CalConnect TC

vObject — vObject Model and vFormat Syntax

Working Draft Standard

Warning for drafts

This document is not a CalConnect Standard. It is distributed for review and comment, and is subject to change without notice and may not be referred to as a Standard. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

The Calendaring and Scheduling Consortium, Inc. 2019

:2019

The Calendaring and Scheduling Consortium, Inc.

4390 Chaffin Lane
McKinleyville
California 95519
United States of America

copyright@calconnect.org
www.calconnect.org

Contents

:2019

Foreword

vObject represents the generalized data model, and vFormat the generalized data format, of the following specifications and fully covers them:

— RFC 6350, vCard version 4.0: the VCARD component;
— RFC 5545, Internet Calendaring and Scheduling Core Object Specification (iCalendar): the VCALENDAR, VEVENT, VJOURNAL, VFREEBUSY, VTIMEZONE, VALARM, VTODO, STANDARD and DAYLIGHT components;
— RFC 7953, Calendar Availability Extensions: the VAVAILABILITY and AVAILABLE components;
— I-Ddaboo-icalendar-vpatch, iCalendar Patching: the VPATCH component; and
— alternative formats for iCalendar and vCard, including RFC 6321, xCal; RFC 7265, jCal; RFC 6351, xCard; and RFC 7095, jCard.

This work is produced by the CalConnect TC-VCARD and TC-CALENDAR committees.

The Calendaring and Scheduling Consortium (“CalConnect”) is a global non-profit organization with the aim to facilitate interoperability of collaborative technologies and tools through open standards.

CalConnect works closely with international and regional partners, of which the full list is available on our website (https://www.calconnect.org/about/liaisons-and-relationships).

The procedures used to develop this document and those intended for its further maintenance are described in the CalConnect Directives.

In particular the different approval criteria needed for the different types of CalConnect documents should be noted. This document was drafted in accordance with the editorial rules of the CalConnect Directives.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CalConnect shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be provided in the Introduction.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

This document was prepared by Technical Committee *VCARD*.

Introduction

The ubiquitous vCard IETF RFC 6350 and iCalendar IETF RFC 5545 standards are known as the "vObject" or "VCOMPONENT" family of standards. Named by the convention where the component type identifiers usually start with the letter "v", all of them use very similar, if not identical, syntax.

While the origin of these formats have a shared history, due to diverged implementations of "vObject" standards, the serialization of such formats often generate different output even when given identical content, causing interoperability concerns and the general inability to determine equivalence of vObjects for integrity concerns (2.1.2).

This document:

— defines the "vObject" data model, a generalization of the implied data models of vObject-compliant standards;
— defines the "vFormat" serialization syntax, a generalized syntax of vObject-compliant serialization formats;
— provides the "vObject Value Type" notation syntax, a method to define value schema of all properties in vObject-compliant standards; and
— describes the normalized form of the vObject data model and the normalization process for vFormat syntax.

The normalized forms and normalization methods described in this document are fully compatible with the vObject-compliant standards, including vCard 4.0 IETF RFC 6350 and iCalendar IETF RFC 5545.

This is a work product of the CalConnect TC-VCARD CalConnect TC VCARD and TC-CALENDAR CalConnect TC CALENDAR committees.

1. Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IETF RFC 2119, S. BRADNER. *Key words for use in RFCs to Indicate Requirement Levels*. 1997. RFC Publisher. https://www.rfc-editor.org/info/rfc2119.

IETF RFC 3986, T. BERNERS-LEE, R. FIELDING and L. MASINTER. *Uniform Resource Identifier (URI): Generic Syntax*. 2005. RFC Publisher. https://www.rfc-editor.org/info/rfc3986.

IETF RFC 5545, B. DESRUISSEAUX (ed.). *Internet Calendaring and Scheduling Core Object Specification (iCalendar)*. 2009. RFC Publisher. https://www.rfc-editor.org/info/rfc5545.

IETF RFC 5646, A. PHILLIPS and M. DAVIS (eds.). *Tags for Identifying Languages*. 2009. RFC Publisher. https://www.rfc-editor.org/info/rfc5646.

IETF RFC 6350, S. PERREAULT. *vCard Format Specification*. 2011. RFC Publisher. https://www.rfc-editor.org/info/rfc6350.

IETF RFC 8126, M. COTTON, B. LEIBA and T. NARTEN. *Guidelines for Writing an IANA Considerations Section in RFCs*. 2017. RFC Publisher. https://www.rfc-editor.org/info/rfc8126.

IETF RFC 8174, B. LEIBA. *Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words*. 2017. RFC Publisher. https://www.rfc-editor.org/info/rfc8174.

vObject — vObject Model and vFormat Syntax

2. Terms and definitions

For the purposes of this document, the terms and definitions given in ISO 8601-1:2019 and the following apply.

2.1.
vObject
information modelling language developed by CalConnect

Note 1 to entry: vObject was originally developed through generalization of the vCard and iCalendar models, and underpins their development.

Note 2 to entry: vFormat is the serialization format designed for the vObject information model.

2.2.
vFormat
serialization format used for vObject

2.3.
inner component
sub-component
vObject located within another vObject

2.4.
outer component
super-component
vObject that this vObject is an *inner component* (Clause 2.3) of

2.5.
client user application
CUA
vObject client implementation that interfaces with a user

3. vObject data model

The vObject data model is the generalized data model from the implied data models of vCard IETF RFC 6350 and iCalendar IETF RFC 5545.

While both vCard IETF RFC 6350 and iCalendar IETF RFC 5545 specify data formats for different purposes, the data model behind them follow an identical logical structure (using components, properties and parameters) with similar requirements.

By creating a generalized data model (“vObject”) that is compatible with both, we are able to ensure that newly developed data modification techniques for vObject would be interoperable on all other vObject-compliant models.

3.1. vObject-compliant models

The implied data models behind these formats are compliant to the vObject data model:

— vCard version 4.0 IETF RFC 6350: the VCARD component
— Internet Calendaring and Scheduling Core Object Specification (iCalendar) IETF RFC 5545, the VCALENDAR, VEVENT, VJOURNAL, VFREEBUSY, VTIMEZONE, VALARM, VTODO, STANDARD, DAYLIGHT components
— Calendar Availability Extensions IETF RFC 7953: the VAVAILABILITY, AVAILABLE components
— iCalendar Patching VPATCH: the VPATCH component

3.2. Elements

Data within a vObject is arranged through a logical hierarchy composed of the following elements:

— component;
— property;
— property parameter;
— property value;
— property parameter value; and
— property group.

The property group is optional and **MAY** not be accepted by all vObject-compliant models.

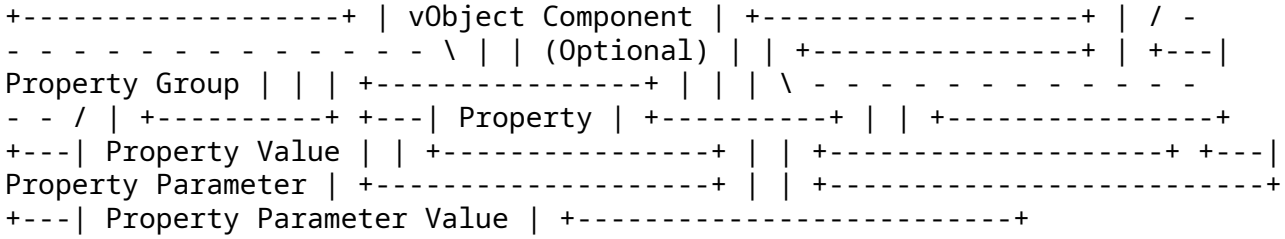

Figure 1 — vObject data model hierarchy

3.2.1. Equivalence

Two vObjects are considered identical in content if their normalized forms are textually equivalent.

3.3. Component

A vObject is based on the representation of the “component” element. All vObjects are composed of at least one component element.

A component:

— **MAY** contain vObject components;
— **MAY** contain properties; and
— **MUST** contain a uniqueness identifier property whose value is called the component’s “unique identifier”.

A component type is identified by the component name, and every component instance is distinguished by the value of its uniqueness identifier property.

A component is often used to model an object in the object-oriented sense, such as a vCard or an iCalendar object.

The order of components **MAY** represent order of the objects, which is important for example in a `VPATCH` component VPATCH, representing the patching order, which is not a commutative process.

3.3.1. Uniqueness identifying property

As a vObject component can contain inner components, it is important that those inner components can be distinguished from each other.

A vObject component’s uniqueness identifier property is a property whose value can uniquely identify the immediate component that contains it.

Every vObject component **MUST** contain a single, component uniqueness identifier property. The uniqueness identifier property can be different according to component types, and the uniqueness scope of that property **MAY** be different. At a minimum, the uniqueness identifier property value **MUST** be unique within the immediate component that contains the uniqueness identifier property.

For example:

— a `VCARD` component can be distinguished among other `VCARD` components by its `UID` property value that is globally unique;
— a `STANDARD` component of `VTIMEZONE` can be distinguished according to its `DTSTART` property within the `VTIMEZONE` component that contains it.

The list of uniqueness identifier properties for every vObject component that complies with this document can be found in the IANA registry described in Clause 11.2.

New vObject components defined according to this document **MUST** define a uniqueness identifier property for that component, and it **MUST** be registered in the said IANA registry.

3.3.2. Normalizing a component

3.3.2.1. Sorted component properties

Properties **MUST** be first normalized and before sorted, meaning that the property’s values, and its parameters and its values, have been normalized.

The sorting of component properties **MUST** be performed according to the following order:

— alphabetically by the property name. If a property spans multiple content lines, the content lines **MUST NOT** be separated after sorting.
— alphabetically by their normalized value.
— alphabetically by treating its parameters as long strings

3.3.2.2. Sorted inner components

Inner components within a vObject must be placed in a sorted order.

The sorting between components **MUST** be performed according to the following order:

— alphabetically by component type, according to component name, such as `VCALENDAR`; then
— if of the same component name, alphabetically according to its unique identifier Clause 3.3.1.

3.3.2.3. Normalized inner components

An inner component **MUST** itself be normalized, meaning that its properties and inner components **MUST** be normalized.

3.3.3. vObject property

Properties represent attributes of the vObject itself.

A property:

— **MUST** contain a value;
— **MAY** contain one or more property parameters.

vObject property value types are listed in Clause 3.3.4.

3.3.3.1. Normalized values

A normalized property **MUST** have normalized values.

3.3.3.2. Normalized parameters

A normalized property **MUST** have normalized property parameters.

Property parameters within the same property **MUST** each be normalized, and then sorted by parameter name alphabetically.

Property parameters of the same property **MUST** have unique names.

3.3.4. Property value

A vObject property **MAY** have one or more values, depending on the value type.

vObject property values are strongly typed, just like in IETF RFC 5545 and IETF RFC 6350. Basic value types accepted in vObject properties are defined in Clause 5.

vObject compliant formats **MAY** define additional value types that are not provided in this document, and **MAY** require separate validation rules, such as the “RECUR” property value type from iCalendar IETF RFC 5545.

Each property **MUST** define a default value type, and **MAY** accept alternative, defined, value types.

3.3.5. Normalization of property values

The property value generally does not require any normalization. Please consult individual normalization instructions in each value type's definition.

3.3.6. Property parameter

A property parameter is an attribute that applies on a property.

A property parameter contains:

— an identifier identifying its type;
— the value of the property parameter.

A property parameter **MAY** represent:

— information about the property; or
— information about the property value

A property **MAY** have multiple property parameters, for example, the "TYPE" of the value or a category that applies to this value.

3.3.6.1. Value lists

Certain property parameters allow multiple values. There is no defined order among property parameters in a property parameter list.

In normalized form, values in a value list **MUST** be sorted alphabetically.

3.3.7. Default property parameter values

Property parameters are allowed to have a default value.

For example, in vObject formats including IETF RFC 5545 and IETF RFC 6350, each property value has a specified data type specified as the VALUE property parameter.

3.3.8. Property parameter value

vObject property parameter values are strongly typed, just like vObject properties, IETF RFC 5545 and IETF RFC 6350. Basic value types accepted in vObject property parameter values are defined in Clause 5.

A property parameter value ***MAY** contain none, one or several property parameter values. No order is defined among a list of property parameter values.

vObject compliant formats **MAY** define additional value types that are not provided in this document, and **MAY** require separate validation rules, such as the values accepted by the FBTYPE property parameter in iCalendar IETF RFC 5545.

Each property parameter **MUST** define its accepted value type. While a property **MAY** accept multiple alternative value types, a property parameter value **MUST** only accept one value type.

3.3.9. Normalizing multiple parameter values

Property parameter values within a property parameter is considered sorted by value.

3.3.10. Property group

A property group (or just “group”) is used to group a set of properties, useful to represent cases where a group of properties are closely related to each other.

In the vCard IETF RFC 6350, the group is used to tie multiple properties into being displayed together.

Each property **MUST** only have a maximum of one group.

Groups are not ordered and group names are case-insensitive.

4. vFormat syntax

The “vFormat” format is the generalized syntax from the vCard IETF RFC 6350 and iCalendar IETF RFC 5545 formats, and is the native format for vObject serialization (“vObject native format”).

vFormat is called the “native” format for vObjects to distinguish it from alternative representations of vObject data, such as their XML (xCal, xCard) or JSON (jCal, jCard) representations.

Its syntax originated from the textual representation of the iCalendar and vCard formats, and is mostly traceable back to the original vCard and iCalendar specifications vCard 2.1.

Both of these formats are considered “instances” (or “downstream formats”) of vFormat, and fully adhere to vFormat requirements.

Parsing and modification operations that work on vFormat **SHOULD** work on all its instances, including iCalendar IETF RFC 5545 and vCard IETF RFC 6350.

4.1. ABNF format definition

The following ABNF notation defines the vFormat syntax, in accordance with IETF RFC 5234, which also defines CRLF, WSP, DQUOTE, VCHAR, ALPHA, and DIGIT.

A vObject is defined by the following notation in vFormat:

```
vobjects = 1*vobject

vobject = "BEGIN:" comp-name CRLF
          *contentline
          *vobject
          "END:" comp-name CRLF

comp-name = name

prop-name = name

prop-values = prop-value / prop-list / prop-structured

prop-value = VALUE-CHAR

prop-list = prop-value *("," prop-value)
  ; An unordered list containing multiple property values

prop-structured = prop-value *(";" prop-value)
  ; A structured list that consist of multiple property fields
  ; for multiple property values
```



```
contentline = [group "."] prop-name params ":" prop-values CRLF
  ; Folding and unfolding procedures described in Section 3.2 of
  ; [RFC6350] applies:
  ;   * When parsing a content line, folded lines *MUST* first be
  ;     unfolded accordingly.
  ;   * When generating a content line, lines longer than 75
  ;     characters *SHOULD* be folded accordingly.
  ;   * When normalizing a content line, the content line *MUST*
  ;     be folded when the line is longer than 75 characters.

group = name

params = *(";" param)

param = name "=" param-value *("," param-value)
  ; Allowed parameters depend on property name.

name = 1*(ALPHA / DIGIT / "-")

NON-ASCII = UTF8-2 / UTF8-3 / UTF8-4
  ; UTF8-{2,3,4} are defined in <<RFC3629>>
  ; TODO: generalize this to UTF-32

QSAFE-CHAR = WSP / "!" / %x23-7E / NON-ASCII
  ; Any character except CTLs, DQUOTE

SAFE-CHAR = WSP / "!" / %x23-39 / %x3C-7E / NON-ASCII
  ; Any character except CTLs, DQUOTE, ";", ":"

VALUE-CHAR = WSP / VCHAR / NON-ASCII
  ; Any textual character
```

Figure 2

4.2. Component

A component:

— begins with line of text that starts with `BEGIN:` following with its component name, ending with a line break;
— ends with a line of text that starts with `END:` following with its component name (matching with the `BEGIN:` line), ending with a line break.

The vCard IETF RFC 6350 and iCalendar IETF RFC 5545 data formats both conform to vFormat, and their syntaxes are considered to be restricted instances of the vObject syntax.

4.2.1. Component name in uppercase

The component name of a vObject **MUST** be uppercased, for both the `BEGIN:` and `END:` content lines.

Example:

```
BEGIN:vCard
```

Should be normalized to:

```
BEGIN:VCARD
```

NOTE In vCard 4.0 IETF RFC 6350, only capital letters are allowed for component names.

4.2.2. Order of inner components and properties

Properties **MUST** be placed before inner components are listed.

4.2.3. Maintain validity

Certain vObject formats places certain restrictions or requirements on property line locations. Normalization procedures **MUST NOT** affect the validity of the normalized vObject.

For example, in the VCARD component IETF RFC 6350, the “VERSION” property line is **REQUIRED** to be placed immediately below the “BEGIN” line.

In this case, when normalizing the VCARD component, the common normalization procedure **MUST** be first applied, and the “VERSION” property line **MUST** be restored to the valid location as required by its specifications IETF RFC 6350.

4.3. Property

A property can be represented by one content line (a line that ends with a line break), but can also be “folded” (3.1) to use multiple lines.

A property begins with the property name (e.g., TEL), followed by a COLON delimiter and the property’s value.

4.3.1. Uppercased property name

The property name **MUST** be normalized to uppercase letters.

4.3.2. Normalized parameters

The last property parameter of a property **MUST NOT** have a trailing SEMICOLON.

4.3.3. Wrapped content line

When exporting a normalized property content line, it **MUST** be folded at the character limit when it exceeds 75 characters. Each folded line **MUST** be delimited by the character sequence of a line break and a single white space (CRLF, SPACE (U+0020)). This rule only applies to normalized output.

For example, the original form:

```
NOTE:This is a very long description on a long line that exceeds 75 characters.
```

Figure 3

When exported to normalized output **MUST** give out:

```
NOTE:This is a very long description on a long line that exceeds 75 charac
 ters.
```

Figure 4

4.4. Property value

The property’s values are defined as the content after the property name and COLON delimiter, until the end of the unfolded content line.

If a property accepts multiple values, the definition of delimitation is defined in Clause 5.

vObject compliant formats that defines additional value types **MAY** require separate validation rules on top of vFormat syntax.

If the property value type of a property value is not the default value type, the `VALUE` parameter **MUST** be present to specify the type of the property value.

vFormat representation of different value types are provided in Clause 5.

4.4.1. Normalizing property values

The property value generally does not require any normalization.

4.5. Property parameter

Property parameters exist between the property name and the COLON delimiter in a property line.

Each property parameter in a list of property parameters **MUST** be separated by a SEMICOLON character.

The property parameter name and the property parameter value is separated with an equal sign (“=”).

4.5.1. Multiple property parameters

If the property accepts multiple property parameters values, they **MUST** be separated by a SEMICOLON character as a list.

4.5.2. Expanded form of property parameter value list

When there are multiple instances of a property parameter on the same property, such as in “TYPE=home;TYPE=work”, it is considered equivalent to “TYPE=home\,work”.

4.5.3. Uppercased property parameter names

The property parameter name **MUST** be normalized into uppercase letters in form of a Unicode string (7).

Property parameter names (together with their values) **MUST** be sorted alphabetically.

Example:

```
TEL;VALUE=uri;type=home:tel:+1-888-888-8888
```

The normalized form is:

```
TEL;TYPE="home";VALUE="uri":tel:+1-888-888-8888
```

4.5.4. Join identical property parameter names

If a property parameter occurs more than once within a property, the property parameter is considered to contain a list of property parameter values joined by the parameter separator.

Such instances of property parameters with identical names **MUST** be joined into one instance with its value a sorted list of the property parameter values.

For example, the vCard `TEL` property’s `TYPE` parameter IETF RFC 6350 describes that `TYPE=home`, `work` and `TYPE=work;TYPE=home` are considered equivalent.

:2019

Example:

```
TEL;TYPE=home;Type=work;VALUE=uri:tel:+1-888-888-8888
```

The normalized form is:

```
TEL;TYPE="home","work";VALUE=uri:tel:+1-888-888-8888
```

4.5.5. Express default property value types

In vObject formats including IETF RFC 5545 and IETF RFC 6350, each property value has a specified data type either as specified by property definition or optionally assigned.

When normalizing a property, the property data value type **MUST** always be specified. If the value type is not explicitly specified, it **MUST** be filled in according to the vObject format.

Example:

```
TEL:+1-888-888-8888
```

The normalized form is:

```
TEL;VALUE="text":+1-888-888-8888
```

4.6. Property parameter value

While a property parameter value may accept any vObject value type, the serialization rules of a vFormat property value and a vFormat property parameter value are different, due to further limitations of allowed characters in property parameter values.

4.6.1. Format definition

```
param-value          = param-single-value *("," param-single-value)
param-single-value   = paramtext / quoted-string

paramtext     = *SAFE-CHAR
quoted-string = DQUOTE *QSAFE-CHAR DQUOTE

QSAFE-CHAR    = WSP / %x21 / %x23-7E / NON-US-ASCII
    ; Any character except CONTROL and DQUOTE

SAFE-CHAR     = WSP / %x21 / %x23-2B / %x2D-39 / %x3C-7E
             / NON-US-ASCII
    ; Any character except CONTROL, DQUOTE, ";", ":", ","

NON-US-ASCII  = UTF8-2 / UTF8-3 / UTF8-4
     ; UTF8-2, UTF8-3, and UTF8-4 are defined in [RFC3629]

CONTROL       = %x00-08 / %x0A-1F / %x7F
     ; All the controls except HTAB
```

Figure 5

4.6.2. Description

In vFormat, if a property parameter accepts multiple values, these value elements **MUST** be separated by a COMMA (U+002C).

The DQUOTE character is used as a delimiter for parameter values that contain restricted characters or URI text.

Property parameter values that contain the COLON (U+003A), SEMICOLON (U+003B) (such as the `LIST` and `MAP` value types), or COMMA (U+002C) character separators **MUST** be specified as quoted-string text values.

Property parameter values that contain the DQUOTE (U+0022) character **MUST** be escaped and specified as quoted-string text values.

An intentional line break **MUST** be represented by the sequence of "\n" or "\N" (BACKSLASH followed by a LATIN SMALL LETTER N (U+006E) or a LATIN CAPITAL LETTER N (U+004E)).

Property parameter values that are not in quoted-strings are case-insensitive.

4.6.3. Example

The value `cid:part1.0001@example.org` in the parameter `ALTREP` within a `DESCRIPTION` property in iCalendar can be specified like this:

```
DESCRIPTION;ALTREP="cid:part1.0001@example.org":The Fall'98 Wild
 Wizards Conference - - Las Vegas\, NV\, USA
```

Figure 6

4.6.4. Normalization of line breaks

In vFormat, vObject property parameter values **MUST** convert all line breaks ("\n" or "\N") into the character sequence "\n" (BACKSLASH followed by a LATIN SMALL LETTER N (U+006E)).

If a value is specified as `paramtext` (i.e., not inside a quoted-string), the value **SHOULD** be down-cased.

4.6.5. Normalizing parameter values via DQOUTE wrapping

vFormat property parameter values **SHOULD** be individually wrapped with the DQUOTE characters.

This is an example application of the rule from IETF RFC 6350:

```
TEL;TYPE=home,work;VALUE=uri:tel:+1-888-888-8888
```

The normalized vFormat output is:

```
TEL;TYPE="home","work";VALUE="uri":tel:+1-888-888-8888
```

4.7. Property group

The syntax of a property group is defined in Clause 4.1.

Property groups **MUST NOT** be removed during normalization. This is contrary to IETF RFC 6350 that allows stripping off groups.

5. vObject value types and notation syntax

vObject value types are identically defined for both:

— vObject property values; and

— vObject property parameter values.

5.1. Value type notation

The vObject value type notation is used for defining the accepted values within a vObject property or parameter values. It fully covers all complete and exhaustive amongst all vObject-compliant standards.

This notation syntax allows a vObject specification to define complex value types by using one or more value primitives defined in the sections below.

The purpose of this syntax is to provide a mechanism to all vObject value definitions, such that any new vObject mechanism (such as, a method that can be applied to any vObject) can ensure uniform applicability on vObject values.

Value type mappings provided in Clause 12, Clause 14, and Clause 13 are denoted using the vObject value type syntax.

Implementation differences within Clause 4 of the same value type are described in Clause 4.4 and Clause 4.6.

5.2. Meta value types

Meta value types are used in conjunction with basic value types (Clause 5.3).

5.2.1. FIELDSET

Some properties and parameters require values defined in terms of multiple parts.

This construct of multiple structured values is called a “FIELDSET”. Each value in FIELDSET **MUST** have the same value type as defined.

5.2.1.1. Value type notation

When used to describe a value type, the `FIELDSET(field-1-value-type, ...)` notation is defined as a structure of fields separated by the SEMICOLON character, where each of its fields is of value type `field-i-value-type`, where `i` represents the index of the specific field.

5.2.1.2. Example 1

The “CLIENTPIDMAP” property of IETF RFC 6350 takes a tuple of “INTEGER” and “URI”.

The notation in vObject given for its value type would be this indicating that the first value is an INTEGER, while the latter value is a URI:

```
FIELDSET(INTEGER, URI)
```

Figure 7

5.2.1.3. Example 2

The “N” property of IETF RFC 6350 defines its value of having 5 values at once, and each of these values are a LIST of TEXT.

The notation in vObject given for its value type would be this indicating that there are 5 fields in this FIELDSET, and each value element of it **MUST** be a LIST of TEXT elements:


```
FIELDSET(5\*LIST(TEXT))
```

Figure 8

5.2.1.4. Normalizing a FIELDSET

When normalizing a FIELDSET, each value **MUST** have been normalized, but the order of FIELDSET elements **MUST NOT** be rearranged.

5.2.2. LIST

Properties and parameters **MAY** specify its value to be an unordered list of values. There is no significance to the order of values in a list.

This construct is called the “LIST”. Each value in the LIST **MUST** have the same value type.

5.2.2.1. Value type notation

The `LIST(value-type)` notation is used to describe this value type, of a list of elements, where each of its elements is of value type `value-type`.

5.2.2.2. Example 1

The “NICKNAME” property of IETF RFC 6350 defines its value to be an unordered list of TEXT.

In vObject notation its value type is defined to be:

```
LIST(TEXT)
```

Figure 9

5.2.2.3. Example 2

The “RECUR” property of IETF RFC 5545 defines its value to be an unordered list of ASSIGN.

In vObject notation its value type is defined to be:

```
LIST(KEYVALUE, ";")
```

Figure 10

5.2.2.4. Normalizing a LIST

When normalizing a LIST, each value of it **MUST** be normalized, and the values **MUST** be sorted alphabetically.

For example, values of IETF RFC 5545 RESOURCES, FREEBUSY, EXDATE, RDATE, CATEGORIES, **MUST** be sorted alphabetically when normalized.

5.2.3. MAP

A `MAP` serves the function of a key-value table. It is realized using the `LIST` value type with values of the value type `KEYVALUE`.

Each value in the MAP **MUST** use the `KEYVALUE` value type.

There is no inherent order of the values within a MAP. Values within its key value pairs **MAY** be of different value types as defined.

5.2.3.1. Value type notation

This value type is described using the `MAP(kv_1, kv_2, …)` notation, where each `kv_i` represents a property of the value type KEYVALUE.

5.2.3.2. Example

The Recurrence Rule property (“RECUR”) of IETF RFC 5545 defines its value to be a MAP.

In vObject notation its value type is defined to be:

```
MAP(
  KEYVALUE(FREQ, TEXT),
  KEYVALUE(UNTIL, ISO-DATE-COMPLETE / ISO-DATE-TIME-BASIC),
  KEYVALUE(COUNT, INTEGER),
  KEYVALUE(INTERVAL, INTEGER),
  KEYVALUE(BYSECOND, LIST(INTEGER)),
  KEYVALUE(BYMINUTE, LIST(INTEGER)),
  KEYVALUE(BYHOUR, LIST(INTEGER)),
  KEYVALUE(BYDAY, LIST(INTEGER)),
  KEYVALUE(BYMONTHDAY, LIST(INTEGER)),
  KEYVALUE(BYYEARDAY, LIST(INTEGER)),
  KEYVALUE(BYWEEKNO, LIST(INTEGER)),
  KEYVALUE(BYMONTH, LIST(INTEGER)),
  KEYVALUE(BYSETPOS, INTEGER),
  KEYVALUE(WKST, TEXT)
)
```

Figure 11

5.2.3.3. Normalizing a MAP

When normalizing a MAP, each value of it **MUST** be normalized, and the values **MUST** be sorted alphabetically according to its key.

5.3. Basic value types

5.3.1. TEXT

This corresponds to the TEXT value type in 4.1 and 3.3.11.

5.3.1.1. Value type notation

TEXT

5.3.1.2. Purpose

This value type defines values that contain free-form, human-readable text.

5.3.1.3. Format definition

```
text = VALUE-CHAR
```

Figure 12

5.3.1.4. Associated parameters

— Language in which the text is represented can be controlled by the “LANGUAGE” property parameter.

5.3.1.5. Example

This multiple line value is a valid value for the NOTE property of vCard:

```
TC VCARD
The Calendaring And Scheduling Consortium
July 20, 2017
```

Figure 13

5.3.2. URI

This corresponds to the URI value type in 4.2 and 3.3.13.

5.3.2.1. Value type notation

URI

5.3.2.2. Purpose

This value type defines values that are represented by data referenced by a uniform resource identifier (URI), the value is what the URI points to, not the URI itself.

5.3.2.3. Format definition

```
uri = <As defined in Section 3 of [RFC3986]>
```

Figure 14

NOTE `uri` is defined in IETF RFC 3986.

5.3.2.4. Description

When a property parameter value is a URI value type, the URI **MUST** be specified as a quoted-string value.

5.3.2.5. Example

This following values for the PHOTO property of vCard are valid.

Example 1:

```
http://www.example.com/pub/photos/jqpublic.gif
```

Figure 15

Example 2:

```

 AQEEBQAwdzELMAkGA1UEBhMCVVMxLDAqBgNVBAoTI05ldHNjYXBlIENvbW11bm
 ljYXRpb25zIENvcnBvcmF0aW9uMRwwGgYDVQQLExNJbmZvcm1hdGlvbiBTeXN0
```



```
<...remainder of base64-encoded data...>
```

Figure 16

5.3.2.6. Normalization

No normalization procedures are needed.

5.3.3. BOOLEAN

This corresponds to the BOOLEAN value types in 4.4 and 3.3.2.

5.3.3.1. Value type notation

BOOLEAN

5.3.3.2. Purpose

This value type is used to identify properties that contain either a “TRUE” or “FALSE” Boolean value.

5.3.3.3. Format definition

```
boolean = "TRUE" / "FALSE"
```

Figure 17

5.3.3.4. Description

Parsing of “TRUE” and “FALSE” values **SHOULD** be case-insensitive, but a writer of such value **MUST** only output of this value type in uppercase.

5.3.3.5. Examples

— TRUE
— false
— True
— FaLSe

5.3.3.6. Normalization

Values of the BOOLEAN data type **MUST** be normalized to uppercase, i.e., the values “TRUE” and “FALSE”.

5.3.4. INTEGER

The INTEGER-64 and INTEGER-32 value types corresponds to the INTEGER value types in 4.5 and in 3.3.8 respectively.

5.3.4.1. Value type notation

INTEGER

(INTEGER-32 for storing 32-bit integer, INTEGER-64 for storing 64-bit integer)

5.3.4.2. Purpose

Representation of a signed integer value.

5.3.4.3. Format definition

```
integer = (["+"] / "-") 1*DIGIT
```

Figure 18

5.3.4.4. Description

If a preceding sign is not specified, the value is assumed positive “”. While the format accepts the optional “” PLUS sign, a writer that conforms to this document **SHOULD** not write the “+” sign for clarity reasons.

The valid ranges for INTEGER-32 and INTEGER-64 are:

— INTEGER-32: -2147483648 (-2^31) to 2147483647 (2^31 — 1)
— INTEGER-64: -9223372036854775807 (-2^63) to 9223372036854775808 (2^63 — 1)

5.3.4.5. Examples

— 1234567890
— -1234567890
— +1234567890
— 432109876

5.3.4.6. Normalization

A positive integer when normalized **MUST** not have the optional “+” sign.

5.3.5. FLOAT

This corresponds to the FLOAT value types in 3.3.7 and 4.6.

5.3.5.1. Value type notation

FLOAT

5.3.5.2. Purpose

Representation of a real-number value.

5.3.5.3. Format definition

```
float = (["+"] / "-") 1*DIGIT ["." 1*DIGIT]
```

Figure 19

5.3.5.4. Description

If a preceding sign is not specified, the value is assumed positive “+”.

Implementations **MUST** support a precision equal or better than that of the IEEE “binary64” format IEEE 754™-2008.

Scientific notation is disallowed.

5.3.5.5. Examples

— 20.30
— 1000000.0000001
— 1.333
— -3.14

5.3.5.6. Normalization

No normalization procedures are needed.

Trailing zeros, such as `100.10000` **MUST** be kept as it indicates accuracy of the number.

5.3.6. LANGUAGE-TAG

This corresponds to the LANGUAGE-TAG value type in 4.8.

5.3.6.1. Value type notation

LANGUAGE-TAG

5.3.6.2. Purpose

Representing a language tag, as defined in IETF RFC 5646.

5.3.6.3. Format definition

Defined in IETF RFC 5646.

5.3.6.4. Description

A single language tag

5.3.6.5. Examples

— de
— en-US
— sr-Cyrl
— zh-yue-HK

5.3.6.6. Normalization

The normalization procedure of the LANGUAGE-TAG data type follows the procedure described in 2.1.1.

— language codes **MUST** be written in lowercase ('mn' Mongolian)
— script codes **MUST** be in lowercase when the initial letter capitalized ('Cyrl' Cyrillic)
— country codes **MUST** be capitalized ('MN' Mongolia)

As the language tag is comprised of a mixture of these components, IETF RFC 5646 provides a rule that applies this procedure across all language tags:

— All subtags, including extension and private use subtags, **MUST** use lowercase letters.
— Except: two-letter subtags that neither appear at the start of the tag nor occur after singletons **MUST** be in uppercase ("en-CA-x-ca" or "sgn-BE-FR").

— Except: four-letter subtags that neither appear at the start of the tag nor occur after singletons **MUST** be in titlecase (“az-Latn-x-latn”).

5.3.7. Binary

This corresponds to the BINARY value type in 3.3.1.

5.3.7.1. Value type notation

BINARY

5.3.7.2. Purpose

This value type defines values that contain inline binary data encoded in characters. For example, an inline “ATTACH” property of an iCalendar object or an inline “PHOTO” property image inside a vCard object.

5.3.7.3. Format definition

```
binary = *(4b-char) [b-end]
  ; A "BASE64" encoded character string, as defined by [RFC4648].

b-end  = (2b-char "==") / (3b-char "=")

b-char = ALPHA / DIGIT / "+" / "/"
```

Figure 20

NOTE The “BASE64” encoded character string is defined in IETF RFC 4648.

5.3.7.4. Description

Property values with this value type **MUST** specify the parameter “ENCODING” with parameter value “BASE64”, and the inline binary data **MUST** be character encoded using the “BASE64” encoding method defined in IETF RFC 2045.

5.3.7.5. Example

This value for the NOTE value of vCard:

The following is an example of a “BASE64” encoded binary value data folded to 72 characters long:

```
AAABAAEAEBAQAAEABAAoAQAAFgAAACgAAAAQAAAAIAAAAAEABAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAACAAAAAgIAAAICAgADAwMAA////AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAMwAAAAAABNEMQAAAAAAAk
QgAAAAAAJEREQgAAACECQ0QgEgAAQxQzM0E0AABERCRCREQAADRDJEJEQwAAAhA0QwEQAAAA
AEREAAAAAAAAREQAAAAAAAAkQgAAAAAAAAMgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

Figure 21

5.3.8. KEYVALUE

5.3.8.1. Value type notation

```
KEYVALUE(key, value-type)
```


5.3.8.2. Purpose

Representation of a key-value pair: a key `key` linked to a value of value type `value-type`.

5.3.8.3. Format definition

```
assign-key   = *(TSAFE-CHAR)
assign-value = prop-values
```

Figure 22

5.3.8.4. Description

This value type is a core component of the `MAP` value type.

If the `KEYVALUE` value is accepted within a list, the `key` value must be unique amongst the list.

5.3.8.5. Examples

— key: `FREQ`; value: `WEEKLY`
— key: `BYHOUR`; value: `3,6,9`
— key: `BYWEEKNO`; value: `MO,TU,WE,TH,FR,SA`

5.3.8.6. Normalization

Its value **MUST** be normalized according to the `value-type` of that value.

5.4. Date and time value types

These date and time related value types are based on ISO 8601:2004 and ISO 8601:2000.

5.4.1. ISO-DATE-COMPLETE

This corresponds to the DATE value type in 3.3.4.

5.4.1.1. Value type notation

ISO-DATE-COMPLETE

5.4.1.2. Purpose

Representation of a complete calendar date defined in ISO 8601:2004.

5.4.1.3. Format definition

```
iso-date              = iso-date-value
iso-date-value        = iso-date-fullyear iso-date-month iso-date-mday
iso-date-fullyear     = 4DIGIT
iso-date-month        = 2DIGIT   ;01-12
iso-date-mday         = 2DIGIT   ;01-28, 01-29, 01-30, 01-31
                                 ;based on month/year
```

Figure 23

5.4.1.4. Description

This value format is based on the “basic format” calendar date (specified in 4.1.2.2 “Complete representations”).

The value **MUST** be represented textually as “YYYYMMDD”, with its components “YYYY” a four-digit year, “MM” a two-digit month, and “DD” a two-digit day of the month as described in the definition.

5.4.1.5. Example

The following represents July 1, 1997:

— 19970701

5.4.1.6. Normalization

No normalization procedures are needed.

5.4.2. ISO-DATE-FLEX

Representation of a calendar date ISO 8601:2004 that does not require complete representation.

This corresponds to the DATE value type in 4.3.1.

5.4.2.1. Value type notation

ISO-DATE-FLEX

5.4.2.2. Purpose

This value type defines a calendar date format that allows entry of a complete calendar date ISO 8601:2004, a reduced accuracy date ISO 8601:2004 and a truncated date ISO 8601:2004.

5.4.2.3. Format definition

```
iso-date-flex   = iso-date /
                  iso-date-reduced /
                  iso-date-truncated

iso-date-reduced = iso-date-fullyear / iso-date-year-month
iso-date-year-month = iso-date-fullyear "-" iso-date-month

iso-date-truncated =  iso-date-truncated-month-day /
                      iso-date-truncated-month-only /
                      iso-date-truncated-day-only

iso-date-truncated-month-day  = "--" iso-date-month iso-date-mday
iso-date-truncated-month-only =  "--" iso-date-month
iso-date-truncated-day-only   =  "---" iso-date-mday
```

Figure 24

5.4.2.4. Description

This value format accepts:

— a complete calendar date, specified in 4.1.2.2 “Complete representations”,

— a reduced accuracy calendar date, specified in 4.1.2.3 “Representations with reduced accuracy”, and
— a truncated representation calendar date, specified in 5.2.1.3 “Truncated representations”.

The value can be represented in these ways:

— “YYYYMMDD” Complete representation basic format, specified in 4.1.2.2.
— “YYYY-MM” Reduced accuracy representation, specified in 4.1.2.3 a).
— “YYYY” Reduced accuracy representation, specified in 4.1.2.3 b).
— “—MMDD” Truncated representation for a specific day of a month in the implied year, basic format, specified in 5.2.1.3 d).
— “—MM” Truncated representation for a specific month in the implied year, basic format, specified in 5.2.1.3 e).
— “—-DD” Truncated representation for a specific day in the implied month, basic format, specified in 5.2.1.3 f).

Example:

— 20170712
— 2017-07
— 2017
— —0712
— —07
— —-12

5.4.2.5. Normalization

No normalization procedures are needed.

5.4.3. ISO-TIME-COMPLETE

This corresponds to the “time” portion of the TIMESTAMP value type in 4.3.5.

5.4.3.1. Value type notation

ISO-TIME-COMPLETE

5.4.3.2. Purpose

Representation of a complete time of day with a UTC offset ISO 8601:2004.

5.4.3.3. Format definition

```
iso-time = time-hour time-minute time-second
           [iso-time-utc / iso-utc-offset]

iso-time-hour    = 2DIGIT        ;00-23
iso-time-minute  = 2DIGIT        ;00-59
iso-time-second  = 2DIGIT        ;00-60
;The "60" value is used to account for positive "leap" seconds.

iso-time-utc     = "Z"
```

Figure 25

5.4.3.4. Description

This value format accepts a time of day value specified as:

— “hhmmss”, the basic format of 4.2.2.2 “Complete representations”.
— “hhmmssZ”, the first basic format of 4.2.4 “UTC of day”.
— “hhmmss±hhmm”, “hhmmss±hh”, the basic formats of 4.2.5.2 “Local time and the difference from UTC”

The components mean: “hh” a two-digit, 24-hour of the day (00-23), “mm” a two-digit minute in the hour (00-59), and “ss” a two-digit second in the minute (00-60).

The seconds value of 60 **MUST** only be used to account for positive “leap” seconds. Fractions of a second are not supported by this format.

This value indicates “local time” as specified in 2.1.16. To indicate UTC time, a “Z” character **MUST** be appended to the basic format as described in 4.2.4 “UTC of day”. To indicate a UTC offset, the “utc-offset” section **MUST** be specified in accordance with 4.2.5.2.

The value of “hhmmssZ” **MUST** be used instead of the equivalent “hhmmss+0000” or “hhmmss-0000”.

Example:

— 140000
— 140000Z
— 140000-05
— 140000-0500

5.4.3.5. Normalization

No normalization procedures are needed.

5.4.4. ISO-TIME-BASIC

This corresponds to the TIME value type in 3.3.12.

5.4.4.1. Value type notation

ISO-TIME-BASIC

5.4.4.2. Purpose

Representation of a complete time of day disallowing a UTC offset ISO 8601:2004.

5.4.4.3. Format definition

```
iso-time-basic = iso-time-hour iso-time-minute iso-time-second
                 [iso-time-utc]
```

Figure 26

5.4.4.4. Description

This value format is similar to “TIME” except it disallows the additional UTC offset, (the basic formats of 4.2.5.2 “Local time and the difference from UTC”).

This value format accepts a time of day value specified as:

— “hhmmss”, the basic format of 4.2.2.2 “Complete representations”.
— “hhmmssZ”, the first basic format of 4.2.4 “UTC of day”.

The seconds value of 60 **MUST** only be used to account for positive “leap” seconds. Fractions of a second are not supported by this format.

This value indicates “local time” as specified in 2.1.16. To indicate UTC time, a “Z” character **MUST** be appended to the basic format as described in 4.2.4 “UTC of day”.

Example:

— 232050
— 232050Z

5.4.4.5. Normalization

No normalization procedures are needed.

5.4.5. ISO-TIME-FLEX

This corresponds to the TIME value type in 4.3.2.

5.4.5.1. Value type notation

ISO-TIME-FLEX

5.4.5.2. Purpose

This value type defines a time of day format that allows a entry of a complete time of day ISO 8601:2004, a reduced accuracy date ISO 8601:2004 and a truncated date representation ISO 8601:2000.

5.4.5.3. Format definition

```
iso-time-flex = iso-time /
                iso-time-reduced /
                iso-time-truncated

iso-time-zone = iso-time-utc / iso-time-utc-offset

iso-time-reduced =  iso-time-reduced-hour-minute /
                    iso-time-hour

iso-time-reduced-hour-minute = iso-time-hour iso-time-minute

iso-time-truncated =  iso-time-truncated-minute-second /
                      iso-time-truncated-minute-only /
                      iso-time-truncated-second-only
iso-time-truncated-minute-second = "-" iso-time-minute iso-time-second
iso-time-truncated-minute-only = "-" iso-time-minute
iso-time-truncated-second-only = "--" iso-time-second
```

Figure 27

5.4.5.4. Description

This value format accepts:

— a complete time of day, specified in 4.2.2.2 “Complete representations”,
— a reduced accuracy time of day, specified in 4.2.2.3 “Representations with reduced accuracy”,
— and a truncated representation time of day, specified in 5.3.1.4 “Truncated representations”.

The value can be represented in these ways:

— “hhmmss” Complete representation basic format, specified in 4.2.2.2.
— “hhmm” Reduced accuracy representation basic format for a specific hour and minute, specified in 4.2.2.3 a).
— “hh” Reduced accuracy representation basic format for a specific hour, specified in 4.2.2.3 b).
— “-mmss” Truncated representation for a specific minute and second of the implied hour, specified in 5.3.1.4 a).
— “-mm” Truncated representation for a specific minute of the implied hour, specified in 5.3.1.4 b).
— “—ss” Truncated representation for a specific second of the implied minute, specified in 5.3.1.4 c).

The seconds value of 60 **MUST** only be used to account for positive “leap” seconds. Fractions of a second are not supported by this format.

This value indicates “local time” as specified in 2.1.16. To indicate UTC time, a “Z” character **MUST** be appended to the basic format as described in 4.2.4 “UTC of day”.

This format requires the midnight hour to be represented by “00” (4.2.3 a)), not “24” (4.2.3 b)).

This format supports the specification of UTC offsets for the complete representation basic format (defined in 4.2.5.2 basic format), in the form of “hhmmss±HHMM”. “HHMM” is the hour and minute of UTC offset, defined in 4.2.5.1 basic format.

Example:

— 102200
— 1022
— 10
— -2200
— —00
— 102200Z
— 102200+0800

5.4.5.5. Normalization

No normalization procedures are needed.

5.4.6. ISO-UTC-OFFSET

This corresponds to the UTC-OFFSET value type in 4.7.

5.4.6.1. Value type notation

ISO-UTC-OFFSET

5.4.6.2. Purpose

Representation of a UTC offset as described in ISO 8601:2004.

5.4.6.3. Format definition

```
sign = "+" / "-"
iso-utc-offset = sign iso-time-hour [iso-time-minute]
```

Figure 28

Description:

The value can be represented in two ways:

— “±hhmm” specified in 4.2.5.1 “Difference between local time and UTC of day”, the first basic format.
— “±hh” specified in 4.2.5.1 “Difference between local time and UTC of day”, the second basic format.

The PLUS SIGN character MUST be specified for positive UTC offsets (i.e., ahead of UTC). The HYPHEN-MINUS character MUST be specified for negative UTC offsets (i.e., behind of UTC).

The value of “-00” and “-0000” are not allowed. The time-minute, if present, MUST NOT be 60; if absent, it defaults to zero.

5.4.6.4. Example

The following UTC offsets are given for standard time for New York (five hours behind UTC) and Geneva (one hour ahead of UTC):

— -05
— -0500
— +01
— +0100

5.4.6.5. Normalization

No normalization procedures are needed.

5.4.7. CAL-UTC-OFFSET

This corresponds to the UTC-OFFSET value type in 3.3.14.

5.4.7.1. Value type notation

CAL-UTC-OFFSET

5.4.7.2. Purpose

Representation of a UTC offset as described in IETF RFC 5545.

5.4.7.3. Format definition

```
cal-utc-offset = sign iso-time-hour iso-time-minute [iso-time-second]
```

Figure 29

5.4.7.4. Description:

The value can be represented in two ways:

— “±hhmm” specified in 4.2.5.1 “Difference between local time and UTC of day”, the first basic format.
— “±hhmmss” which is unique to this value type.

The PLUS SIGN character MUST be specified for positive UTC offsets (i.e., ahead of UTC). The HYPHEN-MINUS character MUST be specified for negative UTC offsets (i.e., behind of UTC).

The value of “-0000” and “-000000” are not allowed. The time-second, if present, MUST NOT be 60; if absent, it defaults to zero.

5.4.7.5. Example

The following UTC offsets are given for standard time for New York (five hours behind UTC) and Geneva (one hour ahead of UTC):

— -0500
— -050000
— +0100
— +010000

5.4.7.6. Normalization

No normalization procedures are needed.

5.4.8. ISO-DATE-TIME-COMPLETE

This corresponds to the TIMESTAMP value type in 4.3.5.

5.4.8.1. Value type notation

ISO-DATE-TIME-COMPLETE

5.4.8.2. Purpose

A complete date and time of day combination as specified in 4.3.2

5.4.8.3. Format definition

```
iso-date-time  = iso-date "T" iso-time
```

Figure 30

5.4.8.4. Description

This value format accepts a complete date and time of day representation, specified in 4.3.2 “Complete representations”.

The value can be represented in these ways:

— “YYYYMMDDThhmmss” 4.3.2 Complete representation basic format, first entry.
— “YYYYMMDDThhmmssZ” 4.3.2 Complete representation basic format, second entry.
— “YYYYMMDDThhmmss±hhmm” 4.3.2 Complete representation basic format, third entry.
— “YYYYMMDDThhmmss±hh” 4.3.2 Complete representation basic format, fourth entry.

5.4.8.5. Examples

— 19850412T101530
— 19850412T101530Z
— 19850412T101530+0400
— 19850412T101530+04

5.4.8.6. Normalization

No normalization procedures are needed.

5.4.9. ISO-DATE-TIME-BASIC

This corresponds to the DATE-TIME value type in 3.3.5.

5.4.9.1. Value type notation

ISO-DATE-TIME-BASIC

5.4.9.2. Purpose

A date and time of day combination without non-UTC timezone as specified in 4.3.2

5.4.9.3. Format definition

```
iso-date-time-no-zone  = iso-date "T" iso-time-basic
```

Figure 31

5.4.9.4. Description

This value format accepts a complete date and time of day representation, specified in 4.3.2 “Complete representations”, identical with ISO-DATE-TIME-COMPLETE, except that the “utc-offset” portion is disallowed.

The value can be represented in these ways:

— “YYYYMMDDThhmmss” 4.3.2 Complete representation basic format, first entry.
— “YYYYMMDDThhmmssZ” 4.3.2 Complete representation basic format, second entry.

Due to the lack of “utc-offset”, properties that use this value type **SHOULD** handle time zone information with other methods such as in property parameters, such as using the “TZID” property parameter defined in IETF RFC 5545.

5.4.9.5. Examples

— 19980118T230000
— 19980118T230000Z

5.4.9.6. Normalization

No normalization procedures are needed.

5.4.10. ISO-DATE-TIME-FLEX

This corresponds to the DATE-TIME value type in 4.3.3.

5.4.10.1. Value type notation

DATE-TIME-FLEX

5.4.10.2. Purpose

This value type defines a date and time of day combination as specified in 4.3 and 5.4.2 c).

5.4.10.3. Format definition

```
iso-date-time-flex = iso-date-time-flex-date "T" iso-date-time-flex-time
```



```
iso-date-time-flex-date = iso-date / iso-date-truncated
iso-date-time-flex-time = iso-time / iso-time-reduced
```

Figure 32

5.4.10.4. Description

This value format allows for the:

— truncation of the date portion and
— the reduced accuracy of the time portion
— according to the requirements of 5.4.2 “Representations other than complete” part c).

5.4.10.5. Examples

— 19961022T150236
— —1022T1502
— —-22T15

5.4.10.6. Normalization

No normalization procedures are needed.

5.4.11. ISO-DATE-AND-OR-TIME

This corresponds to the DATE-AND-OR-TIME value type in 4.3.4.

5.4.11.1. Value type notation

ISO-DATE-AND-OR-TIME

5.4.11.2. Purpose

Representation of a ISO-DATE-FLEX, ISO-TIME-FLEX or an ISO-DATE-TIME-FLEX value.

5.4.11.3. Format definition

```
iso-date-and-or-time = iso-date-flex /
                       "T" iso-time-flex /
                       iso-date-time-flex
```

Figure 33

5.4.11.4. Description

This value format accepts values of ISO-DATE-FLEX, ISO-TIME-FLEX and ISO-DATE-TIME-FLEX.

A stand-alone ISO-TIME value **MUST** be preceded by a “T” for unambiguous interpretation.

5.4.11.5. Example

— 19961022T140000
— —1022T1400
— —-22T14
— 19850412
— 1985-04
— 1985

:2019

— —0412
— —-12
— T102200
— T1022
— T10
— T-2200
— T—00
— T102200Z
— T102200-0800

5.4.11.6. Normalization

No normalization procedures are needed.

5.4.12. ISO-DURATION-COMPLETE

This corresponds to the values accepted by “duration” as specified in ISO 8601:2004.

5.4.12.1. Value type notation

ISO-DURATION-COMPLETE

5.4.12.2. Purpose

Representing a duration of time specified by 4.4.3.2 complete representation basic format.

5.4.12.3. Format definition

```
iso-duration-sign = ["+"] / "-"
iso-duration  = ( iso-duration-sign ) "P" iso-duration-value

iso-duration-value =  iso-duration-date / iso-duration-week

iso-duration-date   = iso-duration-day "T" iso-duration-time

iso-duration-week   = 1*DIGIT "W"

iso-duration-year   = 1*DIGIT "Y"
iso-duration-month  = 1*DIGIT "M"
iso-duration-day    = 1*DIGIT "D"

iso-duration-time   = iso-duration-hour iso-duration-minute
                      iso-duration-second

iso-duration-hour   = 1*DIGIT "H"
iso-duration-minute = 1*DIGIT "M"
iso-duration-second = 1*DIGIT "S"
```

Figure 34

5.4.12.4. Description

The value format is based on the complete representation basic format specified in 4.4.3.2.

It accepts the following formats (“nn” represents):

— “PnnW” 4.4.3.2, complete representation, first basic format, for duration in weeks.

— “PnnYnnMnnDTnnHnnMnnS” 4.4.3.2, complete representation, second basic format, for duration in years, months, days, hours, minutes and seconds.

This format differs from the specification of 4.4.3.2 in the following areas:

— An optional, preceding “sign”, element is used to indicate positive or negative duration. Negative durations are useful in representing reverse scheduling, such as the time to trigger an alarm before an associated time (see IETF RFC 5545).
— Reduced accuracy as defined in 4.4.3.2 is not allowed. Omission of the number and corresponding designator of days, hours, minutes or seconds is not allowed even if any of the expressions are zero (4.4.3.2 c)).
— The duration of a week or a day depends on its position in the calendar.

In the case of discontinuities in the time scale, such as the change from standard time to daylight time and back, the computation of the exact duration requires the subtraction or addition of the change of duration of the discontinuity. Leap seconds **MUST NOT** be considered when computing an exact duration.

5.4.12.5. Examples

A duration of 15 days, 5 hours, and 20 seconds **MAY** be represented as

— P0Y0M15DT5H0M20S

A duration of 7 weeks **MAY** be represented as:

— P7W

5.4.12.6. Normalization

No normalization procedures are needed.

5.4.13. CAL-DURATION

This corresponds to the DURATION value type in 3.3.6.

5.4.13.1. Value type notation

CAL-DURATION

5.4.13.2. Purpose

Representing a duration of time specified by 4.4.3.2 complete representation basic format, similar to ISO-DURATION, but with syntax tailored to calendaring.

5.4.13.3. Format definition

```
cal-duration         = cal-duration-sign cal-duration-no-sign
cal-duration-sign    = (["+"] / "-")
cal-duration-no-sign = "P" cal-duration-value

cal-duration-value   = ( cal-duration-date /
                         cal-duration-time /
                         cal-duration-week )

cal-duration-date    = cal-duration-day [cal-duration-time]

cal-duration-time    = "T" ( cal-duration-hour /
```



```
                          cal-duration-minute /
                          cal-duration-second )

cal-duration-week   = 1*DIGIT "W"
cal-duration-hour   = 1*DIGIT "H" [cal-duration-minute]
cal-duration-minute = 1*DIGIT "M" [cal-duration-second]
cal-duration-second = 1*DIGIT "S"
cal-duration-day    = 1*DIGIT "D"
```

Figure 35

5.4.13.4. Description

The value format is similar to ISO-DURATION and based on the complete representation basic format specified in 4.4.3.2, but given extra flexibility to calendaring usage.

It accepts the following formats (“nn” represents):

— “PnnW” 4.4.3.2, complete representation, first basic format, for duration in weeks.
— “PnnDTnnHnnMnnS” 4.4.3.2, complete representation, second basic format, with the omission of years and months, for duration in days, hours, minutes and seconds.
— “PnnDTnnHnnM” Reduced accuracy with omission of seconds.
— “PnnDTnnH” Reduced accuracy with omission of minutes.
— “PnnD” Reduced accuracy with omission of hours.

This format differs from the specification of 4.4.3.2 in the following areas:

— Years and months are not accepted in this syntax.
— An optional, preceding “sign”, element is used to indicate positive or negative duration. Negative durations are useful in representing reverse scheduling, such as the time to trigger an alarm before an associated time (see IETF RFC 5545).
— Reduced accuracy is allowed for in particular, the omission of the number and designators of hours, minutes or seconds is allowed with the omission starting from the extreme right-hand side. In the case of the omission of the time value, the “T” separator **MUST** also be omitted. The day (“D”) portion **MUST** always be present.
— The duration of a week or a day depends on its position in the calendar.

In the case of discontinuities in the time scale, such as the change from standard time to daylight time and back, the computation of the exact duration requires the subtraction or addition of the change of duration of the discontinuity. Leap seconds **MUST NOT** be considered when computing an exact duration.

When computing an exact duration, the greatest order time components **MUST** be added first, that is, the number of days **MUST** be added first, followed by the number of hours, number of minutes, and number of seconds.

5.4.13.5. Example

A duration of 0 days, 0 hours, and 20 seconds **SHOULD** be represented as

```
P0DT0H0M20S
```

Figure 36

A duration of 15 days, 5 hours, and 3 hours **SHOULD** be represented as

```
P15DT5H3M
```

Figure 37

A duration of 15 days, 5 hours **SHOULD** be represented as

```
P15DT5H
```

Figure 38

A duration of 15 days **SHOULD** be represented as

```
P15D
```

Figure 39

A duration of 7 weeks **SHOULD** be represented as:

```
P7W
```

Figure 40

5.4.13.6. Normalization

No normalization procedures are needed.

5.4.14. ISO-INTERVAL

This corresponds to the values accepted by “time interval” as specified in ISO 8601:2004.

5.4.14.1. Value type notation

ISO-INTERVAL-COMPLETE

5.4.14.2. Purpose

Representation of a time interval.

5.4.14.3. Format definition

```
iso-interval     = iso-interval-explicit / iso-interval-start

iso-interval-explicit = iso-date-time "/" iso-date-time
iso-interval-start    = iso-date-time "/" iso-duration-no-sign
```

Figure 41

5.4.14.4. Description

This value format accepts a time interval representation, specified in 4.4 “Time Interval”.

The value can be represented by:

a) a start and an end;

— “YYYYMMDDThhmmss/YYYYMMDDThhmmss” 4.4.4.1 Complete representation, “Representations of time intervals identified by start and end”, basic format, first entry. The start **MUST** be before the end.

c) a start and a duration;

— “YYYYMMDDThhmmss/PnnYnnMnnDTnnHnnMnnS” 4.4.4.3 Complete representation, “Representations of time interval identified by start and duration”, first basic format. The duration component **MUST** be positive.

— “YYYYMMDDThhmmss/PnnW” 4.4.4.5 Other complete representations, third item, allowing the expression “PnnYnnMnnDTnnHnnMnnS” to be substituted with “PnnW” 4.4.3.2.

d) a duration and an end.

— “PnnYnnMnnDTnnHnnMnnS/YYYYMMDDThhmmss” 4.4.4.4 Complete representation, “Representations of time interval identified by duration and end”, first basic format. The start of the interval can be determined by subtracting the duration component from the end of the interval.
— “PnnW/YYYYMMDDThhmmss” 4.4.4.5 Other complete representations, third item, allowing the expression “PnnYnnMnnDTnnHnnMnnS” to be substituted with “PnnW” 4.4.3.2.

In accordance with 4.4.4.5:

— where representations using local time in a time point component are shown, a complete representation of UTC (4.2.4) or local time and the difference from UTC (4.2.5.2) **MAY** be substituted for local time, i.e. representations using the expression “YYYYMMDDThhmmss” could be substituted with any of these:
— “YYYYMMDDThhmmssZ” 4.3.2 Complete representation basic format, second entry.
— “YYYYMMDDThhmmss±hhmm” 4.3.2 Complete representation basic format, third entry.
— “YYYYMMDDThhmmss±hh” 4.3.2 Complete representation basic format, fourth entry.

In accordance with 4.4.5:

— representations for UTC included with the component preceding the solidus shall be assumed to apply to the component following the solidus, unless a corresponding alternative is included.

5.4.14.5. Examples

— 19850412T232050/P1Y2M15DT12H30M0S
— 19850412T232050Z/P1Y2M15DT12H30M0S
— 19850412T232050Z/19850612T232050
— P1Y2M15DT12H30M0S/19850412T232050

5.4.14.6. Normalization

No normalization procedures are needed.

5.4.15. CAL-INTERVAL

This corresponds to the PERIOD value type in 3.3.9.

5.4.15.1. Value type notation

CAL-INTERVAL

5.4.15.2. Purpose

Representation of a time interval for calendaring.

5.4.15.3. Format definition

```
cal-interval     = cal-interval-explicit / cal-interval-start

cal-interval-explicit = iso-date-time-no-zone "/" iso-date-time-no-zone
```



```
cal-interval-start    = iso-date-time-no-zone "/" cal-duration-no-sign
```

Figure 42

5.4.15.4. Description

This value format accepts a time interval representation, specified in 4.4. “Time Interval” tailored for calendaring purposes.

The value can be represented in two ways.

As an interval with start and end:

— “YYYYMMDDThhmmss/YYYYMMDDThhmmss” 4.4.4.1 Complete representation, “Representations of time intervals identified by start and end”, basic format, first entry. The start **MUST** be before the end.

As an interval with start and duration (positive duration only):

— “YYYYMMDDThhmmss/PnnDTnnHnnMnnS” 4.4.4.3 Complete representation, “Representations of time interval identified by start and duration”, first basic format, modified to omit the “nnYnnM”, which is the “cal-duration” period format.
— “YYYYMMDDThhmmss/PnnW” 4.4.4.5 Other complete representations, third item, allowing the expression “PnnYnnMnnDTnnHnnMnnS” to be substituted with “PnnW” 4.4.3.2.
— “YYYYMMDDThhmmss/PnnDTnnHnnM” with the duration specified in reduced accuracy with omission of seconds as in Clause 5.4.13.
— “YYYYMMDDThhmmss/PnnDTnnH” with the duration specified in reduced accuracy with omission of minutes as in Clause 5.4.13.
— “YYYYMMDDThhmmss/PnnD” with the duration specified in reduced accuracy with omission of hours as in Clause 5.4.13.

In accordance with 4.4.5, representations for UTC included with the component preceding the solidus shall be assumed to apply to the component following the solidus, unless a corresponding alternative is included.

5.4.15.5. Examples

— 19970101T180000Z/19970102T070000Z
— 19850412T232050/19850625T103000
— 19970101T180000Z/PT5H30M
— 19850412T232050/P15DT12H30M0S
— 19850412T232050/P00010215T123000
— Both components are in UTC: 19850412T232050Z/19850625T103000
— Former component in local time, latter in UTC: 19850412T232050/19850625T103000

5.4.15.6. Normalization

No normalization procedures are needed.

6. Normalization

A normalization procedure can be applied to vObjects (in its various representations) to make them compatible prior to comparison, allowing for consistent results. The result of normalization processing of a vObject, is an equivalent vObject described according to vFormat representation.

The normalization method has the following properties:

— stable across different implementations generating the same output from the same input

— compatible with alternative representation formats such as xCard IETF RFC 6351 / jCard IETF RFC 7095 and xCal IETF RFC 6321 / jCal IETF RFC 7265
— generates output adhering to the original vObject format allowing interoperability with existing implementations
— generates output compatible with protocols that utilize these vObject, such as CardDAV IETF RFC 6352 and CalDAV IETF RFC 4791 systems.

There are two levels of normalization.

— vObject normalization, of values and property parameter values, are performed within the vObject data model;
— vFormat normalization, of the format syntax itself, is performed during serialization of a vObject into vFormat.

6.1. Approach

The goals of the normalization procedure are:

— A normalized vObject will be a valid vObject in vFormat syntax. Therefore the normalization procedure requires knowledge of the source specific vObject format.
— A normalized vObject is stable across alternative representation formats, such as xCal and jCal of iCalendar, and xCard and jCard of vCard. This allows comparison of vObject content regardless of the representation format.
— Allows comparison of equivalence of content rather than formatting. E.g., addition of new-lines within a vCard and order of listed properties do not affect the resulting normalized form.
— A normalized vObject **MUST** maintain validity under the original format rules, such as in the case of VCARD IETF RFC 6350 components, the “VERSION” property line **MUST** be located immediately after the “BEGIN” property line.

6.2. Steps

In order to serialize a vObject into normalized vFormat syntax, one would directly serialize the vObject data model into vFormat syntax.

The steps are generally described below.

1) Normalize the vObject
   a) Normalize properties
      i) Normalize property parameters
         A. Normalize property parameter types
         B. Normalize property parameter values
            I. Sort property parameter values alphabetically.
            II. Concatenate property parameter values.
         C. Normalize property parameter key: cast to uppercase.
         D. Concatenate string form of property parameter key, value type and values.
      ii) Normalize property values
   b) Normalize inner components (sub-components)
      i) Perform the same function as (1)

6.3. Application on alternative serializations

The normalization procedure applies to alternative vObject representations as well, including:

— xCard IETF RFC 6351
— jCard IETF RFC 7095
— xCal IETF RFC 6321
— jCal IETF RFC 7265

To normalize a vObject provided in these representations, the vObject data should be first normalized in data model form according to Clause 3, and then serialized into these representations.

7. Client implementations recommendations

A CUA **SHOULD** normalize the vObject upon modification of it.

8. CardDAV

8.1. Additional server semantics for PUT, COPY and MOVE

This specification creates an additional precondition and postcondition for the PUT, COPY, and MOVE methods when:

— A PUT operation requests an address object resource to be placed into an address book collection; and
— A COPY or MOVE operation requests an address object resource to be placed into (or out of) an address book collection.

8.1.1. Provide normalized output

Certain servers perform silent changes or cleanups of client provided vCard data when stored as address object resources, such as the order of property parameters or scrubbed values.

The resulting vCard data stored on the server (and when returned back to the client) **MAY** end up different than that of the client without its knowledge. It is therefore necessary for the client to be reported on such modifications.

Additional Postcondition:

```
(CARDDAV:resource-normalized): Convert to normalized format.
```

Figure 43

9. CalDAV

9.1. Additional server semantics for PUT, COPY and MOVE

This specification creates an additional precondition and postcondition for the PUT, COPY, and MOVE methods when:

— A PUT operation of a calendar object resource into a calendar collection occurs IETF RFC 4791;
— A COPY or MOVE operation of a calendar object resource into a calendar collection occurs IETF RFC 4791; and
— A COPY or MOVE operation occurs on a calendar collection IETF RFC 4791.

9.1.1. Provide normalized output

Certain servers perform silent changes or cleanups of client provided iCalendar data when stored as calendar object resources, such as the order of property parameters or scrubbed values.

The resulting iCalendar data stored on the server (and when returned back to the client) **MAY** end up different than that of the client without its knowledge. It is therefore necessary for the client to be reported on such modifications.

:2019

Additional Postcondition:

```
(CALDAV:resource-normalized): Convert to normalized format.
```

Figure 44

10. Security considerations

Security considerations around vObject formats in the following documents **MUST** be adhered to:

— vCard: IETF RFC 6350
— iCalendar: IETF RFC 5545, IETF RFC 5789, IETF RFC 4791

11. IANA considerations

New vObject and vFormat specifications produced **MUST** adhere to the requirements, including the normalization process, described in this document, and any exceptions or further instructions for normalization **MUST** be described.

11.1. Common vObject registries

CalConnect created and will maintain the following registries for vObject elements with pointers to appropriate reference documents. The registries are grouped together under the heading “vObject Common Elements”.

11.2. vObject component uniqueness identifiers registry

11.2.1. Registration Procedure

This section defines the process to register new or modified uniqueness properties for vObject components with IANA.

The IETF will create a mailing list, vobject@ietf.org, which can be used for public discussion of vObject elements prior to registration.

The registry policy is **Specification Required**; any newly proposed specification **MUST** be reviewed by the designated expert.

The registry **SHOULD** contain the following note:

```
Note: Experts are to verify that the proposed registration
*SHOULD* provide benefits for the wider vObject community,
and provides a publicly-available standard that can be implemented in
an interoperable way. References to IETF-published documents are
preferred. The "Reference" value should point to a document that
details the implementation of this property.
```

Figure 45

The registration procedure begins when a completed registration template, defined in the sections below, is sent to vobject@ietf.org and iana@iana.org.

The designated expert is expected to tell IANA and the submitter of the registration within two weeks whether the registration is approved, approved with minor changes, or rejected with cause. When a registration is rejected with cause, it can be re-submitted if the concerns listed in the cause are addressed. Decisions made by the designated expert can be appealed to the IESG Applications Area Director, then to the IESG. They follow the normal appeals procedure for IESG decisions.

11.2.2. Registration template

A registration for a vObject Component Uniqueness Property is defined by completing the following template.

Component The name of the component.

Property The property of the component that is used to uniquely identify the component it belongs to.

Scope The uniqueness scope of the aforementioned property.

Reference The document that defines the component syntax and the uniqueness identifying property. Generally, this is where the component was originally defined, but if the uniqueness property is defined in an extension document, a reference to the extension document **SHOULD** be given instead.

Description Any special notes about the usage of the uniqueness identifying property, how it is to be used, etc.

Example(s) One or more examples of instances of the component need to be specified.

11.2.3. Initial registrations

The IANA created and maintains this registry for vObject Component Uniqueness Identifiers with pointers to appropriate reference documents.

The following table has been used to initialize the registry.

Table 1

Component	Property	Scope	Reference
VCALENDAR	UID	Global	5.3
VCARD	UID	Global	6.7.6
VEVENT	UID	Global	3.6.1
VTODO	UID	Global	3.6.2
VJOURNAL	UID	Global	3.6.3
VFREEBUSY	UID	Global	3.6.4
VTIMEZONE	TZID	Global	3.6.5
STANDARD	DTSTART	Parent	3.6.5
DAYLIGHT	DTSTART	Parent	3.6.5
VALARM	UID	Global	4
VAVAILABILITY	UID	Global	3.1
AVAILABLE	UID	Global	3.1
VPOLL	UID	Parent	4.5.1
VVOTER	VOTER	Parent	4.5.2
VOTE	POLL-ITEM-ID	Parent	4.5.3

12. Mapping of data value types for existing RFCs

The vObject value types in this section are described using vObject value type notation (see Clause 5.1).

12.1. RFC 6350

Table 2

vObject Value Type	Original Value Type
BOOLEAN	BOOLEAN
ISO-DATE-FLEX	DATE
ISO-DATE-AND-OR-TIME	DATE-AND-OR-TIME
ISO-DATE-TIME-FLEX	DATE-TIME
FLOAT	FLOAT
INTEGER-64	INTEGER
LANGUAGE-TAG	LANGUAGE-TAG
TEXT	TEXT
ISO-TIME-FLEX	TIME
ISO-TIME-COMPLETE	TIMESTAMP
URI	URI
ISO-UTC-OFFSET	UTC-OFFSET

12.2. RFC 5545

Table 3

vObject Value Type	Original Value Type
BINARY	BINARY
BOOLEAN	BOOLEAN
URI	CAL-ADDRESS
ISO-DATE-COMPLETE	DATE
ISO-DATE-TIME-BASIC	DATE-TIME
CAL-DURATION	DURATION
FLOAT	FLOAT
INTEGER-32	INTEGER
CAL-DURATION	PERIOD
TEXT	TEXT
ISO-TIME-BASIC	TIME
URI	URI
CAL-UTC-OFFSET	UTC-OFFSET
RECURMAP (Clause 12.2.1)	RECUR

12.2.1. RECURMAP

RECURMAP is shown here instead of within the tables due to space constraints.

It is defined to be the value type of the following vObject value type:

```
RECURMAP = MAP(
  KEYVALUE(FREQ, TEXT),
  KEYVALUE(UNTIL, ISO-DATE-COMPLETE / ISO-DATE-TIME-BASIC),
  KEYVALUE(COUNT, INTEGER),
  KEYVALUE(INTERVAL, INTEGER),
  KEYVALUE(BYSECOND, LIST(INTEGER)),
  KEYVALUE(BYMINUTE, LIST(INTEGER)),
  KEYVALUE(BYHOUR, LIST(INTEGER)),
  KEYVALUE(BYDAY, LIST(INTEGER)),
  KEYVALUE(BYMONTHDAY, LIST(INTEGER)),
  KEYVALUE(BYYEARDAY, LIST(INTEGER)),
  KEYVALUE(BYWEEKNO, LIST(INTEGER)),
  KEYVALUE(BYMONTH, LIST(INTEGER)),
  KEYVALUE(BYSETPOS, INTEGER),
  KEYVALUE(WKST, TEXT)
```

)

Figure 46

13. Mapping of component property value types for existing RFCs

The default and alternative value types in this section are described using vObject value type notation (see Clause 5.1).

13.1. VCARD component (RFC 6350)

Properties with the default data type as TEXT.

Table 4

Property	Default Value Type	Alt. Value Types	Original Value Type
BEGIN	TEXT		1*TEXT
END	TEXT		1*TEXT
KIND	TEXT		1*TEXT
XML	TEXT		1*TEXT
FN	TEXT		1*TEXT
BDAY	ISO-DATE-AND-OR-TIME	TEXT	1*date-and-or-time, 1*text
ANNIVERSARY	ISO-DATE-AND-OR-TIME	TEXT	1*date-and-or-time, 1*text
EMAIL	TEXT		1*TEXT,
TZ	TEXT	URI, ISO-UTC-OFFSET	1*TEXT, URI, UTC-OFFSET
TITLE	TEXT		1*TEXT
ROLE	TEXT		1*TEXT
NOTE	TEXT		1*TEXT
PRODID	TEXT		1*TEXT
VERSION	TEXT		1*TEXT

Properties with the default data type as URI.

Table 5

Property	Default Value Type	Alt. Value Types	Original Value Type
TEL	URI	TEXT	1*text, URI
SOURCE	URI		URI
PHOTO	URI		URI
IMPP	URI		URI
GEO	URI		URI
LOGO	URI		URI
MEMBER	URI		URI
RELATED	URI	TEXT	URI, 1*text
UID	URI		URI, 1*text
KEY	URI		URI, 1*text
SOUND	URI		URI
URL	URI		URI
FBURL	URI		URI
CALADRURI	URI		URI
CALURI	URI		URI

Properties with FIELDSET.

Table 6

Property	Default Value Type	Alt. Value Types	Original Value Type
N	FIELDSET(5*LIST(TEXT))		TEXT
GENDER	FIELDSET(2*TEXT)		TEXT
ADR	FIELDSET(7*LIST(TEXT))		TEXT
ORG	FIELDSET(1*TEXT)		TEXT
CLIENTPIDMAP	FIELDSET(INTEGER-64, URI)		TEXT

Properties with LIST.

Table 7

Property	Default Value Type	Alt. Value Types	Original Value Type
NICKNAME	LIST(TEXT)		TEXT
CATEGORIES	LIST(TEXT)		TEXT

Properties with ISO-DATE-AND-OR-TIME.

Table 8

Property	Default Value Type	Alt. Value Types	Original Value Type
BDAY	ISO-DATE-AND-OR-TIME	TEXT	date-and-or-time, text
ANNIVERSARY	ISO-DATE-AND-OR-TIME	TEXT	date-and-or-time, text

Properties with ISO-DATE-TIME-COMPLETE.

Table 9

Property	Default Value Type	Alt. Value Types	Original Value Type
REV	ISO-DATE-TIME-COMPLETE		timestamp

Properties with LANGUAGE-TAG.

Table 10

Property	Default Value Type	Alt. Value Types	Original Value Type
LANG	LANGUAGE-TAG		language-tag

13.2. VCALENDAR component (RFC 5545)

Table 11

Property	Default Value Type	Alt. Value Types	Original Value Type
PRODID	TEXT		1*TEXT
VERSION	TEXT		1*TEXT
CALSCALE	TEXT		1*TEXT
METHOD	TEXT		1*TEXT
IANA-REGed/X-	TEXT		1*TEXT

13.3. VEVENT component (RFC 5545)

Table 12

Property	Default Value Type	Alt. Value Types	Original Value Type
DTSTAMP	ISO-DATE-TIME-BASIC		DATE-TIME
UID	TEXT		1*TEXT
DTSTART	ISO-DATE-TIME-BASIC	ISO-DATE-COMPLETE	DATE-TIME, DATE
CLASS	TEXT		1*TEXT
CREATED	ISO-DATE-TIME-BASIC		DATE-TIME
DESCRIPTION	TEXT		1*TEXT
GEO	FIELDSET(2*FLOAT)		FLOAT ";" FLOAT
LAST-MODIFIED	ISO-DATE-TIME-BASIC		DATE-TIME
LOCATION	TEXT		1*TEXT
ORGANIZER	URI		cal-address
PRIORITY	INTEGER-32		INTEGER
SEQUENCE	INTEGER-32		INTEGER
STATUS	TEXT		1*TEXT
SUMMARY	TEXT		1*TEXT
TRANSP	TEXT		1*TEXT
URL	URI		URI

Property	Default Value Type	Alt. Value Types	Original Value Type
RECURRENCE-ID	ISO-DATE-TIME-BASIC	ISO-DATE-COMPLETE	DATE-TIME, DATE
RRULE	RECURMAP (Clause 12.2.1)		RECUR
DTEND	ISO-DATE-TIME-BASIC	ISO-DATE-COMPLETE	DATE-TIME, DATE
DURATION	DURATION		DURATION
ATTACH	URI	BINARY	URI, BINARY
ATTENDEE	URI		cal-address
CATEGORIES	LIST(TEXT)		TEXT
COMMENT	TEXT		1*TEXT
CONTACT	TEXT		1*TEXT
EXDATE	LIST(ISO-DATE-TIME-BASIC / ISO-DATE-COMPLETE)		DATE-TIME, DATE
RELATED-TO	TEXT		1*TEXT
RESOURCES	LIST(TEXT)		TEXT
RDATE	LIST(ISO-DATE-TIME-BASIC / ISO-DATE-COMPLETE / CAL-INTERVAL)		DATE-TIME, DATE, PERIOD
IANA-REGed/X-	TEXT		1*TEXT

13.4. VTODO component (RFC 5545)

Table 13

Property	Default Value Type	Alt. Value Types	Original Value Type
DTSTAMP	ISO-DATE-TIME-BASIC		DATE-TIME
UID	TEXT		1*TEXT
CLASS	TEXT		1*TEXT
CREATED	ISO-DATE-TIME-BASIC		DATE-TIME
COMPLETED	ISO-DATE-TIME-BASIC		DATE-TIME
DESCRIPTION	TEXT		1*TEXT
DTSTART	ISO-DATE-TIME-BASIC	ISO-DATE-COMPLETE	DATE-TIME, DATE
GEO	FIELDSET(2*FLOAT)		FLOAT ";" FLOAT
LAST-MODIFIED	ISO-DATE-TIME-BASIC		DATE-TIME
LOCATION	TEXT		1*TEXT
ORGANIZER	URI		cal-address
PRIORITY	INTEGER-32		INTEGER
SEQUENCE	INTEGER-32		INTEGER
STATUS	TEXT		1*TEXT
SUMMARY	TEXT		1*TEXT
URL	URI		URI
RRULE	RECURMAP (Clause 12.2.1)		RECUR
DUE	ISO-DATE-TIME-BASIC	ISO-DATE-COMPLETE	DATE-TIME, DATE
DURATION	DURATION		DURATION
ATTACH	URI	BINARY	URI, BINARY
ATTENDEE	URI		cal-address
CATEGORIES	LIST(TEXT)		TEXT
COMMENT	TEXT		1*TEXT
CONTACT	TEXT		1*TEXT
EXDATE	LIST(ISO-DATE-TIME-BASIC / ISO-DATE-COMPLETE)		DATE-TIME, DATE
REQUEST-STATUS	TEXT		1*TEXT
RELATED-TO	TEXT		1*TEXT
RESOURCES	LIST(TEXT)		TEXT

Property	Default Value Type	Alt. Value Types	Original Value Type
RDATE	LIST(ISO-DATE-TIME-BASIC / ISO-DATE-COMPLETE / CAL-INTERVAL)		DATE-TIME, DATE, PERIOD
IANA-REGed/X-	TEXT		1*TEXT

13.5. VJOURNAL component (RFC 5545)

Table 14

Property	Default Value Type	Alt. Value Types	Original Value Type
DTSTAMP	ISO-DATE-TIME-BASIC		DATE-TIME
UID	TEXT		1*TEXT
CLASS	TEXT		1*TEXT
CREATED	ISO-DATE-TIME-BASIC		DATE-TIME
DTSTART	ISO-DATE-TIME-BASIC	ISO-DATE-COMPLETE	DATE-TIME, DATE
LAST-MODIFIED	ISO-DATE-TIME-BASIC		DATE-TIME
ORGANIZER	URI		cal-address
SEQUENCE	INTEGER-32		INTEGER
STATUS	TEXT		1*TEXT
SUMMARY	TEXT		1*TEXT
URL	URI		URI
RRULE	RECURMAP (Clause 12.2.1)		RECUR
ATTACH	URI	BINARY	URI, BINARY
ATTENDEE	URI		cal-address
CATEGORIES	LIST(TEXT)		TEXT
COMMENT	TEXT		1*TEXT
CONTACT	TEXT		1*TEXT
DESCRIPTION	TEXT		1*TEXT
EXDATE	LIST(ISO-DATE-TIME-BASIC / ISO-DATE-COMPLETE)		DATE-TIME, DATE
RELATED-TO	TEXT		1*TEXT
RDATE	LIST(ISO-DATE-TIME-BASIC / ISO-DATE-COMPLETE / CAL-INTERVAL)		DATE-TIME, DATE, PERIOD
REQUEST-STATUS	TEXT		1*TEXT
IANA-REGed/X-	TEXT		1*TEXT

13.6. VFREEBUSY component (RFC 5545)

Table 15

Property	Default Value Type	Alt. Value Types	Original Value Type
DTSTAMP	ISO-DATE-TIME-BASIC		DATE-TIME
UID	TEXT		1*TEXT
CONTACT	TEXT		1*TEXT
DTSTART	ISO-DATE-TIME-BASIC	ISO-DATE-COMPLETE	DATE-TIME, DATE
DTEND	ISO-DATE-TIME-BASIC	ISO-DATE-COMPLETE	DATE-TIME, DATE
ORGANIZER	URI		cal-address
URL	URI		URI
ATTENDEE	URI		cal-address
COMMENT	TEXT		1*TEXT
FREEBUSY	LIST(CAL-INTERVAL)		LIST(PERIOD)
REQUEST-STATUS	TEXT		1*TEXT
IANA-REGed/X-	TEXT		1*TEXT

13.7. VTIMEZONE component (RFC 5545)

Table 16

Property	Default Value Type	Alt. Value Types	Original Value Type
TZID	TEXT		1*TEXT
LAST-MODIFIED	ISO-DATE-TIME-BASIC		DATE-TIME
TZURL	URI		URI
IANA-REGed/X-	TEXT		1*TEXT

13.8. STANDARD / DAYLIGHT Components (RFC 5545)

Table 17

Property	Default Value Type	Alt. Value Types	Original Value Type
DTSTART	ISO-DATE-TIME-BASIC	ISO-DATE-COMPLETE	DATE-TIME, DATE
TZOFFSETFROM	CAL-UTC-OFFSET		UTC-OFFSET
TZOFFSETTO	CAL-UTC-OFFSET		UTC-OFFSET
RRULE	RECURMAP (Clause 12.2.1)		RECUR
COMMENT	TEXT		1*TEXT
RDATE	LIST(ISO-DATE-TIME-BASIC / ISO-DATE-COMPLETE / CAL-INTERVAL)		DATE-TIME, DATE, PERIOD
TZNAME	TEXT		1*TEXT
IANA-REGed/X-	TEXT		1*TEXT

13.9. VALARM component (RFC 5545)

Table 18

Property	Default Value Type	Alt. Value Types	Original Value Type
ACTION	TEXT		1*TEXT
DESCRIPTION	TEXT		1*TEXT
SUMMARY	TEXT		1*TEXT
TRIGGER	DURATION	ISO-DATE-TIME-BASIC	DURATION, DATE-TIME
DURATION	DURATION		DURATION
REPEAT	INTEGER-32		INTEGER
ATTACH	URI	BINARY	URI, BINARY
ATTENDEE	URI		cal-address
IANA-REGed/X-	TEXT		1*TEXT

14. Mapping of parameter value types for existing RFCs

The value types in this section are described using vObject value type notation (see Clause 5.1).

14.1. RFC 6350

Table 19

Parameter	Value Type
LANGUAGE	LANGUAGE-TAG
VALUE	TEXT
PREF	INTEGER-64
ALTID	TEXT
PID	TEXT
TYPE	LIST(TEXT)
MEDIATYPE	TEXT
CALSCALE	TEXT

:2019

Parameter	Value Type
SORT-AS	LIST(TEXT)
GEO	URI
TZ	TEXT

14.2. RFC 5545

Table 20

Parameter	Value Type
ALTREP	URI
CN	TEXT
CUTYPE	TEXT
DELEGATED-FROM	URI
DELEGATED-TO	URI
DIR	URI
ENCODING	TEXT
FMTTYPE	TEXT
FBTYPE	TEXT
LANGUAGE	LANGUAGE-TAG
MEMBER	LIST(URI)
PARTSTAT	TEXT
RANGE	TEXT
RELATED	TEXT
RELTYPE	TEXT
ROLE	TEXT
RSVP	BOOLEAN
SENT-BY	URI
TZID	TEXT
VALUE	TEXT

15. Normalization examples for vFormat

Original:

```
BEGIN:VOBJECT
PROPERTY1:10
PROPERTY2:20
END:VOBJECT
```

Figure 47

Normalized:

```
BEGIN:VOBJECT
PROPERTY1:10
PROPERTY2:20
END:VOBJECT
```

Figure 48

15.1. vCard

Original:

```
BEGIN:VCARD
VERSION:4.0
KIND:individual
FN:Martin Van Buren
N:Van Buren;Martin;;;Hon.
```

```
TEL;VALUE=uri;PREF=1;TYPE="voice";TYPE="home":tel:+1-888-888-8888;ext=8888
END:VCARD
```

Figure 49

Normalized:

```
BEGIN:VCARD
VERSION:4.0
KIND:individual
FN:Martin Van Buren
N:Van Buren;Martin;;;Hon.
TEL;VALUE=uri;PREF=1;TYPE="voice","home":tel:+1-888-888-8888;ext=8888
END:VCARD
```

Figure 50

Bibliography

[1] CalConnect TC VCARD, *CalConnect VCARD Technical Committee*

[2] CalConnect TC CALENDAR, *CalConnect CALENDAR Technical Committee*

[3] IEEE 754™-2008, Institute of Electrical and Electronics Engineers. *IEEE Standard for Floating-Point Arithmetic*. 2013. https://ieeexplore.ieee.org/document/4610935.

[4] ISO 8601:2000, International Organization for Standardization. *Data elements and interchange formats — Information interchange — Representation of dates and times*. Second edition. 2000. Geneva. https://www.iso.org/standard/26780.html.

[5] ISO 8601:2004, International Organization for Standardization. *Data elements and interchange formats — Information interchange — Representation of dates and times*. Third edition. 2004. Geneva. https://www.iso.org/standard/40874.html.

[6] ISO 8601-1:2019, International Organization for Standardization. *Date and time — Representations for information interchange — Part 1: Basic rules*. First edition. 2019. Geneva. https://www.iso.org/standard/70907.html.

[7] ISO 8601-2:2019, International Organization for Standardization. *Date and time — Representations for information interchange — Part 2: Extensions*. First edition. 2019. Geneva. https://www.iso.org/standard/70908.html.

[8] IETF RFC 2045, N. FREED and N. BORENSTEIN. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. 1996. RFC Publisher. https://www.rfc-editor.org/info/rfc2045.

[9] IETF RFC 3552, E. RESCORLA and B. KORVER. *Guidelines for Writing RFC Text on Security Considerations*. 2003. RFC Publisher. https://www.rfc-editor.org/info/rfc3552.

[10] IETF RFC 3629, F. YERGEAU. *UTF-8, a transformation format of ISO 10646*. 2003. RFC Publisher. https://www.rfc-editor.org/info/rfc3629.

[11] IETF RFC 4648, S. JOSEFSSON. *The Base16, Base32, and Base64 Data Encodings*. 2006. RFC Publisher. https://www.rfc-editor.org/info/rfc4648.

[12] IETF RFC 4791, C. DABOO, B. DESRUISSEAUX and L. DUSSEAULT. *Calendaring Extensions to WebDAV (CalDAV)*. 2007. RFC Publisher. https://www.rfc-editor.org/info/rfc4791.

[13] IETF RFC 5234, P. OVERELL. *Augmented BNF for Syntax Specifications: ABNF*. 2008. RFC Publisher. https://www.rfc-editor.org/info/rfc5234.

[14] IETF RFC 5789, L. DUSSEAULT and J. SNELL. *PATCH Method for HTTP*. 2010. RFC Publisher. https://www.rfc-editor.org/info/rfc5789.

[15] IETF RFC 6321, C. DABOO, M. DOUGLASS and S. LEES. *xCal: The XML Format for iCalendar*. 2011. RFC Publisher. https://www.rfc-editor.org/info/rfc6321.

[16] IETF RFC 6351, S. PERREAULT. *xCard: vCard XML Representation*. 2011. RFC Publisher. https://www.rfc-editor.org/info/rfc6351.

[17] IETF RFC 6352, C. DABOO. *CardDAV: vCard Extensions to Web Distributed Authoring and Versioning (WebDAV)*. 2011. RFC Publisher. https://www.rfc-editor.org/info/rfc6352.

[18] IETF RFC 7095, P. KEWISCH. *jCard: The JSON Format for vCard*. 2014. RFC Publisher. https://www.rfc-editor.org/info/rfc7095.

[19] IETF RFC 7265, P. KEWISCH, C. DABOO and M. DOUGLASS. *jCal: The JSON Format for iCalendar*. 2014. RFC Publisher. https://www.rfc-editor.org/info/rfc7265.

[20] IETF RFC 7953, C. DABOO and M. DOUGLASS. *Calendar Availability*. 2016. RFC Publisher. https://www.rfc-editor.org/info/rfc7953.

[21] IETF RFC 7986, C. DABOO. *New Properties for iCalendar*. 2016. RFC Publisher. https://www.rfc-editor.org/info/rfc7986.

[22] IETF RFC 8259, T. BRAY (ed.). *The JavaScript Object Notation (JSON) Data Interchange Format*. 2017. RFC Publisher. https://www.rfc-editor.org/info/rfc8259.

[23] VPATCH, *The iCalendar VPATCH Component (draft)*

[24] vCard 2.1, *vCard — The Electronic Business Card Version 2.1*

[25] IETF RFC 9074, C. DABOO. *"VALARM" Extensions for iCalendar*. 2021. RFC Publisher. https://www.rfc-editor.org/info/rfc9074.

[26] Internet-Draft draft-ietf-calext-vpoll-00, ERIC YORK, CYRUS DABOO and MICHAEL DOUGLASS. *VPOLL: Consensus Scheduling Component for iCalendar*. 2019. https://datatracker.ietf.org/doc/html/draft-ietf-calext-vpoll-00.

