );
  x := Result and $F0000000;
  if ( x <> 0 ) then
   Result := Result xor ( x shr 24 );
  Result := Result and ( not x );
 end;
end;
```