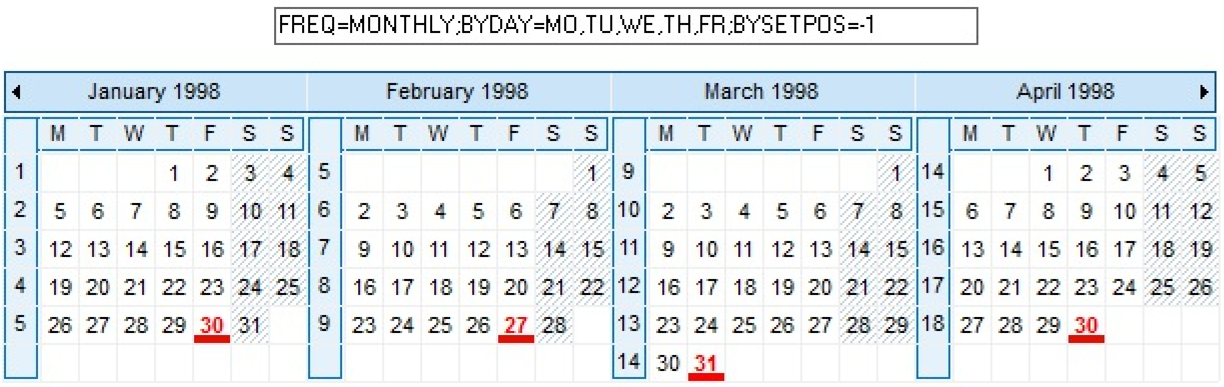
ExICalendar

The ExICalendar library implements the ICalendar data format, according with Internet Calendaring and Scheduling Core Object Specification, RFC 5545. The iCalendar data format represents exchanging calendaring and scheduling information such as events, to-dos, journal entries, and free/busy information, independent of any particular calendar service or protocol. The iCalendar format is suitable as an exchange format between applications or systems. The format is defined in terms of a MIME content type. This will enable the object to be exchanged using several transports, including but not limited to SMTP, HTTP, a file system, desktop interactive protocols such as the use of a memory-based clipboard or drag/drop interactions, point-to-point asynchronous communication, and so on. The eXICalendar library provides ICS capabilities for components like eXCalendar, eXGantt, eXG2antt, eXSchedule, and so on.

Features include:

- Easy way to access the components, properties and parameters of the iCalendar data
- Ability to load/save data from strings, files,...
- Ability to encode/decode properties from/to VARIANT expressions
- Ability to encode/decode properties with multiple values, like Recur, Period, Duration, and so on
- Ability to evaluate Recurrence/Recur rules
- Ability to enumerate occurrences of the Recurrence/Recur rule
- Template/X-Script support



```
FREQ=MONTHLY;BYDAY=MO,TU,WE,TH,FR;BYSETPOS=-1
```

January 1998

	M	T	W	T	F	S	S
1				1	2	3	4
2	5	6	7	8	9	10	11
3	12	13	14	15	16	17	18
4	19	20	21	22	23	24	25
5	26	27	28	29	30	31	

February 1998

	M	T	W	T	F	S	S
5							1
6	2	3	4	5	6	7	8
7	9	10	11	12	13	14	15
8	16	17	18	19	20	21	22
9	23	24	25	26	27	28	

March 1998

	M	T	W	T	F	S	S
9							1
10	2	3	4	5	6	7	8
11	9	10	11	12	13	14	15
12	16	17	18	19	20	21	22
13	23	24	25	26	27	28	29
14	30	31					

April 1998

	M	T	W	T	F	S	S
14			1	2	3	4	5
15	6	7	8	9	10	11	12
16	13	14	15	16	17	18	19
17	20	21	22	23	24	25	26
18	27	28	29	30			

How to get support?

To keep your business applications running, you need support you can count on.

Here are few hints what to do when you're stuck on your programming:

- Check out the samples - they are here to provide some quick info on how things should be done
- Check out the how-to questions using the eXHelper tool
- Check out the help - includes documentation for each method, property or event
- Check out if you have the latest version, and if you don't have it send an update request here.
- Submit your problem(question) here.

Don't forget that you can contact our development team if you have ideas or requests for new components, by sending us an e-mail at support@exontrol.com (please include the name of the product in the subject, ex: exgrid) . We're sure our team of developers will try to find a way to make you happy - and us too, since we helped.

Regards,
Exontrol Development Team

http://www.exontrol.com

constants PropertyTypeEnum

The PropertyTypeEnum type indicates the type of the properties in the iCalendar format. The Type / GuessType property specifies the property's type. The valuesFromICalendar property extracts all values or specified value of the giving value/type in ICalendar format. The Value property specifies the value of the property. The toICalendar property converts the giving value to a specified type as iCalendar format.

The PropertyTypeEnum type supports the following values:

Name	Value	Description
exPropertyTypeUnknown	-1	The VALUE parameter has an unknown value. The Parameters property retrieves the property's parameters. The VALUE parameter specifies the value type and format of the property value. The VALUE parameter can be any of the following: • "BINARY" • "BOOLEAN" • "CAL-ADDRESS" • "DATE" • "DATE-TIME" • "DURATION" • "FLOAT" • "INTEGER" • "PERIOD" • "RECUR" • "TEXT" • "TIME" • "URI" • "UTC-OFFSET" For instance, the following iCalendar format `DTSTART;VALUE=DATE:19971102` represents the DTSTART property of DATE type with the value of #11/02/1997#
exPropertyTypeMissing	0	The VALUE parameter is missing. The Parameters property retrieves the property's parameters.
		This value type is used to identify properties that

exPropertyTypeBinary	1	contain a character encoding of inline binary data (VT_ARRAY | VT_UI1).
		
		Example: The following is an example of a "BASE64" encoded binary value data:
		
		`ATTACH;FMTTYPE=image/vnd.microsoft.icon;ENCC`
		
		`=BINARY:AAABAAEAEBAQAAEABAAoAQAAFgAAA`
		
		`AAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAA`
		
		`AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA`
		
		`AAAAAAAAAAAAAAAAAAAAAMwAAAAAAABNE`
		
		`ACECQ0QgEgAAQxQzM0E0AABERCRCREQAADRD`
		
		`AAAAAAAAREQAAAAAAAAkQgAAAAAAAAMgAA`
		
		`AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA`
		
		`AAAAAAAAAAAA`

exPropertyTypeBoolean	2	This value type is used to identify properties that contain either a TRUE or FALSE Boolean value (VT_BOOL). Example: The following is an example of a BOOLEAN value data: `Boolean1:TRUE`
exPropertyTypeCalAddress	3	This value type is used to identify properties that contain a calendar user address (VT_BSTR). Example: The following is an example of a CAL-ADDRESS value data: `caladdress1:mailto:support@exontrol.com`

exPropertyTypeDate	4	This value type is used to identify values that contain a calendar date (VT_DATE). Example: The following is an example of a DATE value data: `Date2:20010101`
exPropertyTypeDateTime	5	This value type is used to identify values that specify a precise calendar date and time of day (VT_DATE). Example: The following is an example of a DATETIME value data: `DateTime1:20010101T120000`
exPropertyTypeDuration	6	This value type is used to identify properties that contain a duration of time (VT_R4). Example: The following is an example of a DURATION value data: `Duration1:P2DT12H`
exPropertyTypeFloat	7	This value type is used to identify properties that contain a real-number value (VT_R8). Example: The following is an example of a FLOAT value data: `Float1:1.5`
exPropertyTypeInteger	8	This value type is used to identify properties that contain a signed integer value (VT_I4). Example: The following is an example of a INTEGER value data: `Integer1:1`

exPropertyTypePeriod	9	This value type is used to identify values that contain a precise period of time (VT_BSTR). Example: The following is an example of a PERIOD value data: `Period1:20010101T000000/P1D`
exPropertyTypeRecur	10	This value type is used to identify properties that contain a recurrence rule specification (VT_BSTR). Example: The following is an example of a RECUR value data: `DTSTART=19970805T090000;FREQ=WEEKLY;INTER`
exPropertyTypeText	11	This value type is used to identify values that contain human-readable text (VT_BSTR). Example: The following is an example of a TEXT value data: `Text1:A1`
exPropertyTypeTime	12	This value type is used to identify values that contain a time of day (VT_DATE). Example: The following is an example of a TIME value data: `Time1:120000`
exPropertyTypeURI	13	This value type is used to identify values that contain a uniform resource identifier (URI) type of reference to the property value (VT_BSTR). Example: The following is an example of a URI value data: `URI:https://www.exontrol.com`
		This value type is used to identify properties that

exPropertyTypeUTCOffset	14	contain an offset from UTC to local time (VT_BSTR). Example: The following is an example of a UTCOffset value data: `UTCOffset:+0100`

constants RecurAllMethodEnum

The RecurAllMethod property specifies the way the component gets the occurrences of the recurrence rule (RecurAll method). Currently, the RecurAllMethodEnum type supports the following values:

Name	Value	Description
exRecurAllQuickValidate	0	exRecurAllQuickValidate.
exRecurAllCollectAndValidate	1	exRecurAllCollectAndValidate (for internal use).

constants RecurPartEnum

The RecurPartEnum type specifies the parts of the recurrence rule. The RecurPartValue property specifies the value of giving part of the recurrence rule. The RecurPartEnum type supports the following values:

Name	Value	Description
exRecurSyntaxErrorInfo	-2	Returns syntax error description.
exRecurSyntaxError	-1	Returns syntax error code.
exRecurFREQ	0	Returns the FREQ rule value.
exRecurDTSTART	1	Returns the DTSTART property value.
exRecurUNTIL	2	Returns the UNTIL rule value.
exRecurCOUNT	3	Returns the COUNT rule value.
exRecurINTERVAL	4	Returns the INTERVAL rule value.
exRecurBYSECOND	5	Returns the BYSECOND rule value.
exRecurBYMINUTE	6	Returns the BYMINUTE rule value.
exRecurBYHOUR	7	Returns the BYHOUR rule value.
exRecurBYDAY	8	Returns the BYDAY rule value.
exRecurBYMONTHDAY	9	Returns the BYMONTHDAY rule value.
exRecurBYYEARDAY	10	Returns the BYYEARDAY rule value.
exRecurBYWEEKNO	11	Returns the BYWEEKNO rule value.
exRecurBYMONTH	12	Returns the BYMONTH rule value.
exRecurBYSETPOS	13	Returns the BYSETPOS rule value.
exRecurWKST	14	Returns the WKST property value.

Component object

The Component object holds a component object of iCalendar format. The body of the iCalendar object consists of a sequence of calendar properties and one or more calendar components. The calendar properties are attributes that apply to the calendar object as a whole. The calendar components are collections of properties that express a particular calendar semantic. For example, the calendar component can specify an event, a to-do, a journal entry, time zone information, free/busy time information, or an alarm.

The following is a simple example of an iCalendar component:

```
BEGIN:VEVENT
UID:19970610T172345Z-AF23B2@example.com
DTSTAMP:19970610T172345Z
DTSTART:19970714T170000Z
DTEND:19970715T040000Z
SUMMARY:Bastille Day Party
END:VEVENT
```

The Component object supports the following properties and methods:

Name	Description
Clear	Clears the component, by removing the name, properties and components.
Components	Retrieves the child components of the current component.
Name	Indicates the component's name.
Parent	Retrieves the parent of the component.
Properties	Retrieves the properties of the current component.
toICalendar	Gets the iCalendar representation of the component.
UserData	Indicates any extra data associated with the component.

method Component.Clear ()

Clears the component, by removing the name, properties and components.

Type	Description

The Clear method empties the component, by removing the name, properties and components. Use the Remove method to remove a component from the Components collection.

The Clear methods do the following:

- empties the component's Name property.
- clears the Properties collection
- clears the Components collection

property Component.Components as Components

Retrieves the child components of the current component.

Type	Description
Components	A Components collection that holds the child components of the current component.

The Components property retrieves the child components of the current component. The Add method adds a child component to the current's Components collection. The Clear method empties the component, by removing the name, properties and components. Use the Remove method to remove a component from the Components collection.

property Component.Name as String

Indicates the component's name.

Type	Description
String	A String expression that specifies the name of the component.

The Name property specifies the name of the component. The Name parameter of the Add method, specifies the name of the component to be added. The Clear method clears the component, by removing the name, properties and components. The Parent property specifies the parent component of the current component. The Properties property gives access to the component's Properties collection.

The following is a simple example of an iCalendar component:

```
BEGIN:VEVENT
UID:19970610T172345Z-AF23B2@example.com
DTSTAMP:19970610T172345Z
DTSTART:19970714T170000Z
DTEND:19970715T040000Z
SUMMARY:Bastille Day Party
END:VEVENT
```

The VEVENT indicates the name of the component.

property Component.Parent as Component

Retrieves the parent of the component.

Type	Description
Component	A Component object that indicates the parent component.

The Parent property specifies the parent component of the current component. The Name property specifies the name of the component. The Properties property gives access to the component's Properties collection.

The following is a simple example of an iCalendar object:

```
BEGIN:VCALENDAR
VERSION:2.0
PRODID:-//hacksw/handcal//NONSGML v1.0//EN
BEGIN:VEVENT
UID:19970610T172345Z-AF23B2@example.com
DTSTAMP:19970610T172345Z
DTSTART:19970714T170000Z
DTEND:19970715T040000Z
SUMMARY:Bastille Day Party
END:VEVENT
END:VCALENDAR
```

In the sample, the parent of the VEVENT component is VCALENDAR

property Component.Properties as Properties

Retrieves the properties of the current component.

Type	Description
Properties	A Properties collection that holds Property objects that belongs to the current component.

The Properties property retrieves the properties collection of the current component. The Add method of the Properties object adds a new property to the current component. The Clear method clears the properties collection. The Remove method removes the property from the Properties collection.

The following is a simple example of an iCalendar component:

```
BEGIN:VEVENT
UID:19970610T172345Z-AF23B2@example.com
DTSTAMP:19970610T172345Z
DTSTART:19970714T170000Z
DTEND:19970715T040000Z
SUMMARY:Bastille Day Party
END:VEVENT
```

The UID, DTSTAMP, DTSTART, DTEND and SUMMARY are properties of the VEVENT component.

property Component.toICalendar as String

Gets the iCalendar representation of the component.

Type	Description
String	A String expression that specifies the iCalendar format of the current component

The toICalendar property retrieves the iCalendar representation of the component. You can use the Load / LoadFile / LoadFileFromUnicode methods load iCalendar format, while Save / SaveFile / SaveFileAsUnicode methods save the control's content as iCalendar format.

property Component.UserData as Variant

Indicates any extra data associated with the component.

Type	Description
Variant	A VARIANT expression that indicates any extra data associated with the component.

By default, the UserData property holds nothing. Use the UserData property to associate any extra-data to the component object. The AddComponent event notifies your application once a new Component object is added to the Components collection.

Components object

The Components collection holds Component objects. The Components property give access to the current component's child Components collection.

The following is a simple example of an iCalendar format:

```
BEGIN:VCALENDAR
Version:2.0
PRODID:-//hacksw/handcal//NONSGML v1.0//EN
BEGIN:VEVENT
DTSTART:20010101
DTEND:20010102
SUMMARY:First Party
END:VEVENT
BEGIN:VEVENT
DTSTART:20010104
DTEND:20010105
SUMMARY:Second Party
END:VEVENT
END:VCALENDAR
```

The VCALENDAR is the main component, which contains two VEVENT components.

The Components collection supports the following properties and methods:

Name	Description
Add	Adds a Component object to the collection and returns a reference to the newly created object.
Clear	Removes all objects in a collection.
Count	Returns the number of components in the collection.
Enumerate	Enumerates the components in the collection whose name matches the giving mask.
Item	Returns a specific Component of the Components collection, giving its index or name.
Remove	Removes a specific member from the Components collection, giving its index or name.

method Components.Add (Name as String)

Adds a Component object to the collection and returns a reference to the newly created object.

Type	Description
Name as String	A String expression that specifies the name of the component to be added.
Return	**Description**
Component	A Component object being created and added to the Components collection.

The Add method adds a Component object to the collection and returns a reference to the newly created object. The Name property specifies the name of the component. The control fires the AddComponent event once a new component is added, if the control's FireEvents property is True.

The following is a simple example of an iCalendar format:

BEGIN:**VCALENDAR**
Version:2.0
PRODID:-//hacksw/handcal//NONSGML v1.0//EN
BEGIN:**VEVENT**
DTSTART:20010101
DTEND:20010102
SUMMARY:First Party
END:VEVENT
BEGIN:**VEVENT**
DTSTART:20010104
DTEND:20010105
SUMMARY:Second Party
END:VEVENT
END:VCALENDAR

The VCALENDAR, VEVENT indicates the name of the component.

The following samples show how you can generate the above format:

VBA (MS Access, Excell...)

```
Set ICalendar1 = CreateObject("Exontrol.ICalendar.1")
With ICalendar1
	With .Content.Components.Add("VCALENDAR")
```

```
        With .Properties
            .Add "Version","2.0"
            .Add "PRODID","-//hacksw/handcal//NONSGML v1.0//EN"
        End With
        With .Components.Add("VEVENT").Properties
            .Add "DTSTART",#1/1/2001#
            .Add "DTEND",#1/2/2001#
            .Add "SUMMARY","First Party"
        End With
        With .Components.Add("VEVENT").Properties
            .Add "DTSTART",#1/4/2001#
            .Add "DTEND",#1/5/2001#
            .Add "SUMMARY","Second Party"
        End With
    End With
    Debug.Print( .Save )
End With
```

VB6

```
Set ICalendar1 = CreateObject("Exontrol.ICalendar.1")
With ICalendar1
    With .Content.Components.Add("VCALENDAR")
        With .Properties
            .Add "Version","2.0"
            .Add "PRODID","-//hacksw/handcal//NONSGML v1.0//EN"
        End With
        With .Components.Add("VEVENT").Properties
            .Add "DTSTART",#1/1/2001#
            .Add "DTEND",#1/2/2001#
            .Add "SUMMARY","First Party"
        End With
        With .Components.Add("VEVENT").Properties
            .Add "DTSTART",#1/4/2001#
            .Add "DTEND",#1/5/2001#
            .Add "SUMMARY","Second Party"
        End With
```

```
	End With
	Debug.Print( .Save )
End With
```

VB.NET

```
' Add 'exontrol.exicalendar.dll(ExICalendar.dll)' reference to your project.
Exicalendar1 = New exontrol.EXICALENDARLib.exicalendar()
With Exicalendar1
	With .Content.Components.Add("VCALENDAR")
		With .Properties
			.Add("Version","2.0")
			.Add("PRODID","-//hacksw/handcal//NONSGML v1.0//EN")
		End With
		With .Components.Add("VEVENT").Properties
			.Add("DTSTART",#1/1/2001#)
			.Add("DTEND",#1/2/2001#)
			.Add("SUMMARY","First Party")
		End With
		With .Components.Add("VEVENT").Properties
			.Add("DTSTART",#1/4/2001#)
			.Add("DTEND",#1/5/2001#)
			.Add("SUMMARY","Second Party")
		End With
	End With
	Debug.Print( .Save() )
End With
```

VB.NET for /COM

```
AxICalendar1 = CreateObject("Exontrol.ICalendar.1")
With AxICalendar1
	With .Content.Components.Add("VCALENDAR")
		With .Properties
			.Add("Version","2.0")
			.Add("PRODID","-//hacksw/handcal//NONSGML v1.0//EN")
		End With
		With .Components.Add("VEVENT").Properties
```

```
            .Add("DTSTART",#1/1/2001#)
            .Add("DTEND",#1/2/2001#)
            .Add("SUMMARY","First Party")
        End With
        With .Components.Add("VEVENT").Properties
            .Add("DTSTART",#1/4/2001#)
            .Add("DTEND",#1/5/2001#)
            .Add("SUMMARY","Second Party")
        End With
    End With
    Debug.Print( .Save() )
End With
```

C++

```
/*
    Includes the definition for CreateObject function like follows:

    #include <comdef.h>
    IUnknownPtr CreateObject( BSTR Object )
    {
        IUnknownPtr spResult;
        spResult.CreateInstance( Object );
        return spResult;
    };

*/
/*
    Copy and paste the following directives to your header file as
    it defines the namespace 'EXICALENDARLib' for the library: 'ICalendar 1.0 Type
Library'

    #import <ExICalendar.dll>
    using namespace EXICALENDARLib;
*/
EXICALENDARLib::IICalendarPtr spICalendar1 =
::CreateObject(L"Exontrol.ICalendar.1");
```

```
EXICALENDARLib::IComponentPtr var_Component = spICalendar1->GetContent()-
>GetComponents()->Add(L"VCALENDAR");
   EXICALENDARLib::IPropertiesPtr var_Properties = var_Component-
>GetProperties();
      var_Properties->Add(L"Version","2.0");
      var_Properties->Add(L"PRODID","-//hacksw/handcal//NONSGML v1.0//EN");
   EXICALENDARLib::IPropertiesPtr var_Properties1 = var_Component-
>GetComponents()->Add(L"VEVENT")->GetProperties();
      var_Properties1->Add(L"DTSTART",COleDateTime(2001,1,1,0,00,00).operator
DATE());
      var_Properties1->Add(L"DTEND",COleDateTime(2001,1,2,0,00,00).operator
DATE());
      var_Properties1->Add(L"SUMMARY","First Party");
   EXICALENDARLib::IPropertiesPtr var_Properties2 = var_Component-
>GetComponents()->Add(L"VEVENT")->GetProperties();
      var_Properties2->Add(L"DTSTART",COleDateTime(2001,1,4,0,00,00).operator
DATE());
      var_Properties2->Add(L"DTEND",COleDateTime(2001,1,5,0,00,00).operator
DATE());
      var_Properties2->Add(L"SUMMARY","Second Party");
OutputDebugStringW( spICalendar1->Save() );
```

C++ Builder

```
Exicalendarlib_tlb::IICalendarPtr ICalendar1 =
Variant::CreateObject(L"Exontrol.ICalendar.1");
Exicalendarlib_tlb::IComponentPtr var_Component = ICalendar1->Content-
>Components->Add(L"VCALENDAR");
   Exicalendarlib_tlb::IPropertiesPtr var_Properties = var_Component->Properties;
      var_Properties->Add(L"Version",TVariant("2.0"));
      var_Properties->Add(L"PRODID",TVariant("-//hacksw/handcal//NONSGML
v1.0//EN"));
   Exicalendarlib_tlb::IPropertiesPtr var_Properties1 = var_Component-
>Components->Add(L"VEVENT")->Properties;
      var_Properties1->Add(L"DTSTART",TVariant(TDateTime(2001,1,1).operator
double()));
```

```
        var_Properties1->Add(L"DTEND",TVariant(TDateTime(2001,1,2).operator
double()));
        var_Properties1->Add(L"SUMMARY",TVariant("First Party"));
    Exicalendarlib_tlb::IPropertiesPtr var_Properties2 = var_Component-
>Components->Add(L"VEVENT")->Properties;
        var_Properties2->Add(L"DTSTART",TVariant(TDateTime(2001,1,4).operator
double()));
        var_Properties2->Add(L"DTEND",TVariant(TDateTime(2001,1,5).operator
double()));
        var_Properties2->Add(L"SUMMARY",TVariant("Second Party"));
OutputDebugString( ICalendar1->Save() );
```

C#

```
// Add 'exontrol.exicalendar.dll(ExICalendar.dll)' reference to your project.
exontrol.EXICALENDARLib.exicalendar exicalendar1 = new
exontrol.EXICALENDARLib.exicalendar();
exontrol.EXICALENDARLib.Component var_Component =
exicalendar1.Content.Components.Add("VCALENDAR");
    exontrol.EXICALENDARLib.Properties var_Properties = var_Component.Properties;
        var_Properties.Add("Version","2.0");
        var_Properties.Add("PRODID","-//hacksw/handcal//NONSGML v1.0//EN");
    exontrol.EXICALENDARLib.Properties var_Properties1 =
var_Component.Components.Add("VEVENT").Properties;

var_Properties1.Add("DTSTART",Convert.ToDateTime("1/1/2001",System.Globalization.
US")));

var_Properties1.Add("DTEND",Convert.ToDateTime("1/2/2001",System.Globalization.C
US")));
        var_Properties1.Add("SUMMARY","First Party");
    exontrol.EXICALENDARLib.Properties var_Properties2 =
var_Component.Components.Add("VEVENT").Properties;

var_Properties2.Add("DTSTART",Convert.ToDateTime("1/4/2001",System.Globalization.
US")));
```

```
var_Properties2.Add("DTEND",Convert.ToDateTime("1/5/2001",System.Globalization.Cu
US")));
        var_Properties2.Add("SUMMARY","Second Party");
System.Diagnostics.Debug.Print( exicalendar1.Save() );
```

JScript/JavaScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:D6C87100-38E2-4ABB-8AC2-4C0097AEE2D6"
id="ICalendar1"></OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Component = ICalendar1.Content.Components.Add("VCALENDAR");
        var var_Properties = var_Component.Properties;
            var_Properties.Add("Version","2.0");
            var_Properties.Add("PRODID","-//hacksw/handcal//NONSGML v1.0//EN");
        var var_Properties1 = var_Component.Components.Add("VEVENT").Properties;
            var_Properties1.Add("DTSTART","1/1/2001");
            var_Properties1.Add("DTEND","1/2/2001");
            var_Properties1.Add("SUMMARY","First Party");
        var var_Properties2 = var_Component.Components.Add("VEVENT").Properties;
            var_Properties2.Add("DTSTART","1/4/2001");
            var_Properties2.Add("DTEND","1/5/2001");
            var_Properties2.Add("SUMMARY","Second Party");
    alert( ICalendar1.Save() );
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:D6C87100-38E2-4ABB-8AC2-4C0097AEE2D6"
```

```
id="ICalendar1"></OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
   With ICalendar1
      With .Content.Components.Add("VCALENDAR")
         With .Properties
            .Add "Version","2.0"
            .Add "PRODID","-//hacksw/handcal//NONSGML v1.0//EN"
         End With
         With .Components.Add("VEVENT").Properties
            .Add "DTSTART",#1/1/2001#
            .Add "DTEND",#1/2/2001#
            .Add "SUMMARY","First Party"
         End With
         With .Components.Add("VEVENT").Properties
            .Add "DTSTART",#1/4/2001#
            .Add "DTEND",#1/5/2001#
            .Add "SUMMARY","Second Party"
         End With
      End With
      alert( .Save )
   End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
EXICALENDARLib.ICalendar axICalendar1 = new EXICALENDARLib.ICalendar();
EXICALENDARLib.Component var_Component =
axICalendar1.Content.Components.Add("VCALENDAR");
   EXICALENDARLib.Properties var_Properties = var_Component.Properties;
      var_Properties.Add("Version","2.0");
      var_Properties.Add("PRODID","-//hacksw/handcal//NONSGML v1.0//EN");
   EXICALENDARLib.Properties var_Properties1 =
var_Component.Components.Add("VEVENT").Properties;

var_Properties1.Add("DTSTART",Convert.ToDateTime("1/1/2001",System.Globalization.
US")));

var_Properties1.Add("DTEND",Convert.ToDateTime("1/2/2001",System.Globalization.Cu
US")));
	var_Properties1.Add("SUMMARY","First Party");
    EXICALENDARLib.Properties var_Properties2 =
var_Component.Components.Add("VEVENT").Properties;

var_Properties2.Add("DTSTART",Convert.ToDateTime("1/4/2001",System.Globalization.
US")));

var_Properties2.Add("DTEND",Convert.ToDateTime("1/5/2001",System.Globalization.Cu
US")));
	var_Properties2.Add("SUMMARY","Second Party");
System.Diagnostics.Debug.Print( axICalendar1.Save() );
```

X++ (Dynamics Ax 2009)

```
public void init()
{
    COM
com_Component,com_Component1,com_Components,com_Properties,com_Propertie

    anytype
exicalendar1,var_Component,var_Component1,var_Components,var_Properties,var_Pro

    ;

    super();

    // Add 'exicalendar.dll(ExICalendar.dll)' reference to your project.
    exicalendar1 = COM::createFromObject(new EXICALENDARLib.exicalendar());
com_exicalendar1 = exicalendar1;
```

```
    var_Component =
COM::createFromObject(com_exicalendar1.Content().Components()).Add("VCALENDA
 com_Component = var_Component;
        var_Properties = com_Component.Properties(); com_Properties = var_Properties;
            com_Properties.Add("Version","2.0");
            com_Properties.Add("PRODID","-//hacksw/handcal//NONSGML v1.0//EN");
        var_Components = COM::createFromObject(com_Component.Components());
com_Components = var_Components;
        var_Component1 = COM::createFromObject(com_Components).Add("VEVENT");
com_Component1 = var_Component1;
        var_Properties1 = com_Component1.Properties(); com_Properties1 =
var_Properties1;

com_Properties1.Add("DTSTART",COMVariant::createFromDate(str2Date("1/1/2001",21


com_Properties1.Add("DTEND",COMVariant::createFromDate(str2Date("1/2/2001",213

            com_Properties1.Add("SUMMARY","First Party");
        var_Components = COM::createFromObject(com_Component.Components());
com_Components = var_Components;
        var_Component1 = COM::createFromObject(com_Components).Add("VEVENT");
com_Component1 = var_Component1;
        var_Properties2 = com_Component1.Properties(); com_Properties2 =
var_Properties2;

com_Properties2.Add("DTSTART",COMVariant::createFromDate(str2Date("1/4/2001",21


com_Properties2.Add("DTEND",COMVariant::createFromDate(str2Date("1/5/2001",213

            com_Properties2.Add("SUMMARY","Second Party");
    print( com_exicalendar1.Save() );
}
```

Delphi 8 (.NET only)

```
AxICalendar1 :=
(ComObj.CreateComObject(ComObj.ProgIDToClassID('Exontrol.ICalendar.1')) as
EXICALENDARLib.ICalendar);
with AxICalendar1 do
begin
  with Content.Components.Add('VCALENDAR') do
  begin
    with Properties do
    begin
      Add('Version','2.0');
      Add('PRODID','-//hacksw/handcal//NONSGML v1.0//EN');
    end;
    with Components.Add('VEVENT').Properties do
    begin
      Add('DTSTART','1/1/2001');
      Add('DTEND','1/2/2001');
      Add('SUMMARY','First Party');
    end;
    with Components.Add('VEVENT').Properties do
    begin
      Add('DTSTART','1/4/2001');
      Add('DTEND','1/5/2001');
      Add('SUMMARY','Second Party');
    end;
  end;
  OutputDebugString( Save() );
end
```

Delphi (standard)

```
ICalendar1 :=
(IUnknown(ComObj.CreateComObject(ComObj.ProgIDToClassID('Exontrol.ICalendar.1'))
 as EXICALENDARLib_TLB.ICalendar);
with ICalendar1 do
begin
  with Content.Components.Add('VCALENDAR') do
  begin
		with Properties do
		begin
			Add('Version','2.0');
			Add('PRODID','-//hacksw/handcal//NONSGML v1.0//EN');
		end;
		with Components.Add('VEVENT').Properties do
		begin
			Add('DTSTART','1/1/2001');
			Add('DTEND','1/2/2001');
			Add('SUMMARY','First Party');
		end;
		with Components.Add('VEVENT').Properties do
		begin
			Add('DTSTART','1/4/2001');
			Add('DTEND','1/5/2001');
			Add('SUMMARY','Second Party');
		end;
	end;
	OutputDebugString( Save() );
end
```

VFP

```
thisform.ICalendar1 = CreateObject("Exontrol.ICalendar.1")
with thisform.ICalendar1
	with .Content.Components.Add("VCALENDAR")
		with .Properties
			.Add("Version","2.0")
			.Add("PRODID","-//hacksw/handcal//NONSGML v1.0//EN")
		endwith
		with .Components.Add("VEVENT").Properties
			.Add("DTSTART",{^2001-1-1})
			.Add("DTEND",{^2001-1-2})
			.Add("SUMMARY","First Party")
		endwith
		with .Components.Add("VEVENT").Properties
			.Add("DTSTART",{^2001-1-4})
			.Add("DTEND",{^2001-1-5})
			.Add("SUMMARY","Second Party")
		endwith
	endwith
	DEBUGOUT( .Save )
endwith
```

dBASE Plus

```
local oICalendar,var_Component,var_Properties,var_Properties1,var_Properties2

oICalendar = new OleAutoClient("Exontrol.ICalendar.1")

var_Component = oICalendar.Content.Components.Add("VCALENDAR")
	var_Properties = var_Component.Properties
		var_Properties.Add("Version","2.0")
		var_Properties.Add("PRODID","-//hacksw/handcal//NONSGML v1.0//EN")
	var_Properties1 = var_Component.Components.Add("VEVENT").Properties
		var_Properties1.Add("DTSTART","01/01/2001")
		var_Properties1.Add("DTEND","01/02/2001")
		var_Properties1.Add("SUMMARY","First Party")
	var_Properties2 = var_Component.Components.Add("VEVENT").Properties
		var_Properties2.Add("DTSTART","01/04/2001")
		var_Properties2.Add("DTEND","01/05/2001")
		var_Properties2.Add("SUMMARY","Second Party")
? oICalendar.Save()
```

XBasic (Alpha Five)

```
Dim oICalendar as P
Dim var_Component as P
Dim var_Properties as P
Dim var_Properties1 as P
Dim var_Properties2 as P

oICalendar = OLE.Create("Exontrol.ICalendar.1")
```

```
var_Component = oICalendar.Content.Components.Add("VCALENDAR")
   var_Properties = var_Component.Properties
      var_Properties.Add("Version","2.0")
      var_Properties.Add("PRODID","-//hacksw/handcal//NONSGML v1.0//EN")
   var_Properties1 = var_Component.Components.Add("VEVENT").Properties
      var_Properties1.Add("DTSTART",{01/01/2001})
      var_Properties1.Add("DTEND",{01/02/2001})
      var_Properties1.Add("SUMMARY","First Party")
   var_Properties2 = var_Component.Components.Add("VEVENT").Properties
      var_Properties2.Add("DTSTART",{01/04/2001})
      var_Properties2.Add("DTEND",{01/05/2001})
      var_Properties2.Add("SUMMARY","Second Party")
? oICalendar.Save()
```

Visual Objects

```
local var_Component as IComponent
local var_Properties,var_Properties1,var_Properties2 as IProperties

oDCOCX_Exontrol1 := IICalendar{"Exontrol.ICalendar.1"}
var_Component := oDCOCX_Exontrol1:Content:Components:Add("VCALENDAR")
   var_Properties := var_Component:Properties
      var_Properties:Add("Version","2.0")
      var_Properties:Add("PRODID","-//hacksw/handcal//NONSGML v1.0//EN")
   var_Properties1 := var_Component:Components:Add("VEVENT"):Properties
      var_Properties1:Add("DTSTART",SToD("20010101"))
      var_Properties1:Add("DTEND",SToD("20010102"))
      var_Properties1:Add("SUMMARY","First Party")
   var_Properties2 := var_Component:Components:Add("VEVENT"):Properties
      var_Properties2:Add("DTSTART",SToD("20010104"))
      var_Properties2:Add("DTEND",SToD("20010105"))
      var_Properties2:Add("SUMMARY","Second Party")
OutputDebugString(String2Psz( oDCOCX_Exontrol1:Save() ))
```

PowerBuilder

```
OleObject oICalendar,var_Component,var_Properties,var_Properties1,var_Properties2

oICalendar = CREATE OLEObject
oICalendar.ConnectToNewObject("Exontrol.ICalendar.1")

var_Component = oICalendar.Content.Components.Add("VCALENDAR")
   var_Properties = var_Component.Properties
      var_Properties.Add("Version","2.0")
      var_Properties.Add("PRODID","-//hacksw/handcal//NONSGML v1.0//EN")
   var_Properties1 = var_Component.Components.Add("VEVENT").Properties
      var_Properties1.Add("DTSTART",2001-01-01)
      var_Properties1.Add("DTEND",2001-01-02)
      var_Properties1.Add("SUMMARY","First Party")
   var_Properties2 = var_Component.Components.Add("VEVENT").Properties
      var_Properties2.Add("DTSTART",2001-01-04)
      var_Properties2.Add("DTEND",2001-01-05)
      var_Properties2.Add("SUMMARY","Second Party")
MessageBox("Information",string( oICalendar.Save() ))
```

Visual DataFlex

```
Procedure OnCreate
   Forward Send OnCreate
   Variant oComICalendar1
   Get ComCreateObject "Exontrol.ICalendar.1" to oComICalendar1

   Variant voComponent
   Get ComContent to voComponent
   Handle hoComponent
   Get Create (RefClass(cComComponent)) to hoComponent
   Set pvComObject of hoComponent to voComponent
      Variant voComponents
      Get ComComponents of hoComponent to voComponents
      Handle hoComponents
      Get Create (RefClass(cComComponents)) to hoComponents
      Set pvComObject of hoComponents to voComponents
```

```
Variant voComponent1
Get ComAdd of hoComponents "VCALENDAR" to voComponent1
Handle hoComponent1
Get Create (RefClass(cComComponent)) to hoComponent1
Set pvComObject of hoComponent1 to voComponent1
    Variant voProperties
    Get ComProperties of hoComponent1 to voProperties
    Handle hoProperties
    Get Create (RefClass(cComProperties)) to hoProperties
    Set pvComObject of hoProperties to voProperties
        Get ComAdd of hoProperties "Version" "2.0" to Nothing
        Get ComAdd of hoProperties "PRODID" "-//hacksw/handcal//NONSGML
v1.0//EN" to Nothing
    Send Destroy to hoProperties
    Variant voComponents1
    Get ComComponents of hoComponent1 to voComponents1
    Handle hoComponents1
    Get Create (RefClass(cComComponents)) to hoComponents1
    Set pvComObject of hoComponents1 to voComponents1
        Variant voComponent2
        Get ComAdd of hoComponents1 "VEVENT" to voComponent2
        Handle hoComponent2
        Get Create (RefClass(cComComponent)) to hoComponent2
        Set pvComObject of hoComponent2 to voComponent2
            Variant voProperties1
            Get ComProperties of hoComponent2 to voProperties1
            Handle hoProperties1
            Get Create (RefClass(cComProperties)) to hoProperties1
            Set pvComObject of hoProperties1 to voProperties1
                Get ComAdd of hoProperties1 "DTSTART" "1/1/2001" to Nothing
                Get ComAdd of hoProperties1 "DTEND" "1/2/2001" to Nothing
                Get ComAdd of hoProperties1 "SUMMARY" "First Party" to Nothing
            Send Destroy to hoProperties1
        Send Destroy to hoComponent2
    Send Destroy to hoComponents1
    Variant voComponents2
    Get ComComponents of hoComponent1 to voComponents2
```

```
            Handle hoComponents2
            Get Create (RefClass(cComComponents)) to hoComponents2
            Set pvComObject of hoComponents2 to voComponents2
                Variant voComponent3
                Get ComAdd of hoComponents2 "VEVENT" to voComponent3
                Handle hoComponent3
                Get Create (RefClass(cComComponent)) to hoComponent3
                Set pvComObject of hoComponent3 to voComponent3
                    Variant voProperties2
                    Get ComProperties of hoComponent3 to voProperties2
                    Handle hoProperties2
                    Get Create (RefClass(cComProperties)) to hoProperties2
                    Set pvComObject of hoProperties2 to voProperties2
                        Get ComAdd of hoProperties2 "DTSTART" "1/4/2001" to Nothing
                        Get ComAdd of hoProperties2 "DTEND" "1/5/2001" to Nothing
                        Get ComAdd of hoProperties2 "SUMMARY" "Second Party" to
Nothing
                    Send Destroy to hoProperties2
                Send Destroy to hoComponent3
            Send Destroy to hoComponents2
        Send Destroy to hoComponent1
    Send Destroy to hoComponents
  Send Destroy to hoComponent
  Showln (ComSave(Self))
End_Procedure
```

XBase++

```
#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
  LOCAL oForm
  LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
  LOCAL oICalendar
  LOCAL oComponent
  LOCAL oProperties,oProperties1,oProperties2
```

```
   oForm := XbpDialog():new( AppDesktop() )
   oForm:drawingArea:clipChildren := .T.
   oForm:create( ,,{100,100}, {640,480},, .F. )
   oForm:close  := {|| PostAppEvent( xbeP_Quit )}

   oICalendar := XbpActiveXControl():new( oForm:drawingArea )
   oICalendar:CLSID  := "Exontrol.ICalendar.1" /*{D6C87100-38E2-4ABB-8AC2-
4C0097AEE2D6}*/
   oICalendar:create(,, {10,60},{610,370} )

      oComponent := oICalendar:Content():Components():Add("VCALENDAR")
         oProperties := oComponent:Properties()
            oProperties:Add("Version","2.0")
            oProperties:Add("PRODID","-//hacksw/handcal//NONSGML v1.0//EN")
         oProperties1 := oComponent:Components():Add("VEVENT"):Properties()
            oProperties1:Add("DTSTART","01/01/2001")
            oProperties1:Add("DTEND","01/02/2001")
            oProperties1:Add("SUMMARY","First Party")
         oProperties2 := oComponent:Components():Add("VEVENT"):Properties()
            oProperties2:Add("DTSTART","01/04/2001")
            oProperties2:Add("DTEND","01/05/2001")
            oProperties2:Add("SUMMARY","Second Party")
      DevOut( oICalendar:Save() )

   oForm:Show()
   DO WHILE nEvent != xbeP_Quit
      nEvent := AppEvent( @mp1, @mp2, @oXbp )
      oXbp:handleEvent( nEvent, mp1, mp2 )
   ENDDO
RETURN
```

method Components.Clear ()

Removes all objects in a collection.

Type	Description

The Clear method clears the components collection. The Clear method of the Component object, empties the component, by removing the name, properties and components. Use the Remove method to remove a component from the Components collection.

property Components.Count as Long

Returns the number of components in the collection.

Type	Description
Long	A Long expression that specifies the number of Component objects in the Components collection.

The Count property indicates the number of components in the collection. The Item property accesses the Component giving its index. The Item / Count properties can be used to enumerate the Components collection as well as **for each** statement. The Enumerate method enumerates the components in the collection whose name matches the giving mask. The Remove method removes a component from the Components collection. The Clear method clears the Components collection.

The following code enumerates the components of the root component:

```
Dim c As Variant
For Each c In ICalendar1.Root.Components
    Debug.Print c.Name
Next
```

and it's equivalent with the following snippet:

```
Dim i As Long
With ICalendar1.Root.Components
    For i = 0 To .Count - 1
        Debug.Print .Item(i).Name
    Next
End With
```

property Components.Enumerate (Mask as String) as Variant

Enumerates the components in the collection whose name matches the giving mask.

Type	Description
Mask as String	A String expression that specifies the mask of the components to be requested. The Mask parameter can include: • '?' for any single character • '*' for zero or more occurrences of any character • '#' for any digit character
Variant	A safe array of Component objects whose name matches the giving mask.

Use the Enumerate property to enumerate the components giving a mask. The Item / Count properties can be used to enumerate the Components collection as well as **for each** statement.

The following code enumerates the components of the root component, that starts with VEV:

```
Dim c As Variant
For Each c In ICalendar1.Root.Components.Enumerate("VEV*")
   Debug.Print "Name " & c.Name
Next
```

property Components.Item (Index as Variant) as Component

Returns a specific Component of the Components collection, giving its index or name.

Type	Description
Index as Variant	A Long expression that specifies the index of the Component to be requested, or a String expression that specifies the name of the Component to be requested.
Component	A Component object being requested.

The Item property accesses the Component giving its index / 0 - based. The Count property indicates the number of components in the collection. The Item / Count properties can be used to enumerate the Components collection as well as **for each** statement. The Enumerate method enumerates the components in the collection whose name matches the giving mask. The Remove method removes a component from the Components collection. The Clear method clears the Components collection.

The following code enumerates the components of the root component:

```
Dim c As Variant
For Each c In ICalendar1.Root.Components
    Debug.Print c.Name
Next
```

and it's equivalent with the following snippet:

```
Dim i As Long
With ICalendar1.Root.Components
    For i = 0 To .Count - 1
        Debug.Print .Item(i).Name
    Next
End With
```

method Components.Remove (Index as Variant)

Removes a specific member from the Components collection, giving its index or name.

Type	Description
Index as Variant	A Long expression that specifies the index of the Component to be requested, or a String expression that specifies the name of the Component to be requested.

The Remove method removes a component from the Components collection. The Clear method clears the Components collection. The Item property accesses the Component giving its index / 0 - based. The Count property indicates the number of components in the collection. The Item / Count properties can be used to enumerate the Components collection as well as **for each** statement. The Enumerate method enumerates the components in the collection whose name matches the giving mask.

ICalendar object

The ExICalendar library implements the ICalendar data format, according with Internet Calendaring and Scheduling Core Object Specification, RFC 5545. The iCalendar data format represents exchanging calendaring and scheduling information such as events, to-dos, journal entries, and free/busy information, independent of any particular calendar service or protocol. The iCalendar format is suitable as an exchange format between applications or systems. The format is defined in terms of a MIME content type. This will enable the object to be exchanged using several transports, including but not limited to SMTP, HTTP, a file system, desktop interactive protocols such as the use of a memory-based clipboard or drag/drop interactions, point-to-point asynchronous communication, and so on.

The hierarchy of objects/properties of the ICalendar object is:

```
EXICALENDARLib.ICalendar
  "Content" -> EXICALENDARLib.Component
  "Root" -> EXICALENDARLib.Component
EXICALENDARLib.Component
  "Components" -> EXICALENDARLib.Components
  "Parent" -> EXICALENDARLib.Component
  "Properties" -> EXICALENDARLib.Properties
EXICALENDARLib.Components
  "Add(String)" -> EXICALENDARLib.Component
  "Item(Variant)" -> EXICALENDARLib.Component
EXICALENDARLib.Properties
  "Add(String,Variant)" -> EXICALENDARLib.Property
  "Item(Variant)" -> EXICALENDARLib.Property
EXICALENDARLib.Property
  "Component" -> EXICALENDARLib.Component
  "Parameters" -> EXICALENDARLib.Parameters
EXICALENDARLib.Parameters
  "Add(String,Variant)" -> EXICALENDARLib.Parameter
  "Item(Variant)" -> EXICALENDARLib.Parameter
EXICALENDARLib.Parameter
  "Property" -> EXICALENDARLib.Property
```

The ICalendar object supports the following properties and methods:

Name	Description

AttachTemplate	Attaches a script to the current object, including the events, from a string, file, a safe array of bytes.
Content	Retrieves the object's content such as properties, and components.
Debug	Gets debugging information.
EventParam	Retrieves or sets a value that indicates the current's event parameter.
ExecuteTemplate	Executes a template and returns the result.
FireEvents	Specifies whether the control fires the events.
fromICalendar	Converts the giving ICalendar expression to a VARIANT expression. For instance, fromICalendar("P15DT12H", exPropertyTypeDuration) returns 15.5 which indicates 15 days and 12 hours.
Load	Loads and parses the iCalendar format from giving text.
LoadFile	Loads a file.
LoadFileFromUnicode	Loads from an UNICODE file.
RecurAll	Returns all recurrences of the specified rule, as a safe array of dates. .
RecurAllAsString	Returns all recurrences of the specified rule, as string (dates are separated by new line) .
RecurAllMethod	Specifies the way the component gets the occurrences of the recurrence rule (RecurAll method).
RecurAllTime	Specifies the time in milliseconds that took to perform the last RecurAll call.
RecurCheck	Evaluates the date using recurrence rules, and returns 1 if the Date matches the recurrence rule, 0 if not, or a negative number if any error occurs.
RecurPartValue	Returns the value of the giving part of the recur expression.
RecurRange	Returns all occurrences between start and and of the specified rule, as a safe array of dates.
RecurRangeAsString	Returns all occurrences between start and and of the specified rule, as string (dates are separated by new line).
Root	Retrieves the root component of the content (first component).

RuntimeKey	Specifies a runtime key to be used for the component.
Save	Saves the content as iCalendar format.
SaveFile	Saves the control's content to a file.
SaveFileAsUnicode	Saves the control's content to file as UNICODE.
Template	Specifies the control's template.
TemplateDef	Defines inside variables for the next Template/ExecuteTemplate call.
TemplatePut	Defines inside variables for the next Template/ExecuteTemplate call.
toICalendar	Converts the giving VARIANT expression to ICalendar format. For instance, toICalendar(CSng(15.5), exPropertyTypeDuration) returns "P15DT12H".
UserData	Indicates any extra data associated with the control / root.
valuesFromICalendar	Extracts all values or specified value of the giving value in ICalendar format. For instance valuesFromICalendar("P15DT12H", exPropertyTypeDuration, "Duration") returns 15.5, which indicates the duration in days, hours, minutes, seconds as a DATE expression.
valuesToICalendar	Converts the values to a value of ICalendar format. For instance, valuesToICalendar("Duration=15.5",exPropertyTypeDuratior returns "P15DT12H", which indicates 15 days and 12 hours.
Version	Retrieves the control's version.

method ICalendar.AttachTemplate (Template as Variant)

Attaches a script to the current object, including the events, from a string, file, a safe array of bytes.

Type	Description
Template as Variant	A string expression that specifies the Template to execute.

The AttachTemplate/x-script code is a simple way of calling control/object's properties, methods/events using strings. The AttachTemplate features allows you to attach a x-script code to the component. The AttachTemplate method executes x-script code (including events), from a string, file or a safe array of bytes. This feature allows you to run any x-script code for any configuration of the component /COM, /NET or /WPF. Exontrol owns the x-script implementation in its easiest form and it does not require any VB engine or whatever to get executed. The x-script code can be converted to several programming languages using the eXHelper tool.

The following sample opens the Windows Internet Explorer once the user clicks the control (/COM version):

```
AttachTemplate("handle Click(){ CreateObject(`internetexplorer.application`){ Visible =
True; Navigate(`https://www.exontrol.com`) } } ")
```

This script is equivalent with the following VB code:

```
Private Sub ICalendar1_Click()
    With CreateObject("internetexplorer.application")
        .Visible = True
        .Navigate ("https://www.exontrol.com")
    End With
End Sub
```

The AttachTemplate/x-script syntax in BNF notation is defined like follows:

```
<x-script> := <lines>
<lines> := <line>[<eol> <lines>] | <block>
<block> := <call> [<eol>] { [<eol>] <lines> [<eol>] } [<eol>]
<eol> := ";" | "\r\n"
<line> := <dim> | <createobject> | <call> | <set> | <comment> | <handle>[<eol>]{[<eol>]
<lines>[<eol>]}[<eol>]
<dim> := "DIM" <variables>
<variables> := <variable> [, <variables>]
```

```
<variable> := "ME" | <identifier>
<createobject> := "CREATEOBJECT(`"<type>"`)"
<call> := <variable> | <property> | <variable>"."<property> | <createobject>"."<property>
<property> := [<property>"."]<identifier>["("<parameters>")"]
<set> := <call> "=" <value>
<property> := <identifier> | <identifier>"("[<parameters>]")"
<parameters> := <value> [","<parameters>]
<value> := <boolean> | <number> | <color> | <date> | <string> | <createobject> | <call>
<boolean> := "TRUE" | "FALSE"
<number> := "0X"<hexa> | ["-"]<integer>["."<integer>]
<digit10> := 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<digit16> := <digit10> | A | B | C | D | E | F
<integer> := <digit10>[<integer>]
<hexa> := <digit16>[<hexa>]
<color> := "RGB("<integer>","<integer>","<integer>")"
<date> := "#"<integer>"/"<integer>"/"<integer>" "[<integer>":"<integer>":"<integer>"]"#"
<string> := "'"<text>"'" | "`"<text>"`"
<comment> := "'"<text>
<handle> := "handle " <event>
<event> := <identifier>"("[<eparameters>]")"
<eparameters> := <eparameter> [","<eparameters>]
<parameters> := <identifier>
```

where:

<identifier> indicates an identifier of the variable, property, method or event, and should start with a letter.
<type> indicates the type the CreateObject function creates, as a progID for /COM version or the assembly-qualified name of the type to create for /NET or /WPF version
<text> any string of characters

The Template or x-script is composed by lines of instructions. Instructions are separated by "\n\r" (newline characters) or ";" character.

The advantage of the AttachTemplate relative to Template / ExecuteTemplate is that the AttachTemplate can add handlers to the control events.

property ICalendar.Content as Component

Retrieves the object's content such as properties, and components.

Type	Description
Component	A Component object that holds the properties and other components.

The Content property gets the control's content (properties and components). The Root property gets the root component of the content. The Properties property gets the Properties collection of the current component. The Components property returns the collection of the Components of the current component. The Load / LoadFile / LoadFileFromUnicode method loads properties and components from iCalendar format. The StartLoad / EndLoad events notifies that the Content is starting / ending to be loaded. The Name property of the Component returns the name of the current component. The Content.Name property always returns "Exontrol.ICalendar", while the Root.Name property returns the name of the first component being found in the content (VCALENDAR). The Save property gives the iCalendar format of the current content.

For instance, having the following iCalendar format:

```
BEGIN:VCALENDAR
VERSION:2.0
PRODID:-//hacksw/handcal//NONSGML v1.0//EN
BEGIN:VEVENT
UID:19970610T172345Z-AF23B2@example.com
DTSTAMP:19970610T172345Z
DTSTART:19970714T170000Z
DTEND:19970715T040000Z
SUMMARY:Bastille Day Party
END:VEVENT
END:VCALENDAR
```

The Content.Components includes the VCALENDAR object, while the Root property refers to the VCALENDAR object directly.

property ICalendar.Debug ([Reserved as Variant]) as String

Gets debugging information.

Type	Description
Reserved as Variant	A VARIANT expression.
String	A String expression that gets debugging information.

For internal use only.

property ICalendar.EventParam(Parameter as Long) as Variant

Retrieves or sets a value that indicates the current's event parameter.

Type	Description
Parameter as Long	A long expression that indicates the index of the parameter being requested ie 0 means the first parameter, 1 means the second, and so on. • If -1 is used the EventParam property retrieves the number of parameters. • If -2, the EventParam gives full information about the event, such as name, identifier, and parameters. Accessing an not-existing parameter produces an OLE error, such as invalid pointer (E_POINTER)
Variant	A VARIANT expression that specifies the parameter's value.

The EventParam method is provided to allow changing the event's parameters passed by reference, even if your environment does not support changing it (uniPaas 1.5 (formerly known as eDeveloper), DBase, and so on). For instance, Unipaas event-handling logic cannot update ActiveX control variables by updating the received arguments. The EventParam(0) retrieves the value of the first parameter of the event, while the EventParam(1) = 0, changes the value of the second parameter to 0 (the operation is successfully, only if the parameter is passed by reference). The EventParam(-1) retrieves the number of the parameters of the current event.

Let's take the event "event KeyDown (**KeyCode** as Integer, ByVal Shift as Integer)", where the KeyCode parameter is passed by reference. For instance, put the KeyCode parameter on 0, and the arrow keys are disabled while the control has the focus.

In most languages you will type something like:

```
Private Sub Control1_KeyDown(KeyCode As Integer, Shift As Integer)
  KeyCode = 0
End Sub
```

In case your environment does not support events with parameters by reference, you can use a code like follows:

```
Private Sub Control1_KeyDown(KeyCode As Integer, Shift As Integer)
  Control1.EventParam(0) = 0
```

```
End Sub
```

In other words, the EventParam property provides the parameters of the current event for reading or writing access, even if your environment does not allow changing parameters by reference.

Calling the EventParam property outside of an event produces an OLE error, such as pointer invalid, as its scope was designed to be used only during events.

method ICalendar.ExecuteTemplate (Template as String)

Executes a template and returns the result.

Type	Description
Template as String	A Template string being executed
Return	**Description**
Variant	A Variant expression that indicates the result after executing the Template.

Use the ExecuteTemplate property to returns the result of executing a template file. Use the Template property to execute a template without returning any result. Use the ExecuteTemplate property to execute code by passing instructions as a string (template string).

For instance, the following sample retrieves the beginning date (as string) for the default bar in the first visible item:

```
Debug.Print ICalendar1.ExecuteTemplate("Items.ItemBar(FirstVisibleItem(),``,1)")
```

Most of our UI components provide a Template page that's accessible in design mode. No matter what programming language you are using, you can have a quick view of the component's features using the WYSWYG Template editor.

- Place the control to your form or dialog.
- Locate the Properties item, in the control's context menu, in design mode. If your environment doesn't provide a Properties item in the control's context menu, please try to locate in the Properties browser.
- Click it, and locate the Template page.
- Click the Help button. In the left side, you will see the component, in the right side, you will see a x-script code that calls methods and properties of the control.

The control's Template page helps user to initialize the control's look and feel in design mode, using the x-script language that's easy and powerful. The Template page displays the control on the left side of the page. On the right side of the Template page, a simple editor is displayed where user writes the initialization code. The control's look and feel is automatically updated as soon as the user types new instructions. The Template script is saved to the container persistence (when Apply button is pressed), and it is executed when the control is initialized at runtime. Any component that provides a WYSWYG Template page, provides a Template property. The Template property executes code from a string (template string).

The Template or x-script is composed by lines of instructions. Instructions are separated by

"\n\r" (newline characters) or ";" character. The ; character may be available only for newer versions of the components.

An x-script instruction/line can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas. (Sample: Dim h, h1, h2)*
- variable **=** property(list of arguments) *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name of the object in the context. The "list or arguments" may include variables or values separated by commas. (Sample: h = InsertItem(0,"New Child"))*
- property**(** list of arguments **)** = value *Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- method**(** list of arguments **)** *Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- **{** *Beginning the object's context. The properties or methods called between { and } are related to the last object returned by the property prior to { declaration.*
- **}** *Ending the object's context*
- object**.** property(list of arguments)**.**property(list of arguments).... *The .(dot) character splits the object from its property. For instance, the Columns.Add("Column1").HeaderBackColor = RGB(255,0,0), adds a new column and changes the column's header back color.*

The x-script may uses constant expressions as follow:

- *boolean* expression with possible values as *True* or *False*
- *numeric* expression may starts with 0x which indicates a hexa decimal representation, else it should starts with digit, or +/- followed by a digit, and . is the decimal separator. *Sample: 13 indicates the integer 13, or 12.45 indicates the double expression 12,45*
- *date* expression is delimited by **#** character in the format #mm/dd/yyyy hh:mm:ss#. *Sample: #31/12/1971# indicates the December 31, 1971*
- *string* expression is delimited by **"** or **`** characters. If using the ` character, please make sure that it is different than ' which allows adding comments inline. *Sample: "text" indicates the string text.*

Also , the template or x-script code may support general functions as follows:

- **Me** *property indicates the original object.*
- **RGB(**R,G,B**)** *property retrieves an RGB value, where the R, G, B are byte values that indicates the R G B values for the color being specified. For instance, the following code changes the control's background color to red: BackColor = RGB(255,0,0)*
- **LoadPicture(**file**)** *property loads a picture from a file or from BASE64 encoded strings, and returns a Picture object required by the picture properties.*

- **CreateObject(**progID**)** *property creates and retrieves a single uninitialized object of the class associated with a specified program identifier.*

property ICalendar.FireEvents as Boolean

Specifies whether the control fires the events.

Type	Description
Boolean	A Boolean expression that specifies whether the control fires / freezes the events.

By default, the FireEvents property is True, so the control fires the events.

The order of the events for Load, LoadFile, LoadFileFromUnicode methods is:

- StartLoad event, notifies that the Load, LoadFile, LoadFileFromUnicode methods begins
- AddComponent(NewComponent) event, notifies that a new Component object is added to the Components collection.
- AddProperty(NewPropery) event, notifies that a new Property object is added to the Properties collection. The Component property of the Property object specifies owner/parent component of the newly added property.
- AddParameter(NewParameter) event, notifies that a new Parameter object is added to the Parameters collection. The Property property of the Parameter object specifies owner/parent property of the newly added parameter.
- EndLoad event, notifies that the Load, LoadFile, LoadFileFromUnicode methods ends.

property ICalendar.fromICalendar (Value as String, Type as PropertyTypeEnum) as Variant

Converts the giving ICalendar expression to a VARIANT expression. For instance, fromICalendar("P15DT12H", exPropertyTypeDuration) returns 15.5 which indicates 15 days and 12 hours.

Type	Description
Value as String	A String expression that indicates the value to be converted from
Type as PropertyTypeEnum	A PropertyTypeEnum expression that specifies the type to be converted to
Variant	A VARIANT expression that specifies the value being converted

The fromICalendar property converts the ICalendar value to a VARIANT expression. The toICalendar property is the reverse function, so it converts a VARIANT expression back to iCalendar format. The valuesFromICalendar property returns values for known parameters of the giving expression (available for exPropertyTypeDuration, exPropertyTypePeriod and exPropertyTypeRecur types)

Based on the Type parameter of the fromICalendar property, the type of the VARIANT being returned is:

- exPropertyTypeBinary, returns a safe array of bytes (VT_ARRAY | VT_UI1)
- exPropertyTypeBoolean, returns a boolean value (VT_BOOL)
- exPropertyTypeCalAddress, returns a string value (VT_BSTR)
- exPropertyTypeDate, returns a DATE value (VT_DATE)
- exPropertyTypeDateTime, returns a DATE value (VT_DATE)
- exPropertyTypeDuration, returns a float value (VT_R4) that indicates the duration in days for integer part, while the fractional part specifies hours, minutes and seconds. The valuesFromICalendar property returns values for known parameters of the giving expression. The exPropertyTypeDuration expression supports the following parameters:
  - "-", specifies whether the duration is negative or positive
  - "W", indicates the number of weeks
  - "D", indicates the number of days
  - "H", indicates the number of hours
  - "M", indicates the number of minutes
  - "S", indicates the number of seconds
  - "Duration", indicates the duration in days for integer part, while the fractional part specifies hours, minutes and seconds.

For instance, the valuesFromICalendar("P12DT4H",exPropertyTypeDuration,"D") returns the number of days in the duration expression (in this case 12), while the valuesFromICalendar("P12DT4H",exPropertyTypeDuration,"H") returns the number of hours in the duration expression (in this case 4) .

- exPropertyTypeFloat, returns a double value (VT_R8)
- exPropertyTypeInteger, returns an integer value (VT_I4)
- exPropertyTypePeriod, returns a string value (VT_BSTR). The valuesFromICalendar property returns values for known parameters of the giving expression. The exPropertyTypePeriod expression supports the following parameters:
    - "Start", indicates the date to start the period
    - "End", indicates the date to end the period
    - "-", specifies whether the duration of the period is negative or positive
    - "W", indicates the number of weeks within period
    - "D", indicates the number of days within period
    - "H", indicates the number of hours within period
    - "M", indicates the number of minutes within period
    - "S", indicates the number of seconds within period
    - "Duration", indicates the duration of the period in days for integer part, while the fractional part specifies hours, minutes and seconds.

  For instance, the valuesFromICalendar("20010101T000000/P1D",exPropertyTypePeriod,"Start") returns the date to start the period (in this case #1/1/2001#), while the valuesFromICalendar("20010101T000000/P1D",exPropertyTypePeriod,"D") returns the number of days for the period expression (in this case 1).

- exPropertyTypeRecur, returns a string value (VT_BSTR). The valuesFromICalendar property returns values for known parameters of the giving expression. The exPropertyTypeRecur expression supports the following parameters:
    - "FREQ", identifies the type of recurrence rule, and could by any of the following: "SECONDLY", "MINUTELY", "HOURLY", "DAILY", "WEEKLY", "MONTHLY", "YEARLY"
    - "UNTIL", defines a DATE value that bounds the recurrence rule in an inclusive manner.
    - "COUNT", defines the number of occurrences at which to range-bound the recurrence.
    - "INTERVAL", contains a positive integer representing at which intervals the recurrence rule repeats.
    - "BYSECOND", specifies a COMMA-separated list of seconds within a minute.
    - "BYMINUTE", specifies a COMMA-separated list of minutes within an hour
    - "BYHOUR", specifies a COMMA-separated list of hours of the day
    - "BYDAY", specifies a COMMA-separated list of days of the week; SU indicates Sunday; MO indicates Monday; TU indicates Tuesday; WE indicates Wednesday;

TH indicates Thursday; FR indicates Friday; and SA indicates Saturday. Each BYDAY value can also be preceded by a positive (+n) or negative (-n) integer. If present, this indicates the nth occurrence of a specific day within the MONTHLY or YEARLY "RRULE".
    - "BYMONTHDAY", specifies a COMMA-separated list of days of the month. Valid values are 1 to 31 or -31 to -1.
    - "BYYEARDAY", specifies a COMMA-separated list of days of the year. Valid values are 1 to 366 or -366 to -1.
    - "BYWEEKNO", specifies a COMMA-separated list of ordinals specifying weeks of the year. Valid values are 1 to 53 or -53 to -1.
    - "BYMONTH", specifies a COMMA-separated list of months of the year. Valid values are 1 to 12.
    - "BYSETPOS", specifies a COMMA-separated list of values that corresponds to the nth occurrence within the set of recurrence instances specified by the rule.
    - "WKST", specifies the day on which the workweek starts. Valid values are MO, TU, WE, TH, FR, SA, and SU. The default value is MO.

  For instance, the valuesFromICalendar("FREQ=WEEKLY;INTERVAL=2;COUNT=4;BYDAY=SA,SU",exPr returns all known properties of the recurrence rule like "BYDAY=SA,SU;COUNT=4;FREQ=WEEKLY;INTERVAL=2", while the valuesFromICalendar("FREQ=WEEKLY;INTERVAL=2;COUNT=4;BYDAY=SA,SU",exPr returns the value of the FREQ parameter (in this case WEEKLY).

- exPropertyTypeText, returns a string value (VT_BSTR)
- exPropertyTypeTime, returns a DATE value (VT_DATE)
- exPropertyTypeURI, returns a string value (VT_BSTR)
- exPropertyTypeUTCOffset, returns a string value (VT_BSTR)

method ICalendar.Load (Content as String)

Loads and parses the iCalendar format from giving text.

Type	Description
Content as String	A String expression that specifies the iCalendar format to be loaded.
Return	**Description**
Variant	A VARIANT expression. Currently this is not used.

The Load method loads iCalendar format from giving string. The LoadFile / LoadFileFromUnicode method loads iCalendar format from a file.

The order of the events for Load, LoadFile, LoadFileFromUnicode methods is:

- StartLoad event, notifies that the Load, LoadFile, LoadFileFromUnicode methods begins
- AddComponent(NewComponent) event, notifies that a new Component object is added to the Components collection.
- AddProperty(NewPropery) event, notifies that a new Property object is added to the Properties collection. The Component property of the Property object specifies owner/parent component of the newly added property.
- AddParameter(NewParameter) event, notifies that a new Parameter object is added to the Parameters collection. The Property property of the Parameter object specifies owner/parent property of the newly added parameter.
- EndLoad event, notifies that the Load, LoadFile, LoadFileFromUnicode methods ends.

The Save method saves the control's content to a string, as iCalendar format. The SaveFile / SaveFileAsUnicode method saves the control's content to a file as iCalendar format.

method ICalendar.LoadFile (FileName as String)

Loads a file.

Type	Description
FileName as String	A String expression that indicates the ICS file to be loaded.

The LoadFile / LoadFileFromUnicode method loads iCalendar format from a file. The Load method loads iCalendar format from giving string.

The order of the events for Load, LoadFile, LoadFileFromUnicode methods is:

- StartLoad event, notifies that the Load, LoadFile, LoadFileFromUnicode methods begins
- AddComponent(NewComponent) event, notifies that a new Component object is added to the Components collection.
- AddProperty(NewPropery) event, notifies that a new Property object is added to the Properties collection. The Component property of the Property object specifies owner/parent component of the newly added property.
- AddParameter(NewParameter) event, notifies that a new Parameter object is added to the Parameters collection. The Property property of the Parameter object specifies owner/parent property of the newly added parameter.
- EndLoad event, notifies that the Load, LoadFile, LoadFileFromUnicode methods ends.

The Save method saves the control's content to a string, as iCalendar format. The SaveFile / SaveFileAsUnicode method saves the control's content to a file as iCalendar format.

method ICalendar.LoadFileFromUnicode (FileName as String)

Loads from an UNICODE file.

Type	Description
FileName as String	A String expression that specifies the file to be loaded.

The LoadFile / LoadFileFromUnicode method loads iCalendar format from a file. The Load method loads iCalendar format from giving string.

The order of the events for Load, LoadFile, LoadFileFromUnicode methods is:

- StartLoad event, notifies that the Load, LoadFile, LoadFileFromUnicode methods begins
- AddComponent(NewComponent) event, notifies that a new Component object is added to the Components collection.
- AddProperty(NewPropery) event, notifies that a new Property object is added to the Properties collection. The Component property of the Property object specifies owner/parent component of the newly added property.
- AddParameter(NewParameter) event, notifies that a new Parameter object is added to the Parameters collection. The Property property of the Parameter object specifies owner/parent property of the newly added parameter.
- EndLoad event, notifies that the Load, LoadFile, LoadFileFromUnicode methods ends.

The Save method saves the control's content to a string, as iCalendar format. The SaveFile / SaveFileAsUnicode method saves the control's content to a file as iCalendar format.

property ICalendar.RecurAll (Recur as String, [Count as Variant]) as Variant

Returns all recurrences of the specified rule, as a safe array of dates.

Type	Description
Recur as String	A String expression that specifies the recurrence rule according to RFC 5545. The Recur parameter must include DTSTART and FREQ rule parts, else the recurrence rule is not valid.
Count as Variant	A Long expression that indicates the number of maximum occurrences that RecurAll method should return. If Count parameter is missing, maximum 256 occurrences are returned.
Variant	A safe array of dates that specifies the occurrences of the recurrence rule. You can use the **for each** statement to enumerates the dates in the safe array.

The RecurAll property returns all occurrences of the giving recurrence rule, as a safe array. The RecurAllAsString property returns all occurrences of the giving recurrence rule, as a string. The RecurRange property returns the occurrences of the giving recurrence rule between start and end dates (interval, range of dates), as a safe array. The RecurRangeAsString property returns the occurrences of the giving recurrence rule between start and end dates (interval, range of dates), as a string. The RecurCheck property specifies whether the date fits the giving recurrence rule. Use the RecurPartValue(exRecurSyntaxError) or RecurPartValue(exRecurSyntaxErrorInfo) to specify whether the Recur parameter is valid or invalid.

The BNF syntax for Recur parameter is:

```
recur = recur-rule-part *( ";" recur-rule-part )
  ;
  ; The rule parts are not ordered in any
  ; particular sequence.
  ;
  ; The FREQ rule part is REQUIRED,
  ; but MUST NOT occur more than once.
  ;
  ; The UNTIL or COUNT rule parts are OPTIONAL,
  ; but they MUST NOT occur in the same 'recur'.
  ;
```

```
    ; The other rule parts are OPTIONAL,
    ; but MUST NOT occur more than once.

recur-rule-part = ( "FREQ" "=" freq )
                / ( "UNTIL" "=" enddate )
                / ( "COUNT" "=" 1*DIGIT )
                / ( "INTERVAL" "=" 1*DIGIT )
                / ( "BYSECOND" "=" byseclist )
                / ( "BYMINUTE" "=" byminlist )
                / ( "BYHOUR" "=" byhrlist )
                / ( "BYDAY" "=" bywdaylist )
                / ( "BYMONTHDAY" "=" bymodaylist )
                / ( "BYYEARDAY" "=" byyrdaylist )
                / ( "BYWEEKNO" "=" bywknolist )
                / ( "BYMONTH" "=" bymolist )
                / ( "BYSETPOS" "=" bysplist )
                / ( "WKST" "=" weekday )

freq        = "SECONDLY" / "MINUTELY" / "HOURLY" / "DAILY"
            / "WEEKLY" / "MONTHLY" / "YEARLY"
enddate     = date / date-time
byseclist   = ( seconds *("," seconds) )
seconds     = 1*2DIGIT       ;0 to 60
byminlist   = ( minutes *("," minutes) )
minutes     = 1*2DIGIT       ;0 to 59
byhrlist    = ( hour *("," hour) )
hour        = 1*2DIGIT       ;0 to 23
bywdaylist  = ( weekdaynum *("," weekdaynum) )
weekdaynum  = [[plus / minus] ordwk] weekday
plus        = "+"
minus       = "-"
ordwk       = 1*2DIGIT       ;1 to 53
weekday     = "SU" / "MO" / "TU" / "WE" / "TH" / "FR" / "SA" ;Corresponding to SUNDAY,
MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and SATURDAY days of the week.
bymodaylist = ( monthdaynum *("," monthdaynum) )
monthdaynum = [plus / minus] ordmoday
ordmoday    = 1*2DIGIT       ;1 to 31
```

```
byyrdaylist = ( yeardaynum *("," yeardaynum) )
yeardaynum  = [plus / minus] ordyrday
ordyrday    = 1*3DIGIT     ;1 to 366
bywknolist  = ( weeknum *("," weeknum) )
weeknum     = [plus / minus] ordwk
bymolist    = ( monthnum *("," monthnum) )
monthnum    = 1*2DIGIT      ;1 to 12
bysplist    = ( setposday *("," setposday) )
setposday   = yeardaynum
```

This value type is a structured value consisting of a list of one or more recurrence grammar parts. Each rule part is defined by a NAME=VALUE pair. The rule parts are separated from each other by the SEMICOLON character. The rule parts are not ordered in any particular sequence. Individual rule parts MUST only be specified once. Compliant applications MUST accept rule parts ordered in any sequence, but to ensure backward compatibility with applications that pre-date this revision of iCalendar the FREQ rule part MUST be the first rule part specified in a RECUR value.

The FREQ rule part identifies the type of recurrence rule. This rule part MUST be specified in the recurrence rule. Valid values include SECONDLY, to specify repeating events based on an interval of a second or more; MINUTELY, to specify repeating events based on an interval of a minute or more; HOURLY, to specify repeating events based on an interval of an hour or more; DAILY, to specify repeating events based on an interval of a day or more; WEEKLY, to specify repeating events based on an interval of a week or more; MONTHLY, to specify repeating events based on an interval of a month or more; and YEARLY, to specify repeating events based on an interval of a year or more.

The INTERVAL rule part contains a positive integer representing at which intervals the recurrence rule repeats. The default value is "1", meaning every second for a SECONDLY rule, every minute for a MINUTELY rule, every hour for an HOURLY rule, every day for a DAILY rule, every week for a WEEKLY rule, every month for a MONTHLY rule, and every year for a YEARLY rule. For example, within a DAILY rule, a value of "8" means every eight days.

The UNTIL rule part defines a DATE or DATE-TIME value that bounds the recurrence rule in an inclusive manner. If the value specified by UNTIL is synchronized with the specified recurrence, this DATE or DATE-TIME becomes the last instance of the recurrence. The value of the UNTIL rule part MUST have the same value type as the "DTSTART" property. Furthermore, if the "DTSTART" property is specified as a date with local time, then the UNTIL rule part MUST also be specified as a date with local time. If the "DTSTART" property is specified as a date with UTC time or a date with local time and time zone reference, then the UNTIL rule part MUST be specified as a date with UTC time. In the case of the "STANDARD" and "DAYLIGHT" sub-components the UNTIL rule part MUST always be specified as a date with UTC time. If specified as a DATE-TIME value, then it MUST be specified in a UTC time format. If not present, and the COUNT rule part is also not present, the "RRULE" is considered to repeat forever.

The COUNT rule part defines the number of occurrences at which to range-bound the recurrence. The "DTSTART" property value always counts as the first occurrence.

The BYSECOND rule part specifies a COMMA-separated list of seconds within a minute. Valid values are 0 to 60. The BYMINUTE rule part specifies a COMMA-separated list of minutes within an hour. Valid values are 0 to 59. The BYHOUR rule part specifies a COMMA- separated list of hours of the day. Valid values are 0 to 23. The BYSECOND, BYMINUTE and BYHOUR rule parts MUST NOT be specified when the associated "DTSTART" property has a DATE value type. These rule parts MUST be ignored in RECUR value that violate the above requirement (e.g., generated by applications that pre-date this revision of iCalendar).

The BYDAY rule part specifies a COMMA-separated list of days of the week; SU indicates Sunday; MO indicates Monday; TU indicates Tuesday; WE indicates Wednesday; TH indicates Thursday; FR indicates Friday; and SA indicates Saturday. Each BYDAY value can also be preceded by a positive (+n) or negative (-n) integer. If present, this indicates the nth occurrence of a specific day within the MONTHLY or YEARLY "RRULE".
For example, within a MONTHLY rule, +1MO (or simply 1MO) represents the first Monday within the month, whereas -1MO represents the last Monday of the month. The numeric value in a BYDAY rule part with the FREQ rule part set to YEARLY corresponds to an offset within the month when the BYMONTH rule part is present, and corresponds to an offset within the year when the BYWEEKNO or BYMONTH rule parts are present. If an integer modifier is not present, it means all days of this type within the specified frequency. For example, within a MONTHLY rule, MO represents all Mondays within the month. The BYDAY rule part MUST NOT be specified with a numeric value when the FREQ rule part is not set to MONTHLY or YEARLY. Furthermore, the BYDAY rule part MUST NOT be specified with a numeric value with the FREQ rule part set to YEARLY when the BYWEEKNO rule part is specified.

The BYMONTHDAY rule part specifies a COMMA-separated list of days of the month. Valid values are 1 to 31 or -31 to -1. For example, -10 represents the tenth to the last day of the month. The BYMONTHDAY rule part MUST NOT be specified when the FREQ rule part is set to WEEKLY.

The BYYEARDAY rule part specifies a COMMA-separated list of days of the year. Valid values are 1 to 366 or -366 to -1. For example, -1 represents the last day of the year (December 31st) and -306 represents the 306th to the last day of the year (March 1st). The BYYEARDAY rule part MUST NOT be specified when the FREQ rule part is set to DAILY, WEEKLY, or MONTHLY.

The BYWEEKNO rule part specifies a COMMA-separated list of ordinals specifying weeks of the year. Valid values are 1 to 53 or -53 to -1. This corresponds to weeks according to week numbering as defined in [ISO.8601.2004]. A week is defined as a seven day period, starting on the day of the week defined to be the week start (see WKST). Week number one of the calendar year is the first week that contains at least four (4) days in that calendar year. This rule part MUST NOT be used when the FREQ rule part is set to anything other than YEARLY. For example, 3 represents the third week of the year.

Note: Assuming a Monday week start, week 53 can only occur when Thursday is January 1 or if it is a leap year and Wednesday is January 1.

The BYMONTH rule part specifies a COMMA-separated list of months of the year. Valid values are 1 to 12.

The WKST rule part specifies the day on which the workweek starts. Valid values are MO, TU, WE, TH, FR, SA, and SU. This is significant when a WEEKLY "RRULE" has an interval greater than 1, and a BYDAY rule part is specified. This is also significant when in a YEARLY "RRULE" when a BYWEEKNO rule part is specified. The default value is MO.

The BYSETPOS rule part specifies a COMMA-separated list of values that corresponds to the nth occurrence within the set of recurrence instances specified by the rule. BYSETPOS operates on a set of recurrence instances in one interval of the recurrence rule. For example, in a WEEKLY rule, the interval would be one week A set of recurrence instances starts at the beginning of the interval defined by the FREQ rule part. Valid values are 1 to 366 or -366 to -1. It MUST only be used in conjunction with another BYxxx rule part. For example "the last work day of the month" could be represented as:

```
FREQ=MONTHLY;BYDAY=MO,TU,WE,TH,FR;BYSETPOS=-1
```

Each BYSETPOS value can include a positive (+n) or negative (-n) integer. If present, this indicates the nth occurrence of the specific occurrence within the set of occurrences specified by the rule.

Recurrence rules may generate recurrence instances with an invalid date (e.g., February 30) or nonexistent local time (e.g., 1:30 AM on a day where the local time is moved forward by an hour at 1:00 AM). Such recurrence instances MUST be ignored and MUST NOT be counted as part of the recurrence set. Information, not contained in the rule, necessary to determine the various recurrence instance start time and dates are derived from the Start Time ("DTSTART") component attribute. For example, "FREQ=YEARLY;BYMONTH=1" doesn't specify a specific day within the month or a time. This information would be the same as what is specified for "DTSTART".

The following VB sample enumerates the occurrences of the recurrence rule:

```
Set ICalendar1 = CreateObject("Exontrol.ICalendar.1")
With ICalendar1
   For Each v In
.RecurAll("DTSTART=19970805;FREQ=MONTHLY;BYDAY=MO,TU,WE,TH,FR;BYSETPOS=-1",
12)
      Debug.Print v
   Next
End With
```

property ICalendar.RecurAllAsString (Recur as String, [Count as Variant]) as Variant

Returns all recurrences of the specified rule, as string (dates are separated by new line) . The RecurAllAsString method returns the error, in case the rule is incorrectly.

Type	Description
Recur as String	A String expression that specifies the recurrence rule according to RFC 5545. The Recur parameter must include DTSTART and FREQ rule parts, else the recurrence rule is not valid.
Count as Variant	A Long expression that indicates the number of maximum occurrences that RecurAll method should return. If Count parameter is missing, maximum 256 occurrences are returned.
Variant	A String expression of dates that specifies the occurrences of the recurrence rule (separated by crlf "\r\n" sequence).

The RecurAllAsString property returns all occurrences of the giving recurrence rule, as a string. The RecurAll property returns all occurrences of the giving recurrence rule, as a safe array. The RecurRange property returns the occurrences of the giving recurrence rule between start and end dates (interval, range of dates), as a safe array. The RecurRangeAsString property returns the occurrences of the giving recurrence rule between start and end dates (interval, range of dates), as a string. The RecurCheck property specifies whether the date fits the giving recurrence rule. Use the RecurPartValue(exRecurSyntaxError) or RecurPartValue(exRecurSyntaxErrorInfo) to specify whether the Recur parameter is valid or invalid.

The BNF syntax for Recur parameter is:

```
recur = recur-rule-part *( ";" recur-rule-part )
    ;
    ; The rule parts are not ordered in any
    ; particular sequence.
    ;
    ; The FREQ rule part is REQUIRED,
    ; but MUST NOT occur more than once.
    ;
    ; The UNTIL or COUNT rule parts are OPTIONAL,
    ; but they MUST NOT occur in the same 'recur'.
    ;
    ; The other rule parts are OPTIONAL,
    ; but MUST NOT occur more than once.

recur-rule-part = ( "FREQ" "=" freq )
               / ( "UNTIL" "=" enddate )
               / ( "COUNT" "=" 1*DIGIT )
               / ( "INTERVAL" "=" 1*DIGIT )
               / ( "BYSECOND" "=" byseclist )
               / ( "BYMINUTE" "=" byminlist )
               / ( "BYHOUR" "=" byhrlist )
               / ( "BYDAY" "=" bywdaylist )
               / ( "BYMONTHDAY" "=" bymodaylist )
               / ( "BYYEARDAY" "=" byyrdaylist )
               / ( "BYWEEKNO" "=" bywknolist )
               / ( "BYMONTH" "=" bymolist )
               / ( "BYSETPOS" "=" bysplist )
               / ( "WKST" "=" weekday )

freq        = "SECONDLY" / "MINUTELY" / "HOURLY" / "DAILY"
           / "WEEKLY" / "MONTHLY" / "YEARLY"
enddate     = date / date-time
byseclist   = ( seconds *("," seconds) )
seconds     = 1*2DIGIT       ;0 to 60
byminlist   = ( minutes *("," minutes) )
minutes     = 1*2DIGIT       ;0 to 59
byhrlist    = ( hour *("," hour) )
hour        = 1*2DIGIT       ;0 to 23
bywdaylist  = ( weekdaynum *("," weekdaynum) )
weekdaynum  = [[plus / minus] ordwk] weekday
plus        = "+"
minus       = "-"
ordwk       = 1*2DIGIT       ;1 to 53
weekday     = "SU" / "MO" / "TU" / "WE" / "TH" / "FR" / "SA" ;Corresponding to SUNDAY,
MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and SATURDAY days of the week.
bymodaylist = ( monthdaynum *("," monthdaynum) )
monthdaynum = [plus / minus] ordmoday
```

```
ordmoday    = 1*2DIGIT      ;1 to 31
byyrdaylist = ( yeardaynum *("," yeardaynum) )
yeardaynum  = [plus / minus] ordyrday
ordyrday    = 1*3DIGIT      ;1 to 366
bywknolist  = ( weeknum *("," weeknum) )
weeknum     = [plus / minus] ordwk
bymolist    = ( monthnum *("," monthnum) )
monthnum    = 1*2DIGIT      ;1 to 12
bysplist    = ( setposday *("," setposday) )
setposday   = yeardaynum
```

This value type is a structured value consisting of a list of one or more recurrence grammar parts. Each rule part is defined by a NAME=VALUE pair. The rule parts are separated from each other by the SEMICOLON character. The rule parts are not ordered in any particular sequence. Individual rule parts MUST only be specified once. Compliant applications MUST accept rule parts ordered in any sequence, but to ensure backward compatibility with applications that pre-date this revision of iCalendar the FREQ rule part MUST be the first rule part specified in a RECUR value.

The FREQ rule part identifies the type of recurrence rule. This rule part MUST be specified in the recurrence rule. Valid values include SECONDLY, to specify repeating events based on an interval of a second or more; MINUTELY, to specify repeating events based on an interval of a minute or more; HOURLY, to specify repeating events based on an interval of an hour or more; DAILY, to specify repeating events based on an interval of a day or more; WEEKLY, to specify repeating events based on an interval of a week or more; MONTHLY, to specify repeating events based on an interval of a month or more; and YEARLY, to specify repeating events based on an interval of a year or more.

The INTERVAL rule part contains a positive integer representing at which intervals the recurrence rule repeats. The default value is "1", meaning every second for a SECONDLY rule, every minute for a MINUTELY rule, every hour for an HOURLY rule, every day for a DAILY rule, every week for a WEEKLY rule, every month for a MONTHLY rule, and every year for a YEARLY rule. For example, within a DAILY rule, a value of "8" means every eight days.

The UNTIL rule part defines a DATE or DATE-TIME value that bounds the recurrence rule in an inclusive manner. If the value specified by UNTIL is synchronized with the specified recurrence, this DATE or DATE-TIME becomes the last instance of the recurrence. The value of the UNTIL rule part MUST have the same value type as the "DTSTART" property. Furthermore, if the "DTSTART" property is specified as a date with local time, then the UNTIL rule part MUST also be specified as a date with local time. If the "DTSTART" property is specified as a date with UTC time or a date with local time and time zone reference, then the UNTIL rule part MUST be specified as a date with UTC time. In the case of the "STANDARD" and "DAYLIGHT" sub-components the UNTIL rule part MUST always be specified as a date with UTC time. If specified as a DATE-TIME value, then it MUST be specified in a UTC time format. If not present, and the COUNT rule part is also not present, the "RRULE" is considered to repeat forever.

The COUNT rule part defines the number of occurrences at which to range-bound the recurrence. The "DTSTART" property value always counts as the first occurrence.

The BYSECOND rule part specifies a COMMA-separated list of seconds within a minute. Valid values are 0 to 60. The BYMINUTE rule part specifies a COMMA-separated list of minutes within an hour. Valid values are 0 to 59. The BYHOUR rule part specifies a COMMA- separated list of hours of the day. Valid values are 0 to 23. The BYSECOND, BYMINUTE and BYHOUR rule parts MUST NOT be specified when the associated "DTSTART" property has a DATE value type. These rule parts MUST be ignored in RECUR value that violate the above requirement (e.g., generated by applications that pre-date this revision of iCalendar).

The BYDAY rule part specifies a COMMA-separated list of days of the week; SU indicates Sunday; MO indicates Monday; TU indicates Tuesday; WE indicates Wednesday; TH indicates Thursday; FR indicates Friday; and SA indicates Saturday. Each BYDAY value can also be preceded by a positive (+n) or negative (-n) integer. If present, this indicates the nth occurrence of a specific day within the MONTHLY or YEARLY "RRULE".
For example, within a MONTHLY rule, +1MO (or simply 1MO) represents the first Monday within the month, whereas -1MO represents the last Monday of the month. The numeric value in a BYDAY rule part with the FREQ rule part set to YEARLY corresponds to an offset within the month when the BYMONTH rule part is present, and corresponds to an offset within the year when the BYWEEKNO or BYMONTH rule parts are present. If an integer modifier is not present, it means all days of this type within the specified frequency. For example, within a MONTHLY rule, MO represents all Mondays within the month. The BYDAY rule part MUST NOT be specified with a numeric value when the FREQ rule part is not set to MONTHLY or YEARLY. Furthermore, the BYDAY rule part MUST NOT be specified with a numeric value with the FREQ rule part set to YEARLY when the BYWEEKNO rule part is specified.

The BYMONTHDAY rule part specifies a COMMA-separated list of days of the month. Valid values are 1 to 31 or -31 to -1. For example, -10 represents the tenth to the last day of the month. The BYMONTHDAY rule part MUST NOT be specified when the FREQ rule part is set to WEEKLY.

The BYYEARDAY rule part specifies a COMMA-separated list of days of the year. Valid values are 1 to 366 or -366 to -1. For example, -1 represents the last day of the year (December 31st) and -306 represents the 306th to the last day of the year (March 1st). The BYYEARDAY rule part MUST NOT be specified when the FREQ rule part is set to

DAILY, WEEKLY, or MONTHLY.

The BYWEEKNO rule part specifies a COMMA-separated list of ordinals specifying weeks of the year. Valid values are 1 to 53 or -53 to -1. This corresponds to weeks according to week numbering as defined in [ISO.8601.2004]. A week is defined as a seven day period, starting on the day of the week defined to be the week start (see WKST). Week number one of the calendar year is the first week that contains at least four (4) days in that calendar year. This rule part MUST NOT be used when the FREQ rule part is set to anything other than YEARLY. For example, 3 represents the third week of the year.

> Note: Assuming a Monday week start, week 53 can only occur when Thursday is January 1 or if it is a leap year and Wednesday is January 1.

The BYMONTH rule part specifies a COMMA-separated list of months of the year. Valid values are 1 to 12.

The WKST rule part specifies the day on which the workweek starts. Valid values are MO, TU, WE, TH, FR, SA, and SU. This is significant when a WEEKLY "RRULE" has an interval greater than 1, and a BYDAY rule part is specified. This is also significant when in a YEARLY "RRULE" when a BYWEEKNO rule part is specified. The default value is MO.

The BYSETPOS rule part specifies a COMMA-separated list of values that corresponds to the nth occurrence within the set of recurrence instances specified by the rule. BYSETPOS operates on a set of recurrence instances in one interval of the recurrence rule. For example, in a WEEKLY rule, the interval would be one week A set of recurrence instances starts at the beginning of the interval defined by the FREQ rule part. Valid values are 1 to 366 or -366 to -1. It MUST only be used in conjunction with another BYxxx rule part. For example "the last work day of the month" could be represented as:

```
FREQ=MONTHLY;BYDAY=MO,TU,WE,TH,FR;BYSETPOS=-1
```

Each BYSETPOS value can include a positive (+n) or negative (-n) integer. If present, this indicates the nth occurrence of the specific occurrence within the set of occurrences specified by the rule.

Recurrence rules may generate recurrence instances with an invalid date (e.g., February 30) or nonexistent local time (e.g., 1:30 AM on a day where the local time is moved forward by an hour at 1:00 AM). Such recurrence instances MUST be ignored and MUST NOT be counted as part of the recurrence set. Information, not contained in the rule, necessary to determine the various recurrence instance start time and dates are derived from the Start Time ("DTSTART") component attribute. For example, "FREQ=YEARLY;BYMONTH=1" doesn't specify a specific day within the month or a time. This information would be the same as what is specified for "DTSTART".

The following samples shows how you can display the last work day of the each month:

8/29/1997
9/30/1997
10/31/1997
11/28/1997
12/31/1997
1/30/1998
2/27/1998
3/31/1998
4/30/1998
5/29/1998
6/30/1998
...

VBA (MS Access, Excell...)

```
Set ICalendar1 = CreateObject("Exontrol.ICalendar.1")
With ICalendar1
   Debug.Print(
.RecurAllAsString("DTSTART=19970805;FREQ=MONTHLY;BYDAY=MO,TU,WE,TH,FR;BY
)
End With
```

VB6

```
Set ICalendar1 = CreateObject("Exontrol.ICalendar.1")
With ICalendar1
   Debug.Print(
.RecurAllAsString("DTSTART=19970805;FREQ=MONTHLY;BYDAY=MO,TU,WE,TH,FR;BY
)
End With
```

VB.NET

```
' Add 'exontrol.exicalendar.dll(ExICalendar.dll)' reference to your project.
Exicalendar1 = New exontrol.EXICALENDARLib.exicalendar()
With Exicalendar1
   Debug.Print(
.get_RecurAllAsString("DTSTART=19970805;FREQ=MONTHLY;BYDAY=MO,TU,WE,TH,I
)
End With
```

VB.NET for /COM

```
AxICalendar1 = CreateObject("Exontrol.ICalendar.1")
With AxICalendar1
   Debug.Print(
.get_RecurAllAsString("DTSTART=19970805;FREQ=MONTHLY;BYDAY=MO,TU,WE,TH,I
 )
End With
```

C++

```
/*
    Includes the definition for CreateObject function like follows:

    #include <comdef.h>
    IUnknownPtr CreateObject( BSTR Object )
    {
        IUnknownPtr spResult;
        spResult.CreateInstance( Object );
        return spResult;
    };

*/
/*
    Copy and paste the following directives to your header file as
    it defines the namespace 'EXICALENDARLib' for the library: 'ICalendar 1.0 Type
Library'

    #import <ExICalendar.dll>
    using namespace EXICALENDARLib;
*/
EXICALENDARLib::IICalendarPtr spICalendar1 =
::CreateObject(L"Exontrol.ICalendar.1");
OutputDebugStringW( _bstr_t(spICalendar1-
>GetRecurAllAsString(L"DTSTART=19970805;FREQ=MONTHLY;BYDAY=MO,TU,WE,TH
 );
```

C++ Builder

```
Exicalendarlib_tlb::IICalendarPtr ICalendar1 =
Variant::CreateObject(L"Exontrol.ICalendar.1");
OutputDebugString( PChar(ICalendar1-
>RecurAllAsString[L"DTSTART=19970805;FREQ=MONTHLY;BYDAY=MO,TU,WE,TH,FR
 );
```

C#

```
// Add 'exontrol.exicalendar.dll(ExICalendar.dll)' reference to your project.
exontrol.EXICALENDARLib.exicalendar exicalendar1 = new
exontrol.EXICALENDARLib.exicalendar();
System.Diagnostics.Debug.Print(
exicalendar1.get_RecurAllAsString("DTSTART=19970805;FREQ=MONTHLY;BYDAY=M
 );
```

JScript/JavaScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:D6C87100-38E2-4ABB-8AC2-4C0097AEE2D6"
id="ICalendar1"></OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    alert(
ICalendar1.RecurAllAsString("DTSTART=19970805;FREQ=MONTHLY;BYDAY=MO,TU,W
 );
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload="Init()">
```

```
<OBJECT CLASSID="clsid:D6C87100-38E2-4ABB-8AC2-4C0097AEE2D6"
id="ICalendar1"></OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
   With ICalendar1
      alert(
.RecurAllAsString("DTSTART=19970805;FREQ=MONTHLY;BYDAY=MO,TU,WE,TH,FR;BY
 )
   End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
EXICALENDARLib.ICalendar axICalendar1 = new EXICALENDARLib.ICalendar();
System.Diagnostics.Debug.Print(
axICalendar1.get_RecurAllAsString("DTSTART=19970805;FREQ=MONTHLY;BYDAY=M
 );
```

X++ (Dynamics Ax 2009)

```
public void init()
{
   COM com_exicalendar1;
   anytype exicalendar1;
   ;

   super();

   // Add 'exicalendar.dll(ExICalendar.dll)' reference to your project.
   exicalendar1 = COM::createFromObject(new EXICALENDARLib.exicalendar());
com_exicalendar1 = exicalendar1;
   print(
com_exicalendar1.RecurAllAsString("DTSTART=19970805;FREQ=MONTHLY;BYDAY=M
 );
}
```

Delphi 8 (.NET only)

```
AxICalendar1 :=
(ComObj.CreateComObject(ComObj.ProgIDToClassID('Exontrol.ICalendar.1')) as
EXICALENDARLib.ICalendar);
with AxICalendar1 do
begin
   OutputDebugString(
get_RecurAllAsString('DTSTART=19970805;FREQ=MONTHLY;BYDAY=MO,TU,WE,TH,F
 );
end
```

Delphi (standard)

```
ICalendar1 :=
(IUnknown(ComObj.CreateComObject(ComObj.ProgIDToClassID('Exontrol.ICalendar.1'))
 as EXICALENDARLib_TLB.ICalendar);
with ICalendar1 do
begin
   OutputDebugString(
RecurAllAsString['DTSTART=19970805;FREQ=MONTHLY;BYDAY=MO,TU,WE,TH,FR;BY
 );
end
```

VFP

```
thisform.ICalendar1 = CreateObject("Exontrol.ICalendar.1")
with thisform.ICalendar1
   DEBUGOUT(
.RecurAllAsString("DTSTART=19970805;FREQ=MONTHLY;BYDAY=MO,TU,WE,TH,FR;BY
 )
endwith
```

dBASE Plus

```
local oICalendar
```

```
oICalendar = new OleAutoClient("Exontrol.ICalendar.1")

?
Str(oICalendar.RecurAllAsString("DTSTART=19970805;FREQ=MONTHLY;BYDAY=MO,T
```

XBasic (Alpha Five)

```
Dim oICalendar as P

oICalendar = OLE.Create("Exontrol.ICalendar.1")

?
oICalendar.RecurAllAsString("DTSTART=19970805;FREQ=MONTHLY;BYDAY=MO,TU,W
```

Visual Objects

```
oDCOCX_Exontrol1 := IICalendar{"Exontrol.ICalendar.1"}
OutputDebugString(String2Psz( AsString(oDCOCX_Exontrol1:
[RecurAllAsString,"DTSTART=19970805;FREQ=MONTHLY;BYDAY=MO,TU,WE,TH,FR;BYS
 ))
```

PowerBuilder

```
OleObject oICalendar

oICalendar = CREATE OLEObject
oICalendar.ConnectToNewObject("Exontrol.ICalendar.1")

MessageBox("Information",string(
String(oICalendar.RecurAllAsString("DTSTART=19970805;FREQ=MONTHLY;BYDAY=M
 ))
```

Visual DataFlex

```
Procedure OnCreate
    Forward Send OnCreate
    Variant oComICalendar1
    Get ComCreateObject "Exontrol.ICalendar.1" to oComICalendar1

    Showln
(ComRecurAllAsString(Self,"DTSTART=19970805;FREQ=MONTHLY;BYDAY=MO,TU,WE

End_Procedure
```

XBase++

```
#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oICalendar

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,,{100,100}, {640,480},, .F. )
    oForm:close  := {|| PostAppEvent( xbeP_Quit )}

    oICalendar := XbpActiveXControl():new( oForm:drawingArea )
    oICalendar:CLSID  := "Exontrol.ICalendar.1" /*{D6C87100-38E2-4ABB-8AC2-
4C0097AEE2D6}*/
    oICalendar:create(,, {10,60},{610,370} )

        DevOut(
Transform(oICalendar:RecurAllAsString("DTSTART=19970805;FREQ=MONTHLY;BYDAY
 )
```

```
    oForm:Show()
    DO WHILE nEvent != xbeP_Quit
        nEvent := AppEvent( @mp1, @mp2, @oXbp )
        oXbp:handleEvent( nEvent, mp1, mp2 )
    ENDDO
RETURN
```

property ICalendar.RecurAllMethod as RecurAllMethodEnum

Specifies the way the component gets the occurrences of the recurrence rule (RecurAll method).

Type	Description
RecurAllMethodEnum	A RecurAllMethodEnum expression that specifies the way RecurAll method works.

By default, the RecurAllMethod property is exRecurAllQuickValidate. The RecurAllMethod property is for internal use only.

property ICalendar.RecurAllTime as Long

Specifies the time in milliseconds that took to perform the last RecurAll call.

Type	Description
Long	A Long expression that specifies the time in milliseconds that took to perform the last RecurAll call.

The RecurAllTime property specifies the time to perform the RecurAll call. The RecurAllTime property was provided for debugging purposes.

property ICalendar.RecurCheck (Recur as String, Date as Date) as Long

Evaluates the date using recurrence rules, and returns 1 if the Date matches the recurrence rule, 0 if not, or a negative number if any error occurs.

Type	Description
Recur as String	A String expression that specifies the recurrence rule according to RFC 5545. The Recur parameter must include DTSTART and FREQ rule parts, else the recurrence rule is not valid.
Date as Date	A DATE expression to verify if it fits the recurrence rule.
Long	A Long expression that specifies if the Date fits the recurrence rule if 1, 0 if not, or negative number if any error occurs.

The RecurCheck property specifies whether the date fits the giving recurrence rule. Use the RecurPartValue(exRecurSyntaxError) or RecurPartValue(exRecurSyntaxErrorInfo) to specify whether the Recur parameter is valid or invalid. The RecurAll property returns all occurrences of the giving recurrence rule, as a safe array. The RecurAllAsString property returns all occurrences of the giving recurrence rule, as a string. The RecurRange property returns the occurrences of the giving recurrence rule between start and end dates (interval, range of dates), as a safe array. The RecurRangeAsString property returns the occurrences of the giving recurrence rule between start and end dates (interval, range of dates), as a string.

The RecurCheck property may return any of the following values:

- **1**, the Date matches the Recur rule.
- **0**, the Date does not match the Recur rule.
- -1 (exRecurrenceEmpty), Error: Empty or Invalid recurrence.
- -2 (exRecurrenceMissingFreq), Error: The FREQ rule is missing or invalid.
- -3 (exRecurrenceMissingStart), Error: The DTSTART property is missing or invalid..
- -4 (exRecurrenceUntilAndCount), Error: The UNTIL or COUNT rule parts are OPTIONAL, but they MUST NOT occur in the same 'recur'.
- -5 (exRecurrenceYearlyByWeekNo), Error: The BYWEEKNO part MUST NOT be used when the FREQ rule part is set to anything other than YEARLY.
- -6 (exRecurrenceNumberByDay), Error: The BYDAY rule part MUST NOT be specified with a numeric value when the FREQ rule part is not set to MONTHLY or YEARLY.
- -7 (exRecurrenceWeeklyByMonthDay), Error: The BYMONTHDAY rule part MUST NOT be specified when the FREQ rule part is set to WEEKLY.
- -8 (exRecurrenceFreqByYearDay), Error: The BYDAY rule part MUST NOT be specified with a numeric value with the FREQ rule part set to YEARLY when the BYWEEKNO rule part is specified.

- -9 (exRecurrenceNumberByDayAndWeekNo), Error: the BYDAY rule part MUST NOT be specified with a numeric value with the FREQ rule part set to YEARLY when the BYWEEKNO rule part is specified.
- -10 (exRecurrenceByWeekNoInvalid), Error: The BYWEEKNO rule part is empty or specifies no valid value (1 to 53).
- -11 (exRecurrenceByDayInvalid), Error: The BYDAY rule part is empty or specifies no valid value.
- -12 (exRecurrenceByMonthDayInvalid), Error: The BYMONTHDAY rule part is empty or specifies no valid value. Valid values are 1 to 31 or -31 to -1.
- -13 (exRecurrenceByYearDayInvalid), Error: The BYYEARDAY rule part is empty or specifies no valid value. Valid values are 1 to 366 or -366 to -1.
- -14 (exRecurrenceBySetPosInvalid), Error: The BYSETPOS rule part is empty or specifies no valid value. Valid values are 1 to 366 or -366 to -1.
- -15 (exRecurrenceByMonthInvalid), Error: The BYMONTH rule part is empty or specifies no valid value. Valid values are 1 to 12.
- -16 (exRecurrenceBySecondInvalid), Error: The BYSECOND rule part is empty or specifies no valid value. Valid values are 0 to 59.
- -17 (exRecurrenceByMinuteInvalid), Error: The BYMINUTE rule part is empty or specifies no valid value. Valid values are 0 to 59.
- -18 (exRecurrenceByHourInvalid), Error: The BYHOUR rule part is empty or specifies no valid value. Valid values are 0 to 23.

The BNF syntax for Recur parameter is:

```
recur = recur-rule-part *( ";" recur-rule-part )
  ;
  ; The rule parts are not ordered in any
  ; particular sequence.
  ;
  ; The FREQ rule part is REQUIRED,
  ; but MUST NOT occur more than once.
  ;
  ; The UNTIL or COUNT rule parts are OPTIONAL,
  ; but they MUST NOT occur in the same 'recur'.
  ;
  ; The other rule parts are OPTIONAL,
  ; but MUST NOT occur more than once.

recur-rule-part = ( "FREQ" "=" freq )
           / ( "UNTIL" "=" enddate )
```

```
            / ( "COUNT" "=" 1*DIGIT )
            / ( "INTERVAL" "=" 1*DIGIT )
            / ( "BYSECOND" "=" byseclist )
            / ( "BYMINUTE" "=" byminlist )
            / ( "BYHOUR" "=" byhrlist )
            / ( "BYDAY" "=" bywdaylist )
            / ( "BYMONTHDAY" "=" bymodaylist )
            / ( "BYYEARDAY" "=" byyrdaylist )
            / ( "BYWEEKNO" "=" bywknolist )
            / ( "BYMONTH" "=" bymolist )
            / ( "BYSETPOS" "=" bysplist )
            / ( "WKST" "=" weekday )

freq       = "SECONDLY" / "MINUTELY" / "HOURLY" / "DAILY"
          / "WEEKLY" / "MONTHLY" / "YEARLY"
enddate     = date / date-time
byseclist   = ( seconds *("," seconds) )
seconds     = 1*2DIGIT       ;0 to 60
byminlist   = ( minutes *("," minutes) )
minutes     = 1*2DIGIT       ;0 to 59
byhrlist    = ( hour *("," hour) )
hour        = 1*2DIGIT       ;0 to 23
bywdaylist  = ( weekdaynum *("," weekdaynum) )
weekdaynum  = [[plus / minus] ordwk] weekday
plus        = "+"
minus       = "-"
ordwk       = 1*2DIGIT       ;1 to 53
weekday     = "SU" / "MO" / "TU" / "WE" / "TH" / "FR" / "SA" ;Corresponding to SUNDAY,
MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and SATURDAY days of the week.
bymodaylist = ( monthdaynum *("," monthdaynum) )
monthdaynum = [plus / minus] ordmoday
ordmoday    = 1*2DIGIT       ;1 to 31
byyrdaylist = ( yeardaynum *("," yeardaynum) )
yeardaynum  = [plus / minus] ordyrday
ordyrday    = 1*3DIGIT      ;1 to 366
bywknolist  = ( weeknum *("," weeknum) )
weeknum     = [plus / minus] ordwk
```

```
bymolist    = ( monthnum *("," monthnum) )
monthnum    = 1*2DIGIT      ;1 to 12
bysplist    = ( setposday *("," setposday) )
setposday   = yeardaynum
```

This value type is a structured value consisting of a list of one or more recurrence grammar parts. Each rule part is defined by a NAME=VALUE pair. The rule parts are separated from each other by the SEMICOLON character. The rule parts are not ordered in any particular sequence. Individual rule parts MUST only be specified once. Compliant applications MUST accept rule parts ordered in any sequence, but to ensure backward compatibility with applications that pre-date this revision of iCalendar the FREQ rule part MUST be the first rule part specified in a RECUR value.

The FREQ rule part identifies the type of recurrence rule. This rule part MUST be specified in the recurrence rule. Valid values include SECONDLY, to specify repeating events based on an interval of a second or more; MINUTELY, to specify repeating events based on an interval of a minute or more; HOURLY, to specify repeating events based on an interval of an hour or more; DAILY, to specify repeating events based on an interval of a day or more; WEEKLY, to specify repeating events based on an interval of a week or more; MONTHLY, to specify repeating events based on an interval of a month or more; and YEARLY, to specify repeating events based on an interval of a year or more.

The INTERVAL rule part contains a positive integer representing at which intervals the recurrence rule repeats. The default value is "1", meaning every second for a SECONDLY rule, every minute for a MINUTELY rule, every hour for an HOURLY rule, every day for a DAILY rule, every week for a WEEKLY rule, every month for a MONTHLY rule, and every year for a YEARLY rule. For example, within a DAILY rule, a value of "8" means every eight days.

The UNTIL rule part defines a DATE or DATE-TIME value that bounds the recurrence rule in an inclusive manner. If the value specified by UNTIL is synchronized with the specified recurrence, this DATE or DATE-TIME becomes the last instance of the recurrence. The value of the UNTIL rule part MUST have the same value type as the "DTSTART" property. Furthermore, if the "DTSTART" property is specified as a date with local time, then the UNTIL rule part MUST also be specified as a date with local time. If the "DTSTART" property is specified as a date with UTC time or a date with local time and time zone reference, then the UNTIL rule part MUST be specified as a date with UTC time. In the case of the "STANDARD" and "DAYLIGHT" sub-components the UNTIL rule part MUST always be specified as a date with UTC time. If specified as a DATE-TIME value, then it MUST be specified in a UTC time format. If not present, and the COUNT rule part is also not present, the "RRULE" is considered to repeat forever.

The COUNT rule part defines the number of occurrences at which to range-bound the recurrence. The "DTSTART" property value always counts as the first occurrence.

The BYSECOND rule part specifies a COMMA-separated list of seconds within a minute. Valid values are 0 to 60. The BYMINUTE rule part specifies a COMMA-separated list of minutes within an hour. Valid values are 0 to 59. The BYHOUR rule part specifies a COMMA- separated list of hours of the day. Valid values are 0 to 23. The BYSECOND, BYMINUTE and BYHOUR rule parts MUST NOT be specified when the associated "DTSTART" property has a DATE value type. These rule parts MUST be ignored in RECUR value that violate the above requirement (e.g., generated by applications that pre-date this revision of iCalendar).

The BYDAY rule part specifies a COMMA-separated list of days of the week; SU indicates Sunday; MO indicates Monday; TU indicates Tuesday; WE indicates Wednesday; TH indicates Thursday; FR indicates Friday; and SA indicates Saturday. Each BYDAY value can also be preceded by a positive (+n) or negative (-n) integer. If present, this indicates the nth occurrence of a specific day within the MONTHLY or YEARLY "RRULE".
For example, within a MONTHLY rule, +1MO (or simply 1MO) represents the first Monday within the month, whereas -1MO represents the last Monday of the month. The numeric value in a BYDAY rule part with the FREQ rule part set to YEARLY corresponds to an offset within the month when the BYMONTH rule part is present, and corresponds to an offset within the year when the BYWEEKNO or BYMONTH rule parts are present. If an integer modifier is not present, it means all days of this type within the specified frequency. For example, within a MONTHLY rule, MO represents all Mondays within the month. The BYDAY rule part MUST NOT be specified with a numeric value when the FREQ rule part is not set to MONTHLY or YEARLY. Furthermore, the BYDAY rule part MUST NOT be specified with a numeric value with the FREQ rule part set to YEARLY when the BYWEEKNO rule part is specified.

The BYMONTHDAY rule part specifies a COMMA-separated list of days of the month. Valid values are 1 to 31 or -31 to -1. For example, -10 represents the tenth to the last day of the month. The BYMONTHDAY rule part MUST NOT be specified when the FREQ rule part is set to WEEKLY.

The BYYEARDAY rule part specifies a COMMA-separated list of days of the year. Valid values are 1 to 366 or -366 to -1. For example, -1 represents the last day of the year (December 31st) and -306 represents the 306th to the last day of the year (March 1st). The BYYEARDAY rule part MUST NOT be specified when the FREQ rule part is set to DAILY, WEEKLY, or MONTHLY.

The BYWEEKNO rule part specifies a COMMA-separated list of ordinals specifying weeks of the year. Valid values are 1 to 53 or -53 to -1. This corresponds to weeks according to week numbering as defined in [ISO.8601.2004]. A week is defined as a seven day period, starting on the day of the week defined to be the week start (see WKST). Week number one of the calendar year is the first week that contains at least four (4) days in that calendar year. This rule part MUST NOT be used when the FREQ rule part is set to anything other than YEARLY. For example, 3 represents the third week of the year.

Note: Assuming a Monday week start, week 53 can only occur when Thursday is January 1 or if it is a leap year and Wednesday is January 1.

The BYMONTH rule part specifies a COMMA-separated list of months of the year. Valid values are 1 to 12.

The WKST rule part specifies the day on which the workweek starts. Valid values are MO, TU, WE, TH, FR, SA, and SU. This is significant when a WEEKLY "RRULE" has an interval greater than 1, and a BYDAY rule part is specified. This is also significant when in a YEARLY "RRULE" when a BYWEEKNO rule part is specified. The default value is MO.

The BYSETPOS rule part specifies a COMMA-separated list of values that corresponds to the nth occurrence within the set of recurrence instances specified by the rule. BYSETPOS operates on a set of recurrence instances in one interval of the recurrence rule. For example, in a WEEKLY rule, the interval would be one week A set of recurrence instances starts at the beginning of the interval defined by the FREQ rule part. Valid values are 1 to 366 or -366 to -1. It MUST only be used in conjunction with another BYxxx rule part. For example "the last work day of the month" could be represented as:

```
FREQ=MONTHLY;BYDAY=MO,TU,WE,TH,FR;BYSETPOS=-1
```

Each BYSETPOS value can include a positive (+n) or negative (-n) integer. If present, this indicates the nth occurrence of the specific occurrence within the set of occurrences specified by the rule.

Recurrence rules may generate recurrence instances with an invalid date (e.g., February 30) or nonexistent local time (e.g., 1:30 AM on a day where the local time is moved forward by an hour at 1:00 AM). Such recurrence instances MUST be ignored and MUST NOT be counted as part of the recurrence set. Information, not contained in the rule, necessary to determine the various recurrence instance start time and dates are derived from the Start Time ("DTSTART") component attribute. For example, "FREQ=YEARLY;BYMONTH=1" doesn't specify a specific day within the month or a time. This information would be the same as what is specified for "DTSTART".

property ICalendar.RecurPartValue (Recur as String, Part as RecurPartEnum) as Variant

Returns the value of the giving part of the recur expression.

Type	Description
Recur as String	A String expression that specifies the recurrence rule according to RFC 5545. The Recur parameter must include DTSTART and FREQ rule parts, else the recurrence rule is not valid.
Part as RecurPartEnum	A RecurPartEnum expression that specifies the part of the recurrence rule to be requested
Variant	A VARIANT expression that specifies the value of the recurrence part.

The RecurPartValue property returns the value of the giving part of the recur expression. Use the RecurPartValue(exRecurSyntaxError) or RecurPartValue(exRecurSyntaxErrorInfo) to specify whether the Recur parameter is valid or invalid. The RecurAll property returns all occurrences of the giving recurrence rule, as a safe array. The RecurAllAsString property returns all occurrences of the giving recurrence rule, as a string. The RecurCheck property specifies whether the date fits the giving recurrence rule. The RecurRange property returns the occurrences of the giving recurrence rule between start and end dates (interval, range of dates), as a safe array. The RecurRangeAsString property returns the occurrences of the giving recurrence rule between start and end dates (interval, range of dates), as a string.

The BNF syntax for Recur parameter is:

```
recur = recur-rule-part *( ";" recur-rule-part )
   ;
   ; The rule parts are not ordered in any
   ; particular sequence.
   ;
   ; The FREQ rule part is REQUIRED,
   ; but MUST NOT occur more than once.
   ;
   ; The UNTIL or COUNT rule parts are OPTIONAL,
   ; but they MUST NOT occur in the same 'recur'.
   ;
   ; The other rule parts are OPTIONAL,
   ; but MUST NOT occur more than once.

recur-rule-part = ( "FREQ" "=" freq )
              / ( "UNTIL" "=" enddate )
              / ( "COUNT" "=" 1*DIGIT )
              / ( "INTERVAL" "=" 1*DIGIT )
              / ( "BYSECOND" "=" byseclist )
              / ( "BYMINUTE" "=" byminlist )
              / ( "BYHOUR" "=" byhrlist )
              / ( "BYDAY" "=" bywdaylist )
              / ( "BYMONTHDAY" "=" bymodaylist )
              / ( "BYYEARDAY" "=" byyrdaylist )
              / ( "BYWEEKNO" "=" bywknolist )
              / ( "BYMONTH" "=" bymolist )
              / ( "BYSETPOS" "=" bysplist )
              / ( "WKST" "=" weekday )

freq        = "SECONDLY" / "MINUTELY" / "HOURLY" / "DAILY"
           / "WEEKLY" / "MONTHLY" / "YEARLY"
enddate     = date / date-time
byseclist   = ( seconds *("," seconds) )
seconds     = 1*2DIGIT       ;0 to 60
byminlist   = ( minutes *("," minutes) )
minutes     = 1*2DIGIT       ;0 to 59
byhrlist    = ( hour *("," hour) )
hour        = 1*2DIGIT       ;0 to 23
bywdaylist  = ( weekdaynum *("," weekdaynum) )
weekdaynum  = [[plus / minus] ordwk] weekday
plus        = "+"
minus       = "-"
ordwk       = 1*2DIGIT       ;1 to 53
weekday     = "SU" / "MO" / "TU" / "WE" / "TH" / "FR" / "SA" ;Corresponding to SUNDAY,
MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and SATURDAY days of the week.
bymodaylist = ( monthdaynum *("," monthdaynum) )
monthdaynum = [plus / minus] ordmoday
ordmoday    = 1*2DIGIT       ;1 to 31
byyrdaylist = ( yeardaynum *("," yeardaynum) )
```

```
yeardaynum  = [plus / minus] ordyrday
ordyrday    = 1*3DIGIT      ;1 to 366
bywknolist  = ( weeknum *("," weeknum) )
weeknum     = [plus / minus] ordwk
bymolist    = ( monthnum *("," monthnum) )
monthnum    = 1*2DIGIT      ;1 to 12
bysplist    = ( setposday *("," setposday) )
setposday   = yeardaynum
```

The following samples show how you can check if the recurrence expression is syntactically correct:

VBA (MS Access, Excell...)

```
Set ICalendar1 = CreateObject("Exontrol.ICalendar.1")
With ICalendar1
   Debug.Print( "1.A) SyntaxError: " )
   Debug.Print( .RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO",-1)
)
   Debug.Print( "1.B) SyntaxErrorInfo: " )
   Debug.Print( .RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO",-2)
)
   Debug.Print( "2.A) SyntaxError: " )
   Debug.Print( .RecurPartValue("FREQ=DAILY;BYDAY=MO",-1) )
   Debug.Print( "2.B) SyntaxErrorInfo: " )
   Debug.Print( .RecurPartValue("FREQ=DAILY;BYDAY=MO",-2) )
End With
```

VB6

```
Set ICalendar1 = CreateObject("Exontrol.ICalendar.1")
With ICalendar1
   Debug.Print( "1.A) SyntaxError: " )
   Debug.Print(
.RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO",exRecurSyntaxError)
 )
   Debug.Print( "1.B) SyntaxErrorInfo: " )
   Debug.Print(
.RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO",exRecurSyntaxErrorI
 )
   Debug.Print( "2.A) SyntaxError: " )
   Debug.Print( .RecurPartValue("FREQ=DAILY;BYDAY=MO",exRecurSyntaxError) )
   Debug.Print( "2.B) SyntaxErrorInfo: " )
   Debug.Print( .RecurPartValue("FREQ=DAILY;BYDAY=MO",exRecurSyntaxErrorInfo)
)
End With
```

VB.NET

```
' Add 'exontrol.exicalendar.dll(ExICalendar.dll)' reference to your project.
Exicalendar1 = New exontrol.EXICALENDARLib.exicalendar()
With Exicalendar1
   Debug.Print( "1.A) SyntaxError: " )
   Debug.Print(
.get_RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO",exontrol.EXICAL
 )
   Debug.Print( "1.B) SyntaxErrorInfo: " )
   Debug.Print(
.get_RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO",exontrol.EXICAL
 )
   Debug.Print( "2.A) SyntaxError: " )
   Debug.Print(
.get_RecurPartValue("FREQ=DAILY;BYDAY=MO",exontrol.EXICALENDARLib.RecurPartE
 )
   Debug.Print( "2.B) SyntaxErrorInfo: " )
   Debug.Print(
.get_RecurPartValue("FREQ=DAILY;BYDAY=MO",exontrol.EXICALENDARLib.RecurPartE
 )
End With
```

VB.NET for /COM

```
AxICalendar1 = CreateObject("Exontrol.ICalendar.1")
With AxICalendar1
   Debug.Print( "1.A) SyntaxError: " )
   Debug.Print(
.get_RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO",EXICALENDARLi
)
   Debug.Print( "1.B) SyntaxErrorInfo: " )
   Debug.Print(
.get_RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO",EXICALENDARLi
)
   Debug.Print( "2.A) SyntaxError: " )
   Debug.Print(
.get_RecurPartValue("FREQ=DAILY;BYDAY=MO",EXICALENDARLib.RecurPartEnum.exR
)
   Debug.Print( "2.B) SyntaxErrorInfo: " )
   Debug.Print(
.get_RecurPartValue("FREQ=DAILY;BYDAY=MO",EXICALENDARLib.RecurPartEnum.exR
)
End With
```

C++

```
/*
    Includes the definition for CreateObject function like follows:

    #include <comdef.h>
    IUnknownPtr CreateObject( BSTR Object )
    {
        IUnknownPtr spResult;
        spResult.CreateInstance( Object );
        return spResult;
    };

*/
/*
    Copy and paste the following directives to your header file as
    it defines the namespace 'EXICALENDARLib' for the library: 'ICalendar 1.0 Type
Library'

    #import <ExICalendar.dll>
    using namespace EXICALENDARLib;
```

```
*/
EXICALENDARLib::IICalendarPtr spICalendar1 =
::CreateObject(L"Exontrol.ICalendar.1");
OutputDebugStringW( L"1.A) SyntaxError: " );
OutputDebugStringW( _bstr_t(spICalendar1-
>GetRecurPartValue(L"DTSTART=20151205;FREQ=DAILY;BYDAY=MO",EXICALENDAR
 );
OutputDebugStringW( L"1.B) SyntaxErrorInfo: " );
OutputDebugStringW( _bstr_t(spICalendar1-
>GetRecurPartValue(L"DTSTART=20151205;FREQ=DAILY;BYDAY=MO",EXICALENDAR
 );
OutputDebugStringW( L"2.A) SyntaxError: " );
OutputDebugStringW( _bstr_t(spICalendar1-
>GetRecurPartValue(L"FREQ=DAILY;BYDAY=MO",EXICALENDARLib::exRecurSyntaxErr
 );
OutputDebugStringW( L"2.B) SyntaxErrorInfo: " );
OutputDebugStringW( _bstr_t(spICalendar1-
>GetRecurPartValue(L"FREQ=DAILY;BYDAY=MO",EXICALENDARLib::exRecurSyntaxErr
 );
```

C++ Builder

```
Exicalendarlib_tlb::IICalendarPtr ICalendar1 =
Variant::CreateObject(L"Exontrol.ICalendar.1");
OutputDebugString( L"1.A) SyntaxError: " );
OutputDebugString( PChar(ICalendar1-
>RecurPartValue[L"DTSTART=20151205;FREQ=DAILY;BYDAY=MO",Exicalendarlib_tlb::
 );
OutputDebugString( L"1.B) SyntaxErrorInfo: " );
OutputDebugString( PChar(ICalendar1-
>RecurPartValue[L"DTSTART=20151205;FREQ=DAILY;BYDAY=MO",Exicalendarlib_tlb::
 );
OutputDebugString( L"2.A) SyntaxError: " );
OutputDebugString( PChar(ICalendar1-
>RecurPartValue[L"FREQ=DAILY;BYDAY=MO",Exicalendarlib_tlb::RecurPartEnum::exRe
 );
```

```
OutputDebugString( L"2.B) SyntaxErrorInfo: " );
OutputDebugString( PChar(ICalendar1-
>RecurPartValue[L"FREQ=DAILY;BYDAY=MO",Exicalendarlib_tlb::RecurPartEnum::exRe
 );
```

C#

```
// Add 'exontrol.exicalendar.dll(ExICalendar.dll)' reference to your project.
exontrol.EXICALENDARLib.exicalendar exicalendar1 = new
exontrol.EXICALENDARLib.exicalendar();
System.Diagnostics.Debug.Print( "1.A) SyntaxError: " );
System.Diagnostics.Debug.Print(
exicalendar1.get_RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO",exo
 );
System.Diagnostics.Debug.Print( "1.B) SyntaxErrorInfo: " );
System.Diagnostics.Debug.Print(
exicalendar1.get_RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO",exo
 );
System.Diagnostics.Debug.Print( "2.A) SyntaxError: " );
System.Diagnostics.Debug.Print(
exicalendar1.get_RecurPartValue("FREQ=DAILY;BYDAY=MO",exontrol.EXICALENDARL
 );
System.Diagnostics.Debug.Print( "2.B) SyntaxErrorInfo: " );
System.Diagnostics.Debug.Print(
exicalendar1.get_RecurPartValue("FREQ=DAILY;BYDAY=MO",exontrol.EXICALENDARL
 );
```

JScript/JavaScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:D6C87100-38E2-4ABB-8AC2-4C0097AEE2D6"
id="ICalendar1"></OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    alert( "1.A) SyntaxError: " );
    alert(
ICalendar1.RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO",-1) );
    alert( "1.B) SyntaxErrorInfo: " );
    alert(
ICalendar1.RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO",-2) );
    alert( "2.A) SyntaxError: " );
    alert( ICalendar1.RecurPartValue("FREQ=DAILY;BYDAY=MO",-1) );
    alert( "2.B) SyntaxErrorInfo: " );
    alert( ICalendar1.RecurPartValue("FREQ=DAILY;BYDAY=MO",-2) );
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:D6C87100-38E2-4ABB-8AC2-4C0097AEE2D6"
id="ICalendar1"></OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With ICalendar1
        alert( "1.A) SyntaxError: " )
        alert( .RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO",-1) )
        alert( "1.B) SyntaxErrorInfo: " )
        alert( .RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO",-2) )
        alert( "2.A) SyntaxError: " )
        alert( .RecurPartValue("FREQ=DAILY;BYDAY=MO",-1) )
        alert( "2.B) SyntaxErrorInfo: " )
        alert( .RecurPartValue("FREQ=DAILY;BYDAY=MO",-2) )
    End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
EXICALENDARLib.ICalendar axICalendar1 = new EXICALENDARLib.ICalendar();
System.Diagnostics.Debug.Print( "1.A) SyntaxError: " );
System.Diagnostics.Debug.Print(
axICalendar1.get_RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO",EXI
 );
System.Diagnostics.Debug.Print( "1.B) SyntaxErrorInfo: " );
System.Diagnostics.Debug.Print(
axICalendar1.get_RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO",EXI
 );
System.Diagnostics.Debug.Print( "2.A) SyntaxError: " );
System.Diagnostics.Debug.Print(
axICalendar1.get_RecurPartValue("FREQ=DAILY;BYDAY=MO",EXICALENDARLib.RecurP
 );
System.Diagnostics.Debug.Print( "2.B) SyntaxErrorInfo: " );
System.Diagnostics.Debug.Print(
axICalendar1.get_RecurPartValue("FREQ=DAILY;BYDAY=MO",EXICALENDARLib.RecurP
 );
```

X++ (Dynamics Ax 2009)

```
public void init()
{
    COM com_exicalendar1;
    anytype exicalendar1;
    ;

    super();

    // Add 'exicalendar.dll(ExICalendar.dll)' reference to your project.
    exicalendar1 = COM::createFromObject(new EXICALENDARLib.exicalendar());
com_exicalendar1 = exicalendar1;
    print( "1.A) SyntaxError: " );
    print(
com_exicalendar1.RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO",-1,
 );
```

```
    print( "1.B) SyntaxErrorInfo: " );
    print(
com_exicalendar1.RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO",-2/
 );
    print( "2.A) SyntaxError: " );
    print(
com_exicalendar1.RecurPartValue("FREQ=DAILY;BYDAY=MO",-1/*exRecurSyntaxError
 );
    print( "2.B) SyntaxErrorInfo: " );
    print(
com_exicalendar1.RecurPartValue("FREQ=DAILY;BYDAY=MO",-2/*exRecurSyntaxErrorI
 );
}
```

Delphi 8 (.NET only)

```
AxICalendar1 :=
(ComObj.CreateComObject(ComObj.ProgIDToClassID('Exontrol.ICalendar.1')) as
EXICALENDARLib.ICalendar);
with AxICalendar1 do
begin
    OutputDebugString( '1.A) SyntaxError: ' );
    OutputDebugString(
get_RecurPartValue('DTSTART=20151205;FREQ=DAILY;BYDAY=MO',EXICALENDARLib
 );
    OutputDebugString( '1.B) SyntaxErrorInfo: ' );
    OutputDebugString(
get_RecurPartValue('DTSTART=20151205;FREQ=DAILY;BYDAY=MO',EXICALENDARLib
 );
    OutputDebugString( '2.A) SyntaxError: ' );
    OutputDebugString(
get_RecurPartValue('FREQ=DAILY;BYDAY=MO',EXICALENDARLib.RecurPartEnum.exRe
 );
    OutputDebugString( '2.B) SyntaxErrorInfo: ' );
    OutputDebugString(
get_RecurPartValue('FREQ=DAILY;BYDAY=MO',EXICALENDARLib.RecurPartEnum.exRe
 );
```

```
end
```

Delphi (standard)

```
ICalendar1 :=
(IUnknown(ComObj.CreateComObject(ComObj.ProgIDToClassID('Exontrol.ICalendar.1'))
 as EXICALENDARLib_TLB.ICalendar);
with ICalendar1 do
begin
   OutputDebugString( '1.A) SyntaxError: ' );
   OutputDebugString(
RecurPartValue['DTSTART=20151205;FREQ=DAILY;BYDAY=MO',EXICALENDARLib_TLB
 );
   OutputDebugString( '1.B) SyntaxErrorInfo: ' );
   OutputDebugString(
RecurPartValue['DTSTART=20151205;FREQ=DAILY;BYDAY=MO',EXICALENDARLib_TLB
 );
   OutputDebugString( '2.A) SyntaxError: ' );
   OutputDebugString(
RecurPartValue['FREQ=DAILY;BYDAY=MO',EXICALENDARLib_TLB.exRecurSyntaxError]
 );
   OutputDebugString( '2.B) SyntaxErrorInfo: ' );
   OutputDebugString(
RecurPartValue['FREQ=DAILY;BYDAY=MO',EXICALENDARLib_TLB.exRecurSyntaxErrorI
 );
end
```

VFP

```
thisform.ICalendar1 = CreateObject("Exontrol.ICalendar.1")
with thisform.ICalendar1
   DEBUGOUT( "1.A) SyntaxError: " )
   DEBUGOUT( .RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO",-1) )
   DEBUGOUT( "1.B) SyntaxErrorInfo: " )
   DEBUGOUT( .RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO",-2) )
   DEBUGOUT( "2.A) SyntaxError: " )
   DEBUGOUT( .RecurPartValue("FREQ=DAILY;BYDAY=MO",-1) )
   DEBUGOUT( "2.B) SyntaxErrorInfo: " )
```

```
    DEBUGOUT( .RecurPartValue("FREQ=DAILY;BYDAY=MO",-2) )
endwith
```

dBASE Plus

```
local oICalendar

oICalendar = new OleAutoClient("Exontrol.ICalendar.1")

? "1.A) SyntaxError: "
? Str(oICalendar.RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO",-1))
? "1.B) SyntaxErrorInfo: "
? Str(oICalendar.RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO",-2))
? "2.A) SyntaxError: "
? Str(oICalendar.RecurPartValue("FREQ=DAILY;BYDAY=MO",-1))
? "2.B) SyntaxErrorInfo: "
? Str(oICalendar.RecurPartValue("FREQ=DAILY;BYDAY=MO",-2))
```

XBasic (Alpha Five)

```
Dim oICalendar as P

oICalendar = OLE.Create("Exontrol.ICalendar.1")

? "1.A) SyntaxError: "
? oICalendar.RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO",-1)
? "1.B) SyntaxErrorInfo: "
? oICalendar.RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO",-2)
? "2.A) SyntaxError: "
? oICalendar.RecurPartValue("FREQ=DAILY;BYDAY=MO",-1)
? "2.B) SyntaxErrorInfo: "
? oICalendar.RecurPartValue("FREQ=DAILY;BYDAY=MO",-2)
```

Visual Objects

```
oDCOCX_Exontrol1 := IICalendar{"Exontrol.ICalendar.1"}
OutputDebugString(String2Psz( "1.A) SyntaxError: " ))
OutputDebugString(String2Psz( AsString(oDCOCX_Exontrol1:
[RecurPartValue,"DTSTART=20151205;FREQ=DAILY;BYDAY=MO",exRecurSyntaxError])
))
OutputDebugString(String2Psz( "1.B) SyntaxErrorInfo: " ))
OutputDebugString(String2Psz( AsString(oDCOCX_Exontrol1:
[RecurPartValue,"DTSTART=20151205;FREQ=DAILY;BYDAY=MO",exRecurSyntaxErrorInf
 ))
OutputDebugString(String2Psz( "2.A) SyntaxError: " ))
OutputDebugString(String2Psz( AsString(oDCOCX_Exontrol1:
[RecurPartValue,"FREQ=DAILY;BYDAY=MO",exRecurSyntaxError]) ))
OutputDebugString(String2Psz( "2.B) SyntaxErrorInfo: " ))
OutputDebugString(String2Psz( AsString(oDCOCX_Exontrol1:
[RecurPartValue,"FREQ=DAILY;BYDAY=MO",exRecurSyntaxErrorInfo]) ))
```

PowerBuilder

```
OleObject oICalendar

oICalendar = CREATE OLEObject
oICalendar.ConnectToNewObject("Exontrol.ICalendar.1")

MessageBox("Information",string( "1.A) SyntaxError: " ))
MessageBox("Information",string(
String(oICalendar.RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO",-1)
 ))
MessageBox("Information",string( "1.B) SyntaxErrorInfo: " ))
MessageBox("Information",string(
String(oICalendar.RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO",-2)
 ))
MessageBox("Information",string( "2.A) SyntaxError: " ))
MessageBox("Information",string(
String(oICalendar.RecurPartValue("FREQ=DAILY;BYDAY=MO",-1)) ))
MessageBox("Information",string( "2.B) SyntaxErrorInfo: " ))
MessageBox("Information",string(
String(oICalendar.RecurPartValue("FREQ=DAILY;BYDAY=MO",-2)) ))
```

Visual DataFlex

```
Procedure OnCreate
   Forward Send OnCreate
   Variant oComICalendar1
   Get ComCreateObject "Exontrol.ICalendar.1" to oComICalendar1

   Showln "1.A) SyntaxError: "
(ComRecurPartValue(Self,"DTSTART=20151205;FREQ=DAILY;BYDAY=MO",OLEexRecur

   Showln "1.B) SyntaxErrorInfo: "
(ComRecurPartValue(Self,"DTSTART=20151205;FREQ=DAILY;BYDAY=MO",OLEexRecur

   Showln "2.A) SyntaxError: "
(ComRecurPartValue(Self,"FREQ=DAILY;BYDAY=MO",OLEexRecurSyntaxError))
   Showln "2.B) SyntaxErrorInfo: "
(ComRecurPartValue(Self,"FREQ=DAILY;BYDAY=MO",OLEexRecurSyntaxErrorInfo))
End_Procedure
```

XBase++

```
#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
   LOCAL oForm
   LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
   LOCAL oICalendar

   oForm := XbpDialog():new( AppDesktop() )
   oForm:drawingArea:clipChildren := .T.
   oForm:create( ,,{100,100}, {640,480},, .F. )
   oForm:close  := {|| PostAppEvent( xbeP_Quit )}

   oICalendar := XbpActiveXControl():new( oForm:drawingArea )
   oICalendar:CLSID  := "Exontrol.ICalendar.1" /*{D6C87100-38E2-4ABB-8AC2-
4C0097AEE2D6}*/
   oICalendar:create(,, {10,60},{610,370} )

      DevOut( "1.A) SyntaxError: " )
      DevOut(
Transform(oICalendar:RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO
 )
      DevOut( "1.B) SyntaxErrorInfo: " )
      DevOut(
Transform(oICalendar:RecurPartValue("DTSTART=20151205;FREQ=DAILY;BYDAY=MO
 )
      DevOut( "2.A) SyntaxError: " )
      DevOut(
Transform(oICalendar:RecurPartValue("FREQ=DAILY;BYDAY=MO",-1/*exRecurSyntaxEr
 )
      DevOut( "2.B) SyntaxErrorInfo: " )
      DevOut(
Transform(oICalendar:RecurPartValue("FREQ=DAILY;BYDAY=MO",-2/*exRecurSyntaxEr
 )

   oForm:Show()
   DO WHILE nEvent != xbeP_Quit
      nEvent := AppEvent( @mp1, @mp2, @oXbp )
      oXbp:handleEvent( nEvent, mp1, mp2 )
   ENDDO
RETURN
```

property ICalendar.RecurRange (Recur as String, Start as Date, End as Date) as Variant

Returns all occurrences between start and and of the specified rule, as a safe array of dates.

Type	Description
Recur as String	A String expression that specifies the recurrence rule according to RFC 5545. The Recur parameter must include DTSTART and FREQ rule parts, else the recurrence rule is not valid.
Start as Date	A DATE expression that indicates the starting date of the range.
End as Date	A DATE expression that indicates the ending date of the range.
Variant	A safe array of dates/occurrences between start and and of the specified rule. You can use the **for each** statement to enumerates the dates in the safe array.

The RecurRange property returns the occurrences of the giving recurrence rule between start and end dates (interval, range of dates), as a safe array. The RecurRangeAsString property returns the occurrences of the giving recurrence rule between start and end dates (interval, range of dates), as a string. The RecurAll property returns all occurrences of the giving recurrence rule, as a safe array. The RecurAllAsString property returns all occurrences of the giving recurrence rule, as a string. The RecurCheck property specifies whether the date fits the giving recurrence rule. Use the RecurPartValue(exRecurSyntaxError) or RecurPartValue(exRecurSyntaxErrorInfo) to specify whether the Recur parameter is valid or invalid.

The BNF syntax for Recur parameter is:

```
recur = recur-rule-part *( ";" recur-rule-part )
   ;
   ; The rule parts are not ordered in any
   ; particular sequence.
   ;
   ; The FREQ rule part is REQUIRED,
   ; but MUST NOT occur more than once.
   ;
   ; The UNTIL or COUNT rule parts are OPTIONAL,
    ; but they MUST NOT occur in the same 'recur'.
    ;
    ; The other rule parts are OPTIONAL,
    ; but MUST NOT occur more than once.

recur-rule-part = ( "FREQ" "=" freq )
                / ( "UNTIL" "=" enddate )
                / ( "COUNT" "=" 1*DIGIT )
                / ( "INTERVAL" "=" 1*DIGIT )
                / ( "BYSECOND" "=" byseclist )
                / ( "BYMINUTE" "=" byminlist )
                / ( "BYHOUR" "=" byhrlist )
                / ( "BYDAY" "=" bywdaylist )
                / ( "BYMONTHDAY" "=" bymodaylist )
                / ( "BYYEARDAY" "=" byyrdaylist )
                / ( "BYWEEKNO" "=" bywknolist )
                / ( "BYMONTH" "=" bymolist )
                / ( "BYSETPOS" "=" bysplist )
                / ( "WKST" "=" weekday )

freq        = "SECONDLY" / "MINUTELY" / "HOURLY" / "DAILY"
            / "WEEKLY" / "MONTHLY" / "YEARLY"
enddate     = date / date-time
byseclist   = ( seconds *("," seconds) )
seconds     = 1*2DIGIT       ;0 to 60
byminlist   = ( minutes *("," minutes) )
minutes     = 1*2DIGIT       ;0 to 59
byhrlist    = ( hour *("," hour) )
hour        = 1*2DIGIT       ;0 to 23
bywdaylist  = ( weekdaynum *("," weekdaynum) )
weekdaynum  = [[plus / minus] ordwk] weekday
plus        = "+"
minus       = "-"
ordwk       = 1*2DIGIT       ;1 to 53
weekday     = "SU" / "MO" / "TU" / "WE" / "TH" / "FR" / "SA" ;Corresponding to SUNDAY,
MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and SATURDAY days of the week.
bymodaylist = ( monthdaynum *("," monthdaynum) )
```

```
monthdaynum = [plus / minus] ordmoday
ordmoday    = 1*2DIGIT      ;1 to 31
byyrdaylist = ( yeardaynum *("," yeardaynum) )
yeardaynum  = [plus / minus] ordyrday
ordyrday    = 1*3DIGIT     ;1 to 366
bywknolist  = ( weeknum *("," weeknum) )
weeknum     = [plus / minus] ordwk
bymolist    = ( monthnum *("," monthnum) )
monthnum    = 1*2DIGIT      ;1 to 12
bysplist    = ( setposday *("," setposday) )
setposday   = yeardaynum
```

This value type is a structured value consisting of a list of one or more recurrence grammar parts. Each rule part is defined by a NAME=VALUE pair. The rule parts are separated from each other by the SEMICOLON character. The rule parts are not ordered in any particular sequence. Individual rule parts MUST only be specified once. Compliant applications MUST accept rule parts ordered in any sequence, but to ensure backward compatibility with applications that pre-date this revision of iCalendar the FREQ rule part MUST be the first rule part specified in a RECUR value.

The FREQ rule part identifies the type of recurrence rule. This rule part MUST be specified in the recurrence rule. Valid values include SECONDLY, to specify repeating events based on an interval of a second or more; MINUTELY, to specify repeating events based on an interval of a minute or more; HOURLY, to specify repeating events based on an interval of an hour or more; DAILY, to specify repeating events based on an interval of a day or more; WEEKLY, to specify repeating events based on an interval of a week or more; MONTHLY, to specify repeating events based on an interval of a month or more; and YEARLY, to specify repeating events based on an interval of a year or more.

The INTERVAL rule part contains a positive integer representing at which intervals the recurrence rule repeats. The default value is "1", meaning every second for a SECONDLY rule, every minute for a MINUTELY rule, every hour for an HOURLY rule, every day for a DAILY rule, every week for a WEEKLY rule, every month for a MONTHLY rule, and every year for a YEARLY rule. For example, within a DAILY rule, a value of "8" means every eight days.

The UNTIL rule part defines a DATE or DATE-TIME value that bounds the recurrence rule in an inclusive manner. If the value specified by UNTIL is synchronized with the specified recurrence, this DATE or DATE-TIME becomes the last instance of the recurrence. The value of the UNTIL rule part MUST have the same value type as the "DTSTART" property. Furthermore, if the "DTSTART" property is specified as a date with local time, then the UNTIL rule part MUST also be specified as a date with local time. If the "DTSTART" property is specified as a date with UTC time or a date with local time and time zone reference, then the UNTIL rule part MUST be specified as a date with UTC time. In the case of the "STANDARD" and "DAYLIGHT" sub-components the UNTIL rule part MUST always be specified as a date with UTC time. If specified as a DATE-TIME value, then it MUST be specified in a UTC time format. If not present, and the COUNT rule part is also not present, the "RRULE" is considered to repeat forever.

The COUNT rule part defines the number of occurrences at which to range-bound the recurrence. The "DTSTART" property value always counts as the first occurrence.

The BYSECOND rule part specifies a COMMA-separated list of seconds within a minute. Valid values are 0 to 60. The BYMINUTE rule part specifies a COMMA-separated list of minutes within an hour. Valid values are 0 to 59. The BYHOUR rule part specifies a COMMA- separated list of hours of the day. Valid values are 0 to 23. The BYSECOND, BYMINUTE and BYHOUR rule parts MUST NOT be specified when the associated "DTSTART" property has a DATE value type. These rule parts MUST be ignored in RECUR value that violate the above requirement (e.g., generated by applications that pre-date this revision of iCalendar).

The BYDAY rule part specifies a COMMA-separated list of days of the week; SU indicates Sunday; MO indicates Monday; TU indicates Tuesday; WE indicates Wednesday; TH indicates Thursday; FR indicates Friday; and SA indicates Saturday. Each BYDAY value can also be preceded by a positive (+n) or negative (-n) integer. If present, this indicates the nth occurrence of a specific day within the MONTHLY or YEARLY "RRULE".
For example, within a MONTHLY rule, +1MO (or simply 1MO) represents the first Monday within the month, whereas -1MO represents the last Monday of the month. The numeric value in a BYDAY rule part with the FREQ rule part set to YEARLY corresponds to an offset within the month when the BYMONTH rule part is present, and corresponds to an offset within the year when the BYWEEKNO or BYMONTH rule parts are present. If an integer modifier is not present, it means all days of this type within the specified frequency. For example, within a MONTHLY rule, MO represents all Mondays within the month. The BYDAY rule part MUST NOT be specified with a numeric value when the FREQ rule part is not set to MONTHLY or YEARLY. Furthermore, the BYDAY rule part MUST NOT be specified with a numeric value with the FREQ rule part set to YEARLY when the BYWEEKNO rule part is specified.

The BYMONTHDAY rule part specifies a COMMA-separated list of days of the month. Valid values are 1 to 31 or -31 to -1. For example, -10 represents the tenth to the last day of the month. The BYMONTHDAY rule part MUST NOT be specified when the FREQ rule part is set to WEEKLY.

The BYYEARDAY rule part specifies a COMMA-separated list of days of the year. Valid values are 1 to 366 or -366 to -1. For example, -1 represents the last day of the year (December 31st) and -306 represents the 306th to the last day of the year (March 1st).

The BYYEARDAY rule part MUST NOT be specified when the FREQ rule part is set to DAILY, WEEKLY, or MONTHLY.

The BYWEEKNO rule part specifies a COMMA-separated list of ordinals specifying weeks of the year. Valid values are 1 to 53 or -53 to -1. This corresponds to weeks according to week numbering as defined in [ISO.8601.2004]. A week is defined as a seven day period, starting on the day of the week defined to be the week start (see WKST). Week number one of the calendar year is the first week that contains at least four (4) days in that calendar year. This rule part MUST NOT be used when the FREQ rule part is set to anything other than YEARLY. For example, 3 represents the third week of the year.

> Note: Assuming a Monday week start, week 53 can only occur when Thursday is January 1 or if it is a leap year and Wednesday is January 1.

The BYMONTH rule part specifies a COMMA-separated list of months of the year. Valid values are 1 to 12.

The WKST rule part specifies the day on which the workweek starts. Valid values are MO, TU, WE, TH, FR, SA, and SU. This is significant when a WEEKLY "RRULE" has an interval greater than 1, and a BYDAY rule part is specified. This is also significant when in a YEARLY "RRULE" when a BYWEEKNO rule part is specified. The default value is MO.

The BYSETPOS rule part specifies a COMMA-separated list of values that corresponds to the nth occurrence within the set of recurrence instances specified by the rule. BYSETPOS operates on a set of recurrence instances in one interval of the recurrence rule. For example, in a WEEKLY rule, the interval would be one week A set of recurrence instances starts at the beginning of the interval defined by the FREQ rule part. Valid values are 1 to 366 or -366 to -1. It MUST only be used in conjunction with another BYxxx rule part. For example "the last work day of the month" could be represented as:

```
FREQ=MONTHLY;BYDAY=MO,TU,WE,TH,FR;BYSETPOS=-1
```

Each BYSETPOS value can include a positive (+n) or negative (-n) integer. If present, this indicates the nth occurrence of the specific occurrence within the set of occurrences specified by the rule.

Recurrence rules may generate recurrence instances with an invalid date (e.g., February 30) or nonexistent local time (e.g., 1:30 AM on a day where the local time is moved forward by an hour at 1:00 AM). Such recurrence instances MUST be ignored and MUST NOT be counted as part of the recurrence set. Information, not contained in the rule, necessary to determine the various recurrence instance start time and dates are derived from the Start Time ("DTSTART") component attribute. For example, "FREQ=YEARLY;BYMONTH=1" doesn't specify a specific day within the month or a time. This information would be the same as what is specified for "DTSTART".

property ICalendar.RecurRangeAsString (Recur as String, Start as Date, End as Date) as Variant

Returns all occurrences between start and and of the specified rule, as string (dates are separated by new line).

Type	Description
Recur as String	A String expression that specifies the recurrence rule according to RFC 5545. The Recur parameter must include DTSTART and FREQ rule parts, else the recurrence rule is not valid.
Start as Date	A DATE expression that indicates the starting date of the range.
End as Date	A DATE expression that indicates the ending date of the range.
Variant	A String expression of dates that specifies the occurrences of the recurrence rule between start and end dates (separated by crlf "\r\n" sequence).

The RecurRangeAsString property returns the occurrences of the giving recurrence rule between start and end dates (interval, range of dates), as a string. The RecurRange property returns the occurrences of the giving recurrence rule between start and end dates (interval, range of dates), as a safe array. The RecurAllAsString property returns all occurrences of the giving recurrence rule, as a string. The RecurAll property returns all occurrences of the giving recurrence rule, as a safe array. The RecurCheck property specifies whether the date fits the giving recurrence rule. Use the RecurPartValue(exRecurSyntaxError) or RecurPartValue(exRecurSyntaxErrorInfo) to specify whether the Recur parameter is valid or invalid.

The BNF syntax for Recur parameter is:

```
recur = recur-rule-part *( ";" recur-rule-part )
  ;
  ; The rule parts are not ordered in any
  ; particular sequence.
  ;
  ; The FREQ rule part is REQUIRED,
  ; but MUST NOT occur more than once.
  ;
  ; The UNTIL or COUNT rule parts are OPTIONAL,
  ; but they MUST NOT occur in the same 'recur'.
```

```
    ;
    ; The other rule parts are OPTIONAL,
    ; but MUST NOT occur more than once.

recur-rule-part = ( "FREQ" "=" freq )
               / ( "UNTIL" "=" enddate )
               / ( "COUNT" "=" 1*DIGIT )
               / ( "INTERVAL" "=" 1*DIGIT )
               / ( "BYSECOND" "=" byseclist )
               / ( "BYMINUTE" "=" byminlist )
               / ( "BYHOUR" "=" byhrlist )
               / ( "BYDAY" "=" bywdaylist )
               / ( "BYMONTHDAY" "=" bymodaylist )
               / ( "BYYEARDAY" "=" byyrdaylist )
               / ( "BYWEEKNO" "=" bywknolist )
               / ( "BYMONTH" "=" bymolist )
               / ( "BYSETPOS" "=" bysplist )
               / ( "WKST" "=" weekday )

freq        = "SECONDLY" / "MINUTELY" / "HOURLY" / "DAILY"
           / "WEEKLY" / "MONTHLY" / "YEARLY"
enddate     = date / date-time
byseclist   = ( seconds *("," seconds) )
seconds     = 1*2DIGIT       ;0 to 60
byminlist   = ( minutes *("," minutes) )
minutes     = 1*2DIGIT       ;0 to 59
byhrlist    = ( hour *("," hour) )
hour        = 1*2DIGIT       ;0 to 23
bywdaylist  = ( weekdaynum *("," weekdaynum) )
weekdaynum  = [[plus / minus] ordwk] weekday
plus        = "+"
minus       = "-"
ordwk       = 1*2DIGIT       ;1 to 53
weekday     = "SU" / "MO" / "TU" / "WE" / "TH" / "FR" / "SA" ;Corresponding to SUNDAY,
MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and SATURDAY days of the week.
bymodaylist = ( monthdaynum *("," monthdaynum) )
monthdaynum = [plus / minus] ordmoday
```

```
ordmoday    = 1*2DIGIT       ;1 to 31
byyrdaylist = ( yeardaynum *("," yeardaynum) )
yeardaynum  = [plus / minus] ordyrday
ordyrday    = 1*3DIGIT      ;1 to 366
bywknolist  = ( weeknum *("," weeknum) )
weeknum     = [plus / minus] ordwk
bymolist    = ( monthnum *("," monthnum) )
monthnum    = 1*2DIGIT       ;1 to 12
bysplist    = ( setposday *("," setposday) )
setposday   = yeardaynum
```

This value type is a structured value consisting of a list of one or more recurrence grammar parts. Each rule part is defined by a NAME=VALUE pair. The rule parts are separated from each other by the SEMICOLON character. The rule parts are not ordered in any particular sequence. Individual rule parts MUST only be specified once. Compliant applications MUST accept rule parts ordered in any sequence, but to ensure backward compatibility with applications that pre-date this revision of iCalendar the FREQ rule part MUST be the first rule part specified in a RECUR value.

The FREQ rule part identifies the type of recurrence rule. This rule part MUST be specified in the recurrence rule. Valid values include SECONDLY, to specify repeating events based on an interval of a second or more; MINUTELY, to specify repeating events based on an interval of a minute or more; HOURLY, to specify repeating events based on an interval of an hour or more; DAILY, to specify repeating events based on an interval of a day or more; WEEKLY, to specify repeating events based on an interval of a week or more; MONTHLY, to specify repeating events based on an interval of a month or more; and YEARLY, to specify repeating events based on an interval of a year or more.

The INTERVAL rule part contains a positive integer representing at which intervals the recurrence rule repeats. The default value is "1", meaning every second for a SECONDLY rule, every minute for a MINUTELY rule, every hour for an HOURLY rule, every day for a DAILY rule, every week for a WEEKLY rule, every month for a MONTHLY rule, and every year for a YEARLY rule. For example, within a DAILY rule, a value of "8" means every eight days.

The UNTIL rule part defines a DATE or DATE-TIME value that bounds the recurrence rule in an inclusive manner. If the value specified by UNTIL is synchronized with the specified recurrence, this DATE or DATE-TIME becomes the last instance of the recurrence. The value of the UNTIL rule part MUST have the same value type as the "DTSTART" property. Furthermore, if the "DTSTART" property is specified as a date with local time, then the UNTIL rule part MUST also be specified as a date with local time. If the "DTSTART" property is specified as a date with UTC time or a date with local time and time zone reference, then the UNTIL rule part MUST be specified as a date with UTC time. In the case of the "STANDARD" and "DAYLIGHT" sub-components the UNTIL rule part MUST always be specified as a date with UTC time. If specified as a DATE-TIME value, then it MUST be specified in a UTC time format. If not present, and the COUNT rule part is also not present, the "RRULE" is considered to repeat forever.

The COUNT rule part defines the number of occurrences at which to range-bound the recurrence. The "DTSTART" property value always counts as the first occurrence.

The BYSECOND rule part specifies a COMMA-separated list of seconds within a minute. Valid values are 0 to 60. The BYMINUTE rule part specifies a COMMA-separated list of minutes within an hour. Valid values are 0 to 59. The BYHOUR rule part specifies a COMMA- separated list of hours of the day. Valid values are 0 to 23. The BYSECOND, BYMINUTE and BYHOUR rule parts MUST NOT be specified when the associated "DTSTART" property has a DATE value type. These rule parts MUST be ignored in RECUR value that violate the above requirement (e.g., generated by applications that pre-date this revision of iCalendar).

The BYDAY rule part specifies a COMMA-separated list of days of the week; SU indicates Sunday; MO indicates Monday; TU indicates Tuesday; WE indicates Wednesday; TH indicates Thursday; FR indicates Friday; and SA indicates Saturday. Each BYDAY value can also be preceded by a positive (+n) or negative (-n) integer. If present, this indicates the nth occurrence of a specific day within the MONTHLY or YEARLY "RRULE".
For example, within a MONTHLY rule, +1MO (or simply 1MO) represents the first Monday within the month, whereas -1MO represents the last Monday of the month. The numeric value in a BYDAY rule part with the FREQ rule part set to YEARLY corresponds to an offset within the month when the BYMONTH rule part is present, and corresponds to an offset within the year when the BYWEEKNO or BYMONTH rule parts are present. If an integer modifier is not present, it means all days of this type within the specified frequency. For example, within a MONTHLY rule, MO represents all Mondays within the month. The BYDAY rule part MUST NOT be specified with a numeric value when the FREQ rule part is not set to MONTHLY or YEARLY. Furthermore, the BYDAY rule part MUST NOT be specified with a numeric value with the FREQ rule part set to YEARLY when the BYWEEKNO rule part is specified.

The BYMONTHDAY rule part specifies a COMMA-separated list of days of the month. Valid values are 1 to 31 or -31 to -1. For example, -10 represents the tenth to the last day of the month. The BYMONTHDAY rule part MUST NOT be specified when the FREQ rule part is set to WEEKLY.

The BYYEARDAY rule part specifies a COMMA-separated list of days of the year. Valid values are 1 to 366 or -366 to -1. For example, -1 represents the last day of the year (December 31st) and -306 represents the 306th to the last day of the year (March 1st). The BYYEARDAY rule part MUST NOT be specified when the FREQ rule part is set to

DAILY, WEEKLY, or MONTHLY.

The BYWEEKNO rule part specifies a COMMA-separated list of ordinals specifying weeks of the year. Valid values are 1 to 53 or -53 to -1. This corresponds to weeks according to week numbering as defined in [ISO.8601.2004]. A week is defined as a seven day period, starting on the day of the week defined to be the week start (see WKST). Week number one of the calendar year is the first week that contains at least four (4) days in that calendar year. This rule part MUST NOT be used when the FREQ rule part is set to anything other than YEARLY. For example, 3 represents the third week of the year.

Note: Assuming a Monday week start, week 53 can only occur when Thursday is January 1 or if it is a leap year and Wednesday is January 1.

The BYMONTH rule part specifies a COMMA-separated list of months of the year. Valid values are 1 to 12.

The WKST rule part specifies the day on which the workweek starts. Valid values are MO, TU, WE, TH, FR, SA, and SU. This is significant when a WEEKLY "RRULE" has an interval greater than 1, and a BYDAY rule part is specified. This is also significant when in a YEARLY "RRULE" when a BYWEEKNO rule part is specified. The default value is MO.

The BYSETPOS rule part specifies a COMMA-separated list of values that corresponds to the nth occurrence within the set of recurrence instances specified by the rule. BYSETPOS operates on a set of recurrence instances in one interval of the recurrence rule. For example, in a WEEKLY rule, the interval would be one week A set of recurrence instances starts at the beginning of the interval defined by the FREQ rule part. Valid values are 1 to 366 or -366 to -1. It MUST only be used in conjunction with another BYxxx rule part. For example "the last work day of the month" could be represented as:

```
FREQ=MONTHLY;BYDAY=MO,TU,WE,TH,FR;BYSETPOS=-1
```

Each BYSETPOS value can include a positive (+n) or negative (-n) integer. If present, this indicates the nth occurrence of the specific occurrence within the set of occurrences specified by the rule.

Recurrence rules may generate recurrence instances with an invalid date (e.g., February 30) or nonexistent local time (e.g., 1:30 AM on a day where the local time is moved forward by an hour at 1:00 AM). Such recurrence instances MUST be ignored and MUST NOT be counted as part of the recurrence set. Information, not contained in the rule, necessary to determine the various recurrence instance start time and dates are derived from the Start Time ("DTSTART") component attribute. For example, "FREQ=YEARLY;BYMONTH=1" doesn't specify a specific day within the month or a time. This information would be the same as what is specified for "DTSTART".

property ICalendar.Root as Component

Retrieves the root component of the content (first component).

Type	Description
Component	A Component object that holds the properties and other components.

The Root property gets the root component of the content. The Content property gets the control's content (properties and components). The Properties property gets the Properties collection of the current component. The Components property returns the collection of the Components of the current component. The Load / LoadFile / LoadFileFromUnicode method loads properties and components from iCalendar format. The StartLoad / EndLoad events notifies that the Content is starting / ending to be loaded. The Name property of the Component returns the name of the current component. The Content.Name property always returns "Exontrol.ICalendar", while the Root.Name property returns the name of the first component being found in the content (VCALENDAR). The Save property gives the iCalendar format of the current content.

For instance, having the following iCalendar format:

```
BEGIN:VCALENDAR
VERSION:2.0
PRODID:-//hacksw/handcal//NONSGML v1.0//EN
BEGIN:VEVENT
UID:19970610T172345Z-AF23B2@example.com
DTSTAMP:19970610T172345Z
DTSTART:19970714T170000Z
DTEND:19970715T040000Z
SUMMARY:Bastille Day Party
END:VEVENT
END:VCALENDAR
```

The Content.Components includes the VCALENDAR object, while the Root property refers to the VCALENDAR object directly.

property ICalendar.RuntimeKey as String

Specifies a runtime key to be used for the component.

Type	Description
String	A String expression that indicates your runtime-key to use the control at runtime.

The RuntimeKey property specifies a runtime key to be used for the component. The value for the RuntimeKey property is provided by Exontrol, by the time you purchased the component. The RuntimeKey value is not the same as your development license key. Without a runtime key there are limitations of using the control. For instance, the RecurAll method gets only 16 occurrences, if using the evaluation version or the RuntimeKey property is not specifying a valid, not expired runtime-key.

method ICalendar.Save ()

Saves the content as iCalendar format.

Type	Description
Return	**Description**
String	A String value that specifies the iCalendar format of the control's Content.

The Save method saves the control's content to a string, as iCalendar format. The SaveFile / SaveFileAsUnicode method saves the control's content to a file as iCalendar format. The Load method loads iCalendar format from giving string. The LoadFile / LoadFileFromUnicode method loads iCalendar format from a file.

The order of the events for Load, LoadFile, LoadFileFromUnicode methods is:

- StartLoad event, notifies that the Load, LoadFile, LoadFileFromUnicode methods begins
- AddComponent(NewComponent) event, notifies that a new Component object is added to the Components collection.
- AddProperty(NewPropery) event, notifies that a new Property object is added to the Properties collection. The Component property of the Property object specifies owner/parent component of the newly added property.
- AddParameter(NewParameter) event, notifies that a new Parameter object is added to the Parameters collection. The Property property of the Parameter object specifies owner/parent property of the newly added parameter.
- EndLoad event, notifies that the Load, LoadFile, LoadFileFromUnicode methods ends.

method ICalendar.SaveFile (FileName as String)

Saves the control's content to a file.

Type	Description
FileName as String	A String expression that specifies the file where the control's Content should be saved, as iCalendar format.

The SaveFile / SaveFileAsUnicode method saves the control's content to a file as iCalendar format. The Save method saves the control's content to a string, as iCalendar format. The Load method loads iCalendar format from giving string. The LoadFile / LoadFileFromUnicode method loads iCalendar format from a file.

The order of the events for Load, LoadFile, LoadFileFromUnicode methods is:

- StartLoad event, notifies that the Load, LoadFile, LoadFileFromUnicode methods begins
- AddComponent(NewComponent) event, notifies that a new Component object is added to the Components collection.
- AddProperty(NewPropery) event, notifies that a new Property object is added to the Properties collection. The Component property of the Property object specifies owner/parent component of the newly added property.
- AddParameter(NewParameter) event, notifies that a new Parameter object is added to the Parameters collection. The Property property of the Parameter object specifies owner/parent property of the newly added parameter.
- EndLoad event, notifies that the Load, LoadFile, LoadFileFromUnicode methods ends.

method ICalendar.SaveFileAsUnicode (FileName as String)

Saves the control's content to file as UNICODE.

Type	Description
FileName as String	A String expression that specifies the file where the control's Content should be saved, as iCalendar format.

The SaveFile / SaveFileAsUnicode method saves the control's content to a file as iCalendar format. The Save method saves the control's content to a string, as iCalendar format. The Load method loads iCalendar format from giving string. The LoadFile / LoadFileFromUnicode method loads iCalendar format from a file.

The order of the events for Load, LoadFile, LoadFileFromUnicode methods is:

- StartLoad event, notifies that the Load, LoadFile, LoadFileFromUnicode methods begins
- AddComponent(NewComponent) event, notifies that a new Component object is added to the Components collection.
- AddProperty(NewPropery) event, notifies that a new Property object is added to the Properties collection. The Component property of the Property object specifies owner/parent component of the newly added property.
- AddParameter(NewParameter) event, notifies that a new Parameter object is added to the Parameters collection. The Property property of the Parameter object specifies owner/parent property of the newly added parameter.
- EndLoad event, notifies that the Load, LoadFile, LoadFileFromUnicode methods ends.

property ICalendar.Template as String

Specifies the control's template.

Type	Description
String	A string expression that indicates the control's template.

The control's template uses the X-Script language to initialize the control's content. Use the Template property page of the control to update the control's Template property. Use the Template property to execute code by passing instructions as a string (template string). Use the ExecuteTemplate property to execute a template script and gets the result.

Most of our UI components provide a Template page that's accessible in design mode. No matter what programming language you are using, you can have a quick view of the component's features using the WYSWYG Template editor.

- Place the control to your form or dialog.
- Locate the Properties item, in the control's context menu, in design mode. If your environment doesn't provide a Properties item in the control's context menu, please try to locate in the Properties browser.
- Click it, and locate the Template page.
- Click the Help button. In the left side, you will see the component, in the right side, you will see a x-script code that calls methods and properties of the control.

The control's Template page helps user to initialize the control's look and feel in design mode, using the x-script language that's easy and powerful. The Template page displays the control on the left side of the page. On the right side of the Template page, a simple editor is displayed where user writes the initialization code. The control's look and feel is automatically updated as soon as the user types new instructions. The Template script is saved to the container persistence (when Apply button is pressed), and it is executed when the control is initialized at runtime. Any component that provides a WYSWYG Template page, provides a Template property. The Template property executes code from a string (template string).

The Template or x-script is composed by lines of instructions. Instructions are separated by "\n\r" (newline characters) or ";" character. The ; character may be available only for newer versions of the components.

An x-script instruction/line can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas. (Sample: Dim h, h1, h2)*
- variable **=** property(list of arguments) *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name*

of the object in the context. The "list or arguments" may include variables or values separated by commas. (Sample: h = InsertItem(0,"New Child"))

- property**(** list of arguments **)** = value *Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- method**(** list of arguments **)** *Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- **{** *Beginning the object's context. The properties or methods called between { and } are related to the last object returned by the property prior to { declaration.*
- **}** *Ending the object's context*
- object**.** property(list of arguments)**.**property(list of arguments).... *The .(dot) character splits the object from its property. For instance, the Columns.Add("Column1").HeaderBackColor = RGB(255,0,0), adds a new column and changes the column's header back color.*

The x-script may uses constant expressions as follow:

- *boolean* expression with possible values as *True* or *False*
- *numeric* expression may starts with 0x which indicates a hexa decimal representation, else it should starts with digit, or +/- followed by a digit, and . is the decimal separator. *Sample: 13 indicates the integer 13, or 12.45 indicates the double expression 12,45*
- *date* expression is delimited by **#** character in the format #mm/dd/yyyy hh:mm:ss#. *Sample: #31/12/1971# indicates the December 31, 1971*
- *string* expression is delimited by **"** or **`** characters. If using the ` character, please make sure that it is different than ' which allows adding comments inline. *Sample: "text" indicates the string text.*

Also , the template or x-script code may support general functions as follows:

- **Me** *property indicates the original object.*
- **RGB(**R,G,B**)** *property retrieves an RGB value, where the R, G, B are byte values that indicates the R G B values for the color being specified. For instance, the following code changes the control's background color to red: BackColor = RGB(255,0,0)*
- **LoadPicture(**file**)** *property loads a picture from a file or from BASE64 encoded strings, and returns a Picture object required by the picture properties.*
- **CreateObject(**progID**)** *property creates and retrieves a single uninitialized object of the class associated with a specified program identifier.*

property ICalendar.TemplateDef as Variant

Defines inside variables for the next Template/ExecuteTemplate call.

Type	Description
Variant	A string expression that indicates the Dim declaration, or any Object expression to be assigned to previously declared variables.

The TemplateDef property has been added to allow programming languages such as dBASE Plus to set control's properties with multiple parameters. It is known that programming languages such as **dBASE Plus or XBasic from AlphaFive**, does not support setting a property with multiple parameters. In other words, these programming languages does not support something like *Property(Parameters)* = *Value*, so our controls provide an alternative using the TemplateDef method. The first call of the TemplateDef should be a declaration such as "Dim a,b" which means the next 2 calls of the TemplateDef defines the variables a and b. The next call should be Template or ExecuteTemplate property which can use the variable a and b being defined previously.

So, calling the TemplateDef property should be as follows:

```
with (Control)
   TemplateDef = [Dim var_Column]
   TemplateDef = var_Column
   Template = [var_Column.Def(4) = 255]
endwith
```

This sample allocates a variable var_Column, assigns the value to the variable (the second call of the TemplateDef), and the Template call uses the var_Column variable (as an object), to call its Def property with the parameter 4.

Let's say we need to define the background color for a specified column, so we need to call the Def(exCellBackColor) property of the column, to define the color for all cells in the column.

The following **VB6** sample shows setting the Def property such as:

```
With Control
   .Columns.Add("Column 1").Def(exCellBackColor) = 255
   .Columns.Add "Column 2"
   .Items.AddItem 0
   .Items.AddItem 1
```

```
    .Items.AddItem 2
End With
```

In **dBASE Plus**, calling the Def(4) has no effect, instead using the TemplateDef helps you to use properly the Def property as follows:

```
local Control,var_Column

Control = form.Activex1.nativeObject
// Control.Columns.Add("Column 1").Def(4) = 255
var_Column = Control.Columns.Add("Column 1")
with (Control)
    TemplateDef = [Dim var_Column]
    TemplateDef = var_Column
    Template = [var_Column.Def(4) = 255]
endwith
Control.Columns.Add("Column 2")
Control.Items.AddItem(0)
Control.Items.AddItem(1)
Control.Items.AddItem(2)
```

The equivalent sample for **XBasic in A5**, is as follows:

```
Dim Control as P
Dim var_Column as P

Control = topparent:CONTROL_ACTIVEX1.activex
' Control.Columns.Add("Column 1").Def(4) = 255
var_Column = Control.Columns.Add("Column 1")
Control.TemplateDef = "Dim var_Column"
Control.TemplateDef = var_Column
Control.Template = "var_Column.Def(4) = 255"

Control.Columns.Add("Column 2")
Control.Items.AddItem(0)
Control.Items.AddItem(1)
Control.Items.AddItem(2)
```

The samples just call the Column.Def(4) = Value, using the TemplateDef. The first call of TemplateDef property is "Dim var_Column", which indicates that the next call of the TemplateDef will defines the value of the variable var_Column, in other words, it defines the object var_Column. The last call of the Template property uses the var_Column member to use the x-script and so to set the Def property so a new color is being assigned to the column.

The TemplateDef, Template and ExecuteTemplate support x-script language (Template script of the Exontrols), like explained bellow:

The Template or x-script is composed by lines of instructions. Instructions are separated by "\n\r" (newline characters) or ";" character. The ; character may be available only for newer versions of the components.

An x-script instruction/line can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas. (Sample: Dim h, h1, h2)*
- variable **=** property(list of arguments) *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name of the object in the context. The "list or arguments" may include variables or values separated by commas. (Sample: h = InsertItem(0,"New Child"))*
- property**(** list of arguments **)** = value *Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- method**(** list of arguments **)** *Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- **{** *Beginning the object's context. The properties or methods called between { and } are related to the last object returned by the property prior to { declaration.*
- **}** *Ending the object's context*
- object**.** property(list of arguments)**.**property(list of arguments).... *The .(dot) character splits the object from its property. For instance, the Columns.Add("Column1").HeaderBackColor = RGB(255,0,0), adds a new column and changes the column's header back color.*

The x-script may uses constant expressions as follow:

- *boolean* expression with possible values as *True* or *False*
- *numeric* expression may starts with 0x which indicates a hexa decimal representation, else it should starts with digit, or +/- followed by a digit, and . is the decimal separator. *Sample: 13 indicates the integer 13, or 12.45 indicates the double expression 12,45*
- *date* expression is delimited by **#** character in the format #mm/dd/yyyy hh:mm:ss#. *Sample: #31/12/1971# indicates the December 31, 1971*
- *string* expression is delimited by **"** or **`** characters. If using the ` character, please

make sure that it is different than ' which allows adding comments inline. *Sample: "text" indicates the string text.*

Also , the template or x-script code may support general functions as follows:

- **Me** *property indicates the original object.*
- **RGB(**R,G,B**)** *property retrieves an RGB value, where the R, G, B are byte values that indicates the R G B values for the color being specified. For instance, the following code changes the control's background color to red: BackColor = RGB(255,0,0)*
- **LoadPicture(**file**)** *property loads a picture from a file or from BASE64 encoded strings, and returns a Picture object required by the picture properties.*
- **CreateObject(**progID**)** *property creates and retrieves a single uninitialized object of the class associated with a specified program identifier.*

method ICalendar.TemplatePut (newVal as Variant)

Defines inside variables for the next Template/ExecuteTemplate call.

Type	Description
newVal as Variant	A string expression that indicates the Dim declaration, or any Object expression to be assigned to previously declared variables.

The TemplatePut method / TemplateDef property has been added to allow programming languages such as dBASE Plus to set control's properties with multiple parameters. It is known that programming languages such as **dBASE Plus or XBasic from AlphaFive**, does not support setting a property with multiple parameters. In other words, these programming languages does not support something like *Property(Parameters) = Value*, so our controls provide an alternative using the TemplateDef / TemplatePut method. The first call of the TemplateDef should be a declaration such as "Dim a,b" which means the next 2 calls of the TemplateDef defines the variables a and b. The next call should be Template or ExecuteTemplate property which can use the variable a and b being defined previously.

The TemplateDef, TemplatePut, Template and ExecuteTemplate support x-script language (Template script of the Exontrols), like explained bellow:

The Template or x-script is composed by lines of instructions. Instructions are separated by "\n\r" (newline characters) or ";" character. The ; character may be available only for newer versions of the components.

An x-script instruction/line can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas. (Sample: Dim h, h1, h2)*
- variable **=** property(list of arguments) *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name of the object in the context. The "list or arguments" may include variables or values separated by commas. (Sample: h = InsertItem(0,"New Child"))*
- property**(** list of arguments **)** = value *Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- method**(** list of arguments **)** *Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- **{** *Beginning the object's context. The properties or methods called between { and } are related to the last object returned by the property prior to { declaration.*
- **}** *Ending the object's context*
- object**.** property(list of arguments)**.**property(list of arguments).... *The .(dot) character splits the object from its property. For instance, the*

Columns.Add("Column1").HeaderBackColor = RGB(255,0,0), adds a new column and changes the column's header back color.

The x-script may uses constant expressions as follow:

- *boolean* expression with possible values as *True* or *False*
- *numeric* expression may starts with 0x which indicates a hexa decimal representation, else it should starts with digit, or +/- followed by a digit, and . is the decimal separator. *Sample: 13 indicates the integer 13, or 12.45 indicates the double expression 12,45*
- *date* expression is delimited by **#** character in the format #mm/dd/yyyy hh:mm:ss#. *Sample: #31/12/1971# indicates the December 31, 1971*
- *string* expression is delimited by **"** or **`** characters. If using the ` character, please make sure that it is different than ' which allows adding comments inline. *Sample: "text" indicates the string text.*

Also , the template or x-script code may support general functions as follows:

- **Me** *property indicates the original object.*
- **RGB(**R,G,B**)** *property retrieves an RGB value, where the R, G, B are byte values that indicates the R G B values for the color being specified. For instance, the following code changes the control's background color to red: BackColor = RGB(255,0,0)*
- **LoadPicture(**file**)** *property loads a picture from a file or from BASE64 encoded strings, and returns a Picture object required by the picture properties.*
- **CreateObject(**progID**)** *property creates and retrieves a single uninitialized object of the class associated with a specified program identifier.*

property ICalendar.toICalendar (Value as Variant, Type as PropertyTypeEnum) as String

Converts the giving VARIANT expression to ICalendar format. For instance, toICalendar(CSng(15.5), exPropertyTypeDuration) returns "P15DT12H".

Type	Description
Value as Variant	A VARIANT expression to be converted to specified type
Type as PropertyTypeEnum	A PropertyTypeEnum expression that specifies the type of value to be converted to.
String	A String expression that specifies the iCalendar format of the value / type

The toICalendar property converts the giving value to a specified type. The valuesFromICalendar property extracts all values or specified value of the giving value/type in ICalendar format. The GuessType property guesses the property's type, from its value. The Value property specifies the value of the property. The Name property specifies the name of the property. The Parameters property specifies the property's Parameters collection.

The following samples converts a string value to a exPropertyTypeBinary value:

VBA (MS Access, Excell...)

```
Set ICalendar1 = CreateObject("Exontrol.ICalendar.1")
With ICalendar1
   With .Content.Components.Add("VCALENDAR")
      .Properties.Add "Binary1",ICalendar1.toICalendar("This is a bit of text converted to binary",1)
      With .Properties.Add("Binary2")
         .Value = "This is a bit of text converted to binary"
         .Type = 1
      End With
   End With
   Debug.Print( .Save )
End With
```

VB6

```
Set ICalendar1 = CreateObject("Exontrol.ICalendar.1")
```

```
With ICalendar1
    With .Content.Components.Add("VCALENDAR")
        .Properties.Add "Binary1",ICalendar1.toICalendar("This is a bit of text converted
to binary",exPropertyTypeBinary)
        With .Properties.Add("Binary2")
            .Value = "This is a bit of text converted to binary"
            .Type = exPropertyTypeBinary
        End With
    End With
    Debug.Print( .Save )
End With
```

VB.NET

```
' Add 'exontrol.exicalendar.dll(ExICalendar.dll)' reference to your project.
Exicalendar1 = New exontrol.EXICALENDARLib.exicalendar()
With Exicalendar1
    With .Content.Components.Add("VCALENDAR")
        .Properties.Add("Binary1",Exicalendar1.get_toICalendar("This is a bit of text
converted to
binary",exontrol.EXICALENDARLib.PropertyTypeEnum.exPropertyTypeBinary))
        With .Properties.Add("Binary2")
            .Value = "This is a bit of text converted to binary"
            .Type = exontrol.EXICALENDARLib.PropertyTypeEnum.exPropertyTypeBinary
        End With
    End With
    Debug.Print( .Save() )
End With
```

VB.NET for /COM

```
AxICalendar1 = CreateObject("Exontrol.ICalendar.1")
With AxICalendar1
    With .Content.Components.Add("VCALENDAR")
        .Properties.Add("Binary1",AxICalendar1.toICalendar("This is a bit of text
converted to binary",EXICALENDARLib.PropertyTypeEnum.exPropertyTypeBinary))
        With .Properties.Add("Binary2")
            .Value = "This is a bit of text converted to binary"
        .Type = EXICALENDARLib.PropertyTypeEnum.exPropertyTypeBinary
      End With
    End With
    Debug.Print( .Save() )
End With
```

C++

```
/*
    Includes the definition for CreateObject function like follows:

    #include <comdef.h>
    IUnknownPtr CreateObject( BSTR Object )
    {
        IUnknownPtr spResult;
        spResult.CreateInstance( Object );
        return spResult;
    };

*/
/*
    Copy and paste the following directives to your header file as
    it defines the namespace 'EXICALENDARLib' for the library: 'ICalendar 1.0 Type
Library'

    #import <ExICalendar.dll>
    using namespace EXICALENDARLib;
*/
EXICALENDARLib::IICalendarPtr spICalendar1 =
::CreateObject(L"Exontrol.ICalendar.1");
EXICALENDARLib::IComponentPtr var_Component = spICalendar1->GetContent()-
>GetComponents()->Add(L"VCALENDAR");
    var_Component->GetProperties()->Add(L"Binary1",spICalendar1-
>GettoICalendar("This is a bit of text converted to
binary",EXICALENDARLib::exPropertyTypeBinary));
    EXICALENDARLib::IPropertyPtr var_Property = var_Component->GetProperties()-
>Add(L"Binary2",vtMissing);
```

```
        var_Property->PutValue("This is a bit of text converted to binary");
        var_Property->PutType(EXICALENDARLib::exPropertyTypeBinary);
OutputDebugStringW( spICalendar1->Save() );
```

C++ Builder

```
Exicalendarlib_tlb::IICalendarPtr ICalendar1 =
Variant::CreateObject(L"Exontrol.ICalendar.1");
Exicalendarlib_tlb::IComponentPtr var_Component = ICalendar1->Content-
>Components->Add(L"VCALENDAR");
    var_Component->Properties->Add(L"Binary1",TVariant(ICalendar1-
>get_toICalendar(TVariant("This is a bit of text converted to
binary"),Exicalendarlib_tlb::PropertyTypeEnum::exPropertyTypeBinary)));
    Exicalendarlib_tlb::IPropertyPtr var_Property = var_Component->Properties-
>Add(L"Binary2",TNoParam());
        var_Property->set_Value(TVariant("This is a bit of text converted to binary"));
        var_Property->Type =
Exicalendarlib_tlb::PropertyTypeEnum::exPropertyTypeBinary;
OutputDebugString( ICalendar1->Save() );
```

C#

```
// Add 'exontrol.exicalendar.dll(ExICalendar.dll)' reference to your project.
exontrol.EXICALENDARLib.exicalendar exicalendar1 = new
exontrol.EXICALENDARLib.exicalendar();
exontrol.EXICALENDARLib.Component var_Component =
exicalendar1.Content.Components.Add("VCALENDAR");
    var_Component.Properties.Add("Binary1",exicalendar1.get_toICalendar("This is a
bit of text converted to
binary",exontrol.EXICALENDARLib.PropertyTypeEnum.exPropertyTypeBinary));
    exontrol.EXICALENDARLib.Property var_Property =
var_Component.Properties.Add("Binary2",null);
        var_Property.Value = "This is a bit of text converted to binary";
        var_Property.Type =
exontrol.EXICALENDARLib.PropertyTypeEnum.exPropertyTypeBinary;
System.Diagnostics.Debug.Print( exicalendar1.Save() );
```

JScript/JavaScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:D6C87100-38E2-4ABB-8AC2-4C0097AEE2D6"
id="ICalendar1"></OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
	var var_Component = ICalendar1.Content.Components.Add("VCALENDAR");
		var_Component.Properties.Add("Binary1",ICalendar1.toICalendar("This is a bit
of text converted to binary",1));
		var var_Property = var_Component.Properties.Add("Binary2",null);
			var_Property.Value = "This is a bit of text converted to binary";
			var_Property.Type = 1;
	alert( ICalendar1.Save() );
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:D6C87100-38E2-4ABB-8AC2-4C0097AEE2D6"
id="ICalendar1"></OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
	With ICalendar1
		With .Content.Components.Add("VCALENDAR")
			.Properties.Add "Binary1",ICalendar1.toICalendar("This is a bit of text
converted to binary",1)
			With .Properties.Add("Binary2")
				.Value = "This is a bit of text converted to binary"
				.Type = 1
```

```
        End With
      End With
      alert( .Save )
   End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
EXICALENDARLib.ICalendar axICalendar1 = new EXICALENDARLib.ICalendar();
EXICALENDARLib.Component var_Component =
axICalendar1.Content.Components.Add("VCALENDAR");
   var_Component.Properties.Add("Binary1",axICalendar1.get_toICalendar("This is a
bit of text converted to
binary",EXICALENDARLib.PropertyTypeEnum.exPropertyTypeBinary));
   EXICALENDARLib.Property var_Property =
var_Component.Properties.Add("Binary2",null);
      var_Property.Value = "This is a bit of text converted to binary";
      var_Property.Type = EXICALENDARLib.PropertyTypeEnum.exPropertyTypeBinary;
System.Diagnostics.Debug.Print( axICalendar1.Save() );
```

X++ (Dynamics Ax 2009)

```
public void init()
{
   COM com_Component,com_Properties,com_Property,com_exicalendar1;
   anytype exicalendar1,var_Component,var_Properties,var_Property;
   ;

   super();

   // Add 'exicalendar.dll(ExICalendar.dll)' reference to your project.
   exicalendar1 = COM::createFromObject(new EXICALENDARLib.exicalendar());
com_exicalendar1 = exicalendar1;
   var_Component =
COM::createFromObject(com_exicalendar1.Content().Components()).Add("VCALENDAR
 com_Component = var_Component;
       var_Properties = COM::createFromObject(com_Component.Properties());
com_Properties = var_Properties;

com_Properties.Add("Binary1",COMVariant::createFromStr(com_exicalendar1.toICalend
 is a bit of text converted to binary",1/*exPropertyTypeBinary*/)));
       var_Properties = COM::createFromObject(com_Component.Properties());
com_Properties = var_Properties;
       var_Property = COM::createFromObject(com_Properties).Add("Binary2");
com_Property = var_Property;
          com_Property.Value("This is a bit of text converted to binary");
          com_Property.Type(1/*exPropertyTypeBinary*/);
   print( com_exicalendar1.Save() );
}
```

Delphi 8 (.NET only)

```
AxICalendar1 :=
(ComObj.CreateComObject(ComObj.ProgIDToClassID('Exontrol.ICalendar.1')) as
EXICALENDARLib.ICalendar);
with AxICalendar1 do
begin
   with Content.Components.Add('VCALENDAR') do
   begin
      Properties.Add('Binary1',TObject(AxICalendar1.toICalendar['This is a bit of text
converted to binary',EXICALENDARLib.PropertyTypeEnum.exPropertyTypeBinary]));
      with Properties.Add('Binary2',Nil) do
      begin
         Value := 'This is a bit of text converted to binary';
         Type := EXICALENDARLib.PropertyTypeEnum.exPropertyTypeBinary;
      end;
   end;
   OutputDebugString( Save() );
end
```

Delphi (standard)

```
ICalendar1 :=
(IUnknown(ComObj.CreateComObject(ComObj.ProgIDToClassID('Exontrol.ICalendar.1'))
 as EXICALENDARLib_TLB.ICalendar);
with ICalendar1 do
begin
   with Content.Components.Add('VCALENDAR') do
   begin
      Properties.Add('Binary1',OleVariant(ICalendar1.toICalendar['This is a bit of text
converted to binary',EXICALENDARLib_TLB.exPropertyTypeBinary]));
      with Properties.Add('Binary2',Null) do
      begin
         Value := 'This is a bit of text converted to binary';
         Type := EXICALENDARLib_TLB.exPropertyTypeBinary;
      end;
   end;
   OutputDebugString( Save() );
end
```

VFP

```
thisform.ICalendar1 = CreateObject("Exontrol.ICalendar.1")
with thisform.ICalendar1
   with .Content.Components.Add("VCALENDAR")
      .Properties.Add("Binary1",thisform.ICalendar1.toICalendar("This is a bit of text
converted to binary",1))
      with .Properties.Add("Binary2")
         .Value = "This is a bit of text converted to binary"
         .Type = 1
      endwith
   endwith
   DEBUGOUT( .Save )
endwith
```

dBASE Plus

```
local oICalendar,var_Component,var_Property

oICalendar = new OleAutoClient("Exontrol.ICalendar.1")
```

```
var_Component = oICalendar.Content.Components.Add("VCALENDAR")
   var_Component.Properties.Add("Binary1",oICalendar.toICalendar("This is a bit of
text converted to binary",1))
   var_Property = var_Component.Properties.Add("Binary2")
      var_Property.Value = "This is a bit of text converted to binary"
      var_Property.Type = 1
? oICalendar.Save()
```

XBasic (Alpha Five)

```
Dim oICalendar as P
Dim var_Component as P
Dim var_Property as P

oICalendar = OLE.Create("Exontrol.ICalendar.1")

var_Component = oICalendar.Content.Components.Add("VCALENDAR")
   var_Component.Properties.Add("Binary1",oICalendar.toICalendar("This is a bit of
text converted to binary",1))
   var_Property = var_Component.Properties.Add("Binary2")
      var_Property.Value = "This is a bit of text converted to binary"
      var_Property.Type = 1
? oICalendar.Save()
```

Visual Objects

```
local var_Component as IComponent
local var_Property as IProperty

oDCOCX_Exontrol1 := IICalendar{"Exontrol.ICalendar.1"}
var_Component := oDCOCX_Exontrol1:Content:Components:Add("VCALENDAR")
   var_Component:Properties:Add("Binary1",oDCOCX_Exontrol1:[toICalendar,"This is a
bit of text converted to binary",exPropertyTypeBinary])
   var_Property := var_Component:Properties:Add("Binary2",nil)
      var_Property:Value := "This is a bit of text converted to binary"
```

```
		var_Property:Type := exPropertyTypeBinary
OutputDebugString(String2Psz( oDCOCX_Exontrol1:Save() ))
```

PowerBuilder

```
OleObject oICalendar,var_Component,var_Property

oICalendar = CREATE OLEObject
oICalendar.ConnectToNewObject("Exontrol.ICalendar.1")

var_Component = oICalendar.Content.Components.Add("VCALENDAR")
	var_Component.Properties.Add("Binary1",oICalendar.toICalendar("This is a bit of
text converted to binary",1))
	var_Property = var_Component.Properties.Add("Binary2")
		var_Property.Value = "This is a bit of text converted to binary"
		var_Property.Type = 1
MessageBox("Information",string( oICalendar.Save() ))
```

Visual DataFlex

```
Procedure OnCreate
	Forward Send OnCreate
	Variant oComICalendar1
	Get ComCreateObject "Exontrol.ICalendar.1" to oComICalendar1

	Variant voComponent
	Get ComContent to voComponent
	Handle hoComponent
	Get Create (RefClass(cComComponent)) to hoComponent
	Set pvComObject of hoComponent to voComponent
		Variant voComponents
		Get ComComponents of hoComponent to voComponents
		Handle hoComponents
		Get Create (RefClass(cComComponents)) to hoComponents
		Set pvComObject of hoComponents to voComponents
			Variant voComponent1
```

```
        Get ComAdd of hoComponents "VCALENDAR" to voComponent1
        Handle hoComponent1
        Get Create (RefClass(cComComponent)) to hoComponent1
        Set pvComObject of hoComponent1 to voComponent1
          Variant voProperties
          Get ComProperties of hoComponent1 to voProperties
          Handle hoProperties
          Get Create (RefClass(cComProperties)) to hoProperties
          Set pvComObject of hoProperties to voProperties
            Variant vValue
              Get ComtoICalendar "This is a bit of text converted to binary"
OLEexPropertyTypeBinary to vValue
            Get ComAdd of hoProperties "Binary1" vValue to Nothing
          Send Destroy to hoProperties
          Variant voProperties1
          Get ComProperties of hoComponent1 to voProperties1
          Handle hoProperties1
          Get Create (RefClass(cComProperties)) to hoProperties1
          Set pvComObject of hoProperties1 to voProperties1
            Variant voProperty
            Get ComAdd of hoProperties1 "Binary2" Nothing to voProperty
            Handle hoProperty
            Get Create (RefClass(cComProperty)) to hoProperty
            Set pvComObject of hoProperty to voProperty
              Set ComValue of hoProperty to "This is a bit of text converted to
binary"
              Set ComType of hoProperty to OLEexPropertyTypeBinary
            Send Destroy to hoProperty
          Send Destroy to hoProperties1
        Send Destroy to hoComponent1
      Send Destroy to hoComponents
    Send Destroy to hoComponent
    Showln (ComSave(Self))
End_Procedure
```

XBase++

```
#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
   LOCAL oForm
   LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
   LOCAL oICalendar
   LOCAL oComponent
   LOCAL oProperty

   oForm := XbpDialog():new( AppDesktop() )
   oForm:drawingArea:clipChildren := .T.
   oForm:create( ,,{100,100}, {640,480},, .F. )
   oForm:close  := {|| PostAppEvent( xbeP_Quit )}

   oICalendar := XbpActiveXControl():new( oForm:drawingArea )
   oICalendar:CLSID  := "Exontrol.ICalendar.1" /*{D6C87100-38E2-4ABB-8AC2-
4C0097AEE2D6}*/
   oICalendar:create(,, {10,60},{610,370} )

      oComponent := oICalendar:Content():Components():Add("VCALENDAR")
         oComponent:Properties():Add("Binary1",oICalendar:toICalendar("This is a bit
of text converted to binary",1/*exPropertyTypeBinary*/))
         oProperty := oComponent:Properties():Add("Binary2")
            oProperty:Value := "This is a bit of text converted to binary"
            oProperty:Type := 1/*exPropertyTypeBinary*/
      DevOut( oICalendar:Save() )

   oForm:Show()
   DO WHILE nEvent != xbeP_Quit
      nEvent := AppEvent( @mp1, @mp2, @oXbp )
      oXbp:handleEvent( nEvent, mp1, mp2 )
   ENDDO
RETURN
```

property ICalendar.UserData as Variant

Indicates any extra data associated with the control / root.

Type	Description
Variant	A VARIANT expression that indicates any extra data associated with the control / root.

The UserData property indicates any extra data associated with the control / root. The Content property gets the control's content (properties and components). The Root property gets the root component of the content. The Properties property gets the Properties collection of the current component. The Components property returns the collection of the Components of the current component.

property ICalendar.valuesFromICalendar (Value as String, Type as PropertyTypeEnum, Parameter as String) as Variant

Extracts all values or specified value of the giving value in ICalendar format. For instance valuesFromICalendar("P15DT12H", exPropertyTypeDuration, "Duration") returns 15.5, which indicates the duration in days, hours, minutes, seconds as a DATE expression.

Type	Description
Value as String	A String expression to retrieve values from
Type as PropertyTypeEnum	A PropertyTypeEnum expression that indicates the type of the Value
Parameter as String	A String expression that specifies the Parameter whose value is to be requested. If Parameter is "" (empty string), the valuesFromICalendar property returns all known parameters with their values.
Variant	A VARIANT expression that specifies the requested value for giving parameter.

The valuesFromICalendar property returns values for known parameters of the giving expression (available for exPropertyTypeDuration, exPropertyTypePeriod and exPropertyTypeRecur types as explained bellow). The fromICalendar property converts the ICalendar value to a VARIANT expression. The toICalendar property is the reverse function, so it converts a VARIANT expression back to iCalendar format.

The valuesFromICalendar property is available for following types:

- exPropertyTypeDuration, The valuesFromICalendar property returns values for known parameters of the giving expression. The exPropertyTypeDuration expression supports the following parameters:
  - "-", specifies whether the duration is negative or positive
  - "W", indicates the number of weeks
  - "D", indicates the number of days
  - "H", indicates the number of hours
  - "M", indicates the number of minutes
  - "S", indicates the number of seconds
  - "Duration", indicates the duration in days for integer part, while the fractional part specifies hours, minutes and seconds.

  For instance, the valuesFromICalendar("P12DT4H",exPropertyTypeDuration,"D") returns the number of days in the duration expression (in this case 12), while the valuesFromICalendar("P12DT4H",exPropertyTypeDuration,"H") returns the number of hours in the duration expression (in this case 4) .

- exPropertyTypePeriod, returns a string value (VT_BSTR). The valuesFromICalendar property returns values for known parameters of the giving expression. The exPropertyTypePeriod expression supports the following parameters:
  - "Start", indicates the date to start the period
  - "End", indicates the date to end the period
  - "-", specifies whether the duration of the period is negative or positive
  - "W", indicates the number of weeks within period
  - "D", indicates the number of days within period
  - "H", indicates the number of hours within period
  - "M", indicates the number of minutes within period
  - "S", indicates the number of seconds within period
  - "Duration", indicates the duration of the period in days for integer part, while the fractional part specifies hours, minutes and seconds.

  For instance, the valuesFromICalendar("20010101T000000/P1D",exPropertyTypePeriod,"Start") returns the date to start the period (in this case #1/1/2001#), while the valuesFromICalendar("20010101T000000/P1D",exPropertyTypePeriod,"D") returns the number of days for the period expression (in this case 1).

- exPropertyTypeRecur, returns a string value (VT_BSTR). The valuesFromICalendar property returns values for known parameters of the giving expression. The exPropertyTypeRecur expression supports the following parameters:
  - "FREQ", identifies the type of recurrence rule, and could by any of the following: "SECONDLY", "MINUTELY", "HOURLY", "DAILY", "WEEKLY", "MONTHLY", "YEARLY"
  - "UNTIL", defines a DATE value that bounds the recurrence rule in an inclusive manner.
  - "COUNT", defines the number of occurrences at which to range-bound the recurrence.
  - "INTERVAL", contains a positive integer representing at which intervals the recurrence rule repeats.
  - "BYSECOND", specifies a COMMA-separated list of seconds within a minute.
  - "BYMINUTE", specifies a COMMA-separated list of minutes within an hour
  - "BYHOUR", specifies a COMMA-separated list of hours of the day
  - "BYDAY", specifies a COMMA-separated list of days of the week; SU indicates Sunday; MO indicates Monday; TU indicates Tuesday; WE indicates Wednesday; TH indicates Thursday; FR indicates Friday; and SA indicates Saturday. Each BYDAY value can also be preceded by a positive (+n) or negative (-n) integer. If present, this indicates the nth occurrence of a specific day within the MONTHLY or YEARLY "RRULE".
  - "BYMONTHDAY", specifies a COMMA-separated list of days of the month. Valid values are 1 to 31 or -31 to -1.

- "BYYEARDAY", specifies a COMMA-separated list of days of the year. Valid values are 1 to 366 or -366 to -1.
- "BYWEEKNO", specifies a COMMA-separated list of ordinals specifying weeks of the year. Valid values are 1 to 53 or -53 to -1.
- "BYMONTH", specifies a COMMA-separated list of months of the year. Valid values are 1 to 12.
- "BYSETPOS", specifies a COMMA-separated list of values that corresponds to the nth occurrence within the set of recurrence instances specified by the rule.
- "WKST", specifies the day on which the workweek starts. Valid values are MO, TU, WE, TH, FR, SA, and SU. The default value is MO.

For instance, the valuesFromICalendar("FREQ=WEEKLY;INTERVAL=2;COUNT=4;BYDAY=SA,SU",exPr returns all known properties of the recurrence rule like "BYDAY=SA,SU;COUNT=4;FREQ=WEEKLY;INTERVAL=2", while the valuesFromICalendar("FREQ=WEEKLY;INTERVAL=2;COUNT=4;BYDAY=SA,SU",exPr returns the value of the FREQ parameter (in this case WEEKLY).

The following samples show how can you can find the duration in weeks, days, hours, minutes, seconds from a property of duration type.

Having the "P3DT7H48M" iCalendar format for duration it includes:

all: -=0;D=3;Duration=3.325;H=7;M=48;S=0;W=0
duration: 3.325
weeks: 0
days: 3
hour: 7
min: 48
sec: 0

VBA (MS Access, Excell...)

```
Set ICalendar1 = CreateObject("Exontrol.ICalendar.1")
With ICalendar1
   With .Content.Components.Add("VCALENDAR")
      .Properties.Add "Duration",ICalendar1.toICalendar(3.325,6)
   End With
   Set p = .Root.Properties.Item("Duration")
   i = .toICalendar(p.Value,p.GuessType) ' p.GuessType
   Debug.Print( "icalendar:" )
   Debug.Print( i )
```

```
    Debug.Print( "all:" )
    Debug.Print( .valuesFromICalendar(i,p.GuessType,"") ) ' p.GuessType
    Debug.Print( "duration:" )
    Debug.Print( .valuesFromICalendar(i,p.GuessType,"Duration") ) ' p.GuessType
    Debug.Print( "weeks:" )
    Debug.Print( .valuesFromICalendar(i,p.GuessType,"W") ) ' p.GuessType
    Debug.Print( "days:" )
    Debug.Print( .valuesFromICalendar(i,p.GuessType,"D") ) ' p.GuessType
    Debug.Print( "hour:" )
    Debug.Print( .valuesFromICalendar(i,p.GuessType,"H") ) ' p.GuessType
    Debug.Print( "min:" )
    Debug.Print( .valuesFromICalendar(i,p.GuessType,"M") ) ' p.GuessType
    Debug.Print( "sec:" )
    Debug.Print( .valuesFromICalendar(i,p.GuessType,"S") ) ' p.GuessType
End With
```

VB6

```
Set ICalendar1 = CreateObject("Exontrol.ICalendar.1")
With ICalendar1
    With .Content.Components.Add("VCALENDAR")
        .Properties.Add
"Duration",ICalendar1.toICalendar(3.325,exPropertyTypeDuration)
    End With
    Set p = .Root.Properties.Item("Duration")
    i = .toICalendar(p.Value,p.GuessType)
    Debug.Print( "icalendar:" )
    Debug.Print( i )
    Debug.Print( "all:" )
    Debug.Print( .valuesFromICalendar(i,p.GuessType,"") )
    Debug.Print( "duration:" )
    Debug.Print( .valuesFromICalendar(i,p.GuessType,"Duration") )
    Debug.Print( "weeks:" )
    Debug.Print( .valuesFromICalendar(i,p.GuessType,"W") )
    Debug.Print( "days:" )
    Debug.Print( .valuesFromICalendar(i,p.GuessType,"D") )
    Debug.Print( "hour:" )
    Debug.Print( .valuesFromICalendar(i,p.GuessType,"H") )
    Debug.Print( "min:" )
    Debug.Print( .valuesFromICalendar(i,p.GuessType,"M") )
    Debug.Print( "sec:" )
    Debug.Print( .valuesFromICalendar(i,p.GuessType,"S") )
End With
```

VB.NET

```
' Add 'exontrol.exicalendar.dll(ExICalendar.dll)' reference to your project.
Exicalendar1 = New exontrol.EXICALENDARLib.exicalendar()
Dim i,p
With Exicalendar1
    With .Content.Components.Add("VCALENDAR")

.Properties.Add("Duration",Exicalendar1.get_toICalendar(3.325,exontrol.EXICALENDARI

    End With
    p = .Root.Properties.Item("Duration")
    i = .get_toICalendar(p.Value,p.GuessType)
    Debug.Print( "icalendar:" )
    Debug.Print( i )
    Debug.Print( "all:" )
    Debug.Print( .get_valuesFromICalendar(i,p.GuessType,"") )
    Debug.Print( "duration:" )
    Debug.Print( .get_valuesFromICalendar(i,p.GuessType,"Duration") )
    Debug.Print( "weeks:" )
    Debug.Print( .get_valuesFromICalendar(i,p.GuessType,"W") )
    Debug.Print( "days:" )
    Debug.Print( .get_valuesFromICalendar(i,p.GuessType,"D") )
    Debug.Print( "hour:" )
    Debug.Print( .get_valuesFromICalendar(i,p.GuessType,"H") )
    Debug.Print( "min:" )
    Debug.Print( .get_valuesFromICalendar(i,p.GuessType,"M") )
    Debug.Print( "sec:" )
    Debug.Print( .get_valuesFromICalendar(i,p.GuessType,"S") )
End With
```

VB.NET for /COM

```
AxICalendar1 = CreateObject("Exontrol.ICalendar.1")
Dim i,p
With AxICalendar1
   With .Content.Components.Add("VCALENDAR")

.Properties.Add("Duration",AxICalendar1.toICalendar(3.325,EXICALENDARLib.Property

   End With
   p = .Root.Properties.Item("Duration")
   i = .get_toICalendar(p.Value,p.GuessType)
   Debug.Print( "icalendar:" )
   Debug.Print( i )
   Debug.Print( "all:" )
   Debug.Print( .get_valuesFromICalendar(i,p.GuessType,"") )
   Debug.Print( "duration:" )
   Debug.Print( .get_valuesFromICalendar(i,p.GuessType,"Duration") )
   Debug.Print( "weeks:" )
   Debug.Print( .get_valuesFromICalendar(i,p.GuessType,"W") )
   Debug.Print( "days:" )
   Debug.Print( .get_valuesFromICalendar(i,p.GuessType,"D") )
   Debug.Print( "hour:" )
   Debug.Print( .get_valuesFromICalendar(i,p.GuessType,"H") )
   Debug.Print( "min:" )
   Debug.Print( .get_valuesFromICalendar(i,p.GuessType,"M") )
   Debug.Print( "sec:" )
   Debug.Print( .get_valuesFromICalendar(i,p.GuessType,"S") )
End With
```

C++

```
/*
   Includes the definition for CreateObject function like follows:

   #include <comdef.h>
   IUnknownPtr CreateObject( BSTR Object )
   {
```

```
		IUnknownPtr spResult;
		spResult.CreateInstance( Object );
		return spResult;
	};

*/
/*
	Copy and paste the following directives to your header file as
	it defines the namespace 'EXICALENDARLib' for the library: 'ICalendar 1.0 Type
Library'

	#import <ExICalendar.dll>
	using namespace EXICALENDARLib;
*/
EXICALENDARLib::IICalendarPtr spICalendar1 =
::CreateObject(L"Exontrol.ICalendar.1");
EXICALENDARLib::IComponentPtr var_Component = spICalendar1->GetContent()-
>GetComponents()->Add(L"VCALENDAR");
	var_Component->GetProperties()->Add(L"Duration",spICalendar1-
>GettoICalendar(double(3.325),EXICALENDARLib::exPropertyTypeDuration));
EXICALENDARLib::IPropertyPtr p = spICalendar1->GetRoot()->GetProperties()-
>GetItem("Duration");
_bstr_t i = spICalendar1->GettoICalendar(p->GetValue(),p->GetGuessType());
OutputDebugStringW( L"icalendar:" );
OutputDebugStringW( L"i" );
OutputDebugStringW( L"all:" );
OutputDebugStringW( _bstr_t(spICalendar1->GetvaluesFromICalendar(L"i",p-
>GetGuessType(),L"")) );
OutputDebugStringW( L"duration:" );
OutputDebugStringW( _bstr_t(spICalendar1->GetvaluesFromICalendar(L"i",p-
>GetGuessType(),L"Duration")) );
OutputDebugStringW( L"weeks:" );
OutputDebugStringW( _bstr_t(spICalendar1->GetvaluesFromICalendar(L"i",p-
>GetGuessType(),L"W")) );
OutputDebugStringW( L"days:" );
OutputDebugStringW( _bstr_t(spICalendar1->GetvaluesFromICalendar(L"i",p-
>GetGuessType(),L"D")) );
```

```
OutputDebugStringW( L"hour:" );
OutputDebugStringW( _bstr_t(spICalendar1->GetvaluesFromICalendar(L"i",p-
>GetGuessType(),L"H")) );
OutputDebugStringW( L"min:" );
OutputDebugStringW( _bstr_t(spICalendar1->GetvaluesFromICalendar(L"i",p-
>GetGuessType(),L"M")) );
OutputDebugStringW( L"sec:" );
OutputDebugStringW( _bstr_t(spICalendar1->GetvaluesFromICalendar(L"i",p-
>GetGuessType(),L"S")) );
```

C++ Builder

```
Exicalendarlib_tlb::IICalendarPtr ICalendar1 =
Variant::CreateObject(L"Exontrol.ICalendar.1");
Exicalendarlib_tlb::IComponentPtr var_Component = ICalendar1->Content-
>Components->Add(L"VCALENDAR");
   var_Component->Properties->Add(L"Duration",TVariant(ICalendar1-
>get_toICalendar(TVariant(3.325),Exicalendarlib_tlb::PropertyTypeEnum::exPropertyTyp

Exicalendarlib_tlb::IPropertyPtr p = ICalendar1->Root->Properties-
>get_Item(TVariant("Duration"));
String i = ICalendar1->toICalendar[TVariant(p->get_Value()),p->GuessType];
OutputDebugString( L"icalendar:" );
OutputDebugString( L"i" );
OutputDebugString( L"all:" );
OutputDebugString( PChar(ICalendar1->valuesFromICalendar[L"i",p-
>GuessType,L""]) );
OutputDebugString( L"duration:" );
OutputDebugString( PChar(ICalendar1->valuesFromICalendar[L"i",p-
>GuessType,L"Duration"]) );
OutputDebugString( L"weeks:" );
OutputDebugString( PChar(ICalendar1->valuesFromICalendar[L"i",p-
>GuessType,L"W"]) );
OutputDebugString( L"days:" );
OutputDebugString( PChar(ICalendar1->valuesFromICalendar[L"i",p-
>GuessType,L"D"]) );
```

```
OutputDebugString( L"hour:" );
OutputDebugString( PChar(ICalendar1->valuesFromICalendar[L"i",p-
>GuessType,L"H"]) );
OutputDebugString( L"min:" );
OutputDebugString( PChar(ICalendar1->valuesFromICalendar[L"i",p-
>GuessType,L"M"]) );
OutputDebugString( L"sec:" );
OutputDebugString( PChar(ICalendar1->valuesFromICalendar[L"i",p-
>GuessType,L"S"]) );
```

C#

```
// Add 'exontrol.exicalendar.dll(ExICalendar.dll)' reference to your project.
exontrol.EXICALENDARLib.exicalendar exicalendar1 = new
exontrol.EXICALENDARLib.exicalendar();
exontrol.EXICALENDARLib.Component var_Component =
exicalendar1.Content.Components.Add("VCALENDAR");

var_Component.Properties.Add("Duration",exicalendar1.get_toICalendar(3.325,exontro

exontrol.EXICALENDARLib.Property p = exicalendar1.Root.Properties["Duration"];
string i = exicalendar1.get_toICalendar(p.Value,p.GuessType);
System.Diagnostics.Debug.Print( "icalendar:" );
System.Diagnostics.Debug.Print( i.ToString() );
System.Diagnostics.Debug.Print( "all:" );
System.Diagnostics.Debug.Print(
exicalendar1.get_valuesFromICalendar(i.ToString(),p.GuessType,"").ToString() );
System.Diagnostics.Debug.Print( "duration:" );
System.Diagnostics.Debug.Print(
exicalendar1.get_valuesFromICalendar(i.ToString(),p.GuessType,"Duration").ToString()
 );
System.Diagnostics.Debug.Print( "weeks:" );
System.Diagnostics.Debug.Print(
exicalendar1.get_valuesFromICalendar(i.ToString(),p.GuessType,"W").ToString() );
System.Diagnostics.Debug.Print( "days:" );
System.Diagnostics.Debug.Print(
exicalendar1.get_valuesFromICalendar(i.ToString(),p.GuessType,"D").ToString() );
System.Diagnostics.Debug.Print( "hour:" );
System.Diagnostics.Debug.Print(
exicalendar1.get_valuesFromICalendar(i.ToString(),p.GuessType,"H").ToString() );
System.Diagnostics.Debug.Print( "min:" );
System.Diagnostics.Debug.Print(
exicalendar1.get_valuesFromICalendar(i.ToString(),p.GuessType,"M").ToString() );
System.Diagnostics.Debug.Print( "sec:" );
System.Diagnostics.Debug.Print(
exicalendar1.get_valuesFromICalendar(i.ToString(),p.GuessType,"S").ToString() );
```

JScript/JavaScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:D6C87100-38E2-4ABB-8AC2-4C0097AEE2D6"
id="ICalendar1"></OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
   var var_Component = ICalendar1.Content.Components.Add("VCALENDAR");
      var_Component.Properties.Add("Duration",ICalendar1.toICalendar(3.325,6));
   var p = ICalendar1.Root.Properties.Item("Duration");
   var i = ICalendar1.toICalendar(p.Value,p.GuessType);
   alert( "icalendar:" );
   alert( i );
   alert( "all:" );
   alert( ICalendar1.valuesFromICalendar(i,p.GuessType,"") );
   alert( "duration:" );
   alert( ICalendar1.valuesFromICalendar(i,p.GuessType,"Duration") );
   alert( "weeks:" );
   alert( ICalendar1.valuesFromICalendar(i,p.GuessType,"W") );
   alert( "days:" );
   alert( ICalendar1.valuesFromICalendar(i,p.GuessType,"D") );
   alert( "hour:" );
   alert( ICalendar1.valuesFromICalendar(i,p.GuessType,"H") );
```

```
    alert( "min:" );
    alert( ICalendar1.valuesFromICalendar(i,p.GuessType,"M") );
    alert( "sec:" );
    alert( ICalendar1.valuesFromICalendar(i,p.GuessType,"S") );
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:D6C87100-38E2-4ABB-8AC2-4C0097AEE2D6"
id="ICalendar1"></OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With ICalendar1
        With .Content.Components.Add("VCALENDAR")
            .Properties.Add "Duration",ICalendar1.toICalendar(3.325,6)
        End With
        Set p = .Root.Properties.Item("Duration")
        i = .toICalendar(p.Value,p.GuessType) ' p.GuessType
        alert( "icalendar:" )
        alert( i )
        alert( "all:" )
        alert( .valuesFromICalendar(i,p.GuessType,"") ) ' p.GuessType
        alert( "duration:" )
        alert( .valuesFromICalendar(i,p.GuessType,"Duration") ) ' p.GuessType
        alert( "weeks:" )
        alert( .valuesFromICalendar(i,p.GuessType,"W") ) ' p.GuessType
        alert( "days:" )
        alert( .valuesFromICalendar(i,p.GuessType,"D") ) ' p.GuessType
        alert( "hour:" )
        alert( .valuesFromICalendar(i,p.GuessType,"H") ) ' p.GuessType
        alert( "min:" )
        alert( .valuesFromICalendar(i,p.GuessType,"M") ) ' p.GuessType
        alert( "sec:" )
        alert( .valuesFromICalendar(i,p.GuessType,"S") ) ' p.GuessType
    End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
EXICALENDARLib.ICalendar axICalendar1 = new EXICALENDARLib.ICalendar();
EXICALENDARLib.Component var_Component =
axICalendar1.Content.Components.Add("VCALENDAR");

var_Component.Properties.Add("Duration",axICalendar1.get_toICalendar(3.325,EXICALI

EXICALENDARLib.Property p = axICalendar1.Root.Properties["Duration"];
string i = axICalendar1.get_toICalendar(p.Value,p.GuessType);
System.Diagnostics.Debug.Print( "icalendar:" );
System.Diagnostics.Debug.Print( i.ToString() );
System.Diagnostics.Debug.Print( "all:" );
System.Diagnostics.Debug.Print(
axICalendar1.get_valuesFromICalendar(i.ToString(),p.GuessType,"").ToString() );
System.Diagnostics.Debug.Print( "duration:" );
System.Diagnostics.Debug.Print(
axICalendar1.get_valuesFromICalendar(i.ToString(),p.GuessType,"Duration").ToString()
 );
System.Diagnostics.Debug.Print( "weeks:" );
System.Diagnostics.Debug.Print(
axICalendar1.get_valuesFromICalendar(i.ToString(),p.GuessType,"W").ToString() );
System.Diagnostics.Debug.Print( "days:" );
System.Diagnostics.Debug.Print(
axICalendar1.get_valuesFromICalendar(i.ToString(),p.GuessType,"D").ToString() );
System.Diagnostics.Debug.Print( "hour:" );
System.Diagnostics.Debug.Print(
axICalendar1.get_valuesFromICalendar(i.ToString(),p.GuessType,"H").ToString() );
System.Diagnostics.Debug.Print( "min:" );
System.Diagnostics.Debug.Print(
axICalendar1.get_valuesFromICalendar(i.ToString(),p.GuessType,"M").ToString() );
System.Diagnostics.Debug.Print( "sec:" );
System.Diagnostics.Debug.Print(
axICalendar1.get_valuesFromICalendar(i.ToString(),p.GuessType,"S").ToString() );
```

X++ (Dynamics Ax 2009)

```
public void init()
{
    COM com_Component,com_Properties,com_exicalendar1,com_p;
    anytype exicalendar1,p,var_Component,var_Properties;
    str i;
    ;

    super();

    // Add 'exicalendar.dll(ExICalendar.dll)' reference to your project.
    exicalendar1 = COM::createFromObject(new EXICALENDARLib.exicalendar());
com_exicalendar1 = exicalendar1;
    var_Component =
COM::createFromObject(com_exicalendar1.Content().Components()).Add("VCALENDAR
 com_Component = var_Component;
        var_Properties = COM::createFromObject(com_Component.Properties());
com_Properties = var_Properties;

com_Properties.Add("Duration",COMVariant::createFromStr(com_exicalendar1.toICalen

    p =
COM::createFromObject(com_exicalendar1.Root().Properties()).Item("Duration");
com_p = p;
    i = com_exicalendar1.toICalendar(p.Value(),p.GuessType());
    print( "icalendar:" );
    print( i );
    print( "all:" );
    print( com_exicalendar1.valuesFromICalendar(i,p.GuessType(),"") );
    print( "duration:" );
    print( com_exicalendar1.valuesFromICalendar(i,p.GuessType(),"Duration") );
    print( "weeks:" );
    print( com_exicalendar1.valuesFromICalendar(i,p.GuessType(),"W") );
    print( "days:" );
    print( com_exicalendar1.valuesFromICalendar(i,p.GuessType(),"D") );
    print( "hour:" );
    print( com_exicalendar1.valuesFromICalendar(i,p.GuessType(),"H") );
    print( "min:" );
    print( com_exicalendar1.valuesFromICalendar(i,p.GuessType(),"M") );
    print( "sec:" );
    print( com_exicalendar1.valuesFromICalendar(i,p.GuessType(),"S") );
}
```

Delphi 8 (.NET only)

```
AxICalendar1 :=
(ComObj.CreateComObject(ComObj.ProgIDToClassID('Exontrol.ICalendar.1')) as
EXICALENDARLib.ICalendar);
with AxICalendar1 do
begin
    with Content.Components.Add('VCALENDAR') do
    begin

Properties.Add('Duration',TObject(AxICalendar1.toICalendar[TObject(3.325),EXICALEND

    end;
    p := Root.Properties.Item['Duration'];
    i := get_toICalendar(p.Value,p.GuessType);
    OutputDebugString( 'icalendar:' );
    OutputDebugString( i );
    OutputDebugString( 'all:' );
    OutputDebugString( get_valuesFromICalendar(i,p.GuessType,'') );
    OutputDebugString( 'duration:' );
    OutputDebugString( get_valuesFromICalendar(i,p.GuessType,'Duration') );
    OutputDebugString( 'weeks:' );
    OutputDebugString( get_valuesFromICalendar(i,p.GuessType,'W') );
    OutputDebugString( 'days:' );
    OutputDebugString( get_valuesFromICalendar(i,p.GuessType,'D') );
    OutputDebugString( 'hour:' );
    OutputDebugString( get_valuesFromICalendar(i,p.GuessType,'H') );
    OutputDebugString( 'min:' );
    OutputDebugString( get_valuesFromICalendar(i,p.GuessType,'M') );
    OutputDebugString( 'sec:' );
    OutputDebugString( get_valuesFromICalendar(i,p.GuessType,'S') );
end
```

Delphi (standard)

```
ICalendar1 :=
(IUnknown(ComObj.CreateComObject(ComObj.ProgIDToClassID('Exontrol.ICalendar.1'))
 as EXICALENDARLib_TLB.ICalendar);
with ICalendar1 do
begin
    with Content.Components.Add('VCALENDAR') do
    begin

Properties.Add('Duration',OleVariant(ICalendar1.toICalendar[OleVariant(3.325),EXICALE

    end;
    p := Root.Properties.Item['Duration'];
    i := toICalendar[p.Value,p.GuessType];
    OutputDebugString( 'icalendar:' );
    OutputDebugString( i );
    OutputDebugString( 'all:' );
    OutputDebugString( valuesFromICalendar[i,p.GuessType,''] );
    OutputDebugString( 'duration:' );
    OutputDebugString( valuesFromICalendar[i,p.GuessType,'Duration'] );
    OutputDebugString( 'weeks:' );
    OutputDebugString( valuesFromICalendar[i,p.GuessType,'W'] );
    OutputDebugString( 'days:' );
    OutputDebugString( valuesFromICalendar[i,p.GuessType,'D'] );
    OutputDebugString( 'hour:' );
    OutputDebugString( valuesFromICalendar[i,p.GuessType,'H'] );
    OutputDebugString( 'min:' );
    OutputDebugString( valuesFromICalendar[i,p.GuessType,'M'] );
    OutputDebugString( 'sec:' );
    OutputDebugString( valuesFromICalendar[i,p.GuessType,'S'] );
end
```

VFP

```
thisform.ICalendar1 = CreateObject("Exontrol.ICalendar.1")
with thisform.ICalendar1
    with .Content.Components.Add("VCALENDAR")
        .Properties.Add("Duration",thisform.ICalendar1.toICalendar(3.325,6))
    endwith
    p = .Root.Properties.Item("Duration")
    i = .toICalendar(p.Value,p.GuessType) && p.GuessType
    DEBUGOUT( "icalendar:" )
    DEBUGOUT( i )
    DEBUGOUT( "all:" )
    DEBUGOUT( .valuesFromICalendar(i,p.GuessType,"") ) && p.GuessType
    DEBUGOUT( "duration:" )
    DEBUGOUT( .valuesFromICalendar(i,p.GuessType,"Duration") ) && p.GuessType
    DEBUGOUT( "weeks:" )
    DEBUGOUT( .valuesFromICalendar(i,p.GuessType,"W") ) && p.GuessType
    DEBUGOUT( "days:" )
    DEBUGOUT( .valuesFromICalendar(i,p.GuessType,"D") ) && p.GuessType
    DEBUGOUT( "hour:" )
    DEBUGOUT( .valuesFromICalendar(i,p.GuessType,"H") ) && p.GuessType
    DEBUGOUT( "min:" )
    DEBUGOUT( .valuesFromICalendar(i,p.GuessType,"M") ) && p.GuessType
    DEBUGOUT( "sec:" )
    DEBUGOUT( .valuesFromICalendar(i,p.GuessType,"S") ) && p.GuessType
endwith
```

dBASE Plus

```
local i,oICalendar,p,var_Component

oICalendar = new OleAutoClient("Exontrol.ICalendar.1")
```

```
var_Component = oICalendar.Content.Components.Add("VCALENDAR")
	var_Component.Properties.Add("Duration",oICalendar.toICalendar(3.325,6))
p = oICalendar.Root.Properties.Item("Duration")
i = oICalendar.toICalendar(p.Value,p.GuessType)
? "icalendar:"
? Str(i)
? "all:"
? Str(oICalendar.valuesFromICalendar(Str(i),p.GuessType,""))
? "duration:"
? Str(oICalendar.valuesFromICalendar(Str(i),p.GuessType,"Duration"))
? "weeks:"
? Str(oICalendar.valuesFromICalendar(Str(i),p.GuessType,"W"))
? "days:"
? Str(oICalendar.valuesFromICalendar(Str(i),p.GuessType,"D"))
? "hour:"
? Str(oICalendar.valuesFromICalendar(Str(i),p.GuessType,"H"))
? "min:"
? Str(oICalendar.valuesFromICalendar(Str(i),p.GuessType,"M"))
? "sec:"
? Str(oICalendar.valuesFromICalendar(Str(i),p.GuessType,"S"))
```

XBasic (Alpha Five)

```
Dim i as
Dim oICalendar as P
Dim p as P
Dim var_Component as P

oICalendar = OLE.Create("Exontrol.ICalendar.1")

var_Component = oICalendar.Content.Components.Add("VCALENDAR")
	var_Component.Properties.Add("Duration",oICalendar.toICalendar(3.325,6))
p = oICalendar.Root.Properties.Item("Duration")
i = oICalendar.toICalendar(p.Value,p.GuessType)
? "icalendar:"
? i
? "all:"
? oICalendar.valuesFromICalendar(i,p.GuessType,"")
? "duration:"
? oICalendar.valuesFromICalendar(i,p.GuessType,"Duration")
? "weeks:"
? oICalendar.valuesFromICalendar(i,p.GuessType,"W")
? "days:"
? oICalendar.valuesFromICalendar(i,p.GuessType,"D")
? "hour:"
? oICalendar.valuesFromICalendar(i,p.GuessType,"H")
? "min:"
? oICalendar.valuesFromICalendar(i,p.GuessType,"M")
? "sec:"
? oICalendar.valuesFromICalendar(i,p.GuessType,"S")
```

Visual Objects

```
local var_Component as IComponent
local p as IProperty
local i as USUAL

oDCOCX_Exontrol1 := IICalendar{"Exontrol.ICalendar.1"}
var_Component := oDCOCX_Exontrol1:Content:Components:Add("VCALENDAR")
   var_Component:Properties:Add("Duration",oDCOCX_Exontrol1:
[toICalendar,3.325,exPropertyTypeDuration])
p := oDCOCX_Exontrol1:Root:Properties:[Item,"Duration"]
i := oDCOCX_Exontrol1:[toICalendar,p:Value,p:GuessType]
OutputDebugString(String2Psz( "icalendar:" ))
OutputDebugString(String2Psz( AsString(i) ))
OutputDebugString(String2Psz( "all:" ))
OutputDebugString(String2Psz( AsString(oDCOCX_Exontrol1:
[valuesFromICalendar,AsString(i),p:GuessType,""]) ))
OutputDebugString(String2Psz( "duration:" ))
OutputDebugString(String2Psz( AsString(oDCOCX_Exontrol1:
[valuesFromICalendar,AsString(i),p:GuessType,"Duration"]) ))
```

```
OutputDebugString(String2Psz( "weeks:" ))
OutputDebugString(String2Psz( AsString(oDCOCX_Exontrol1:
[valuesFromICalendar,AsString(i),p:GuessType,"W"]) ))
OutputDebugString(String2Psz( "days:" ))
OutputDebugString(String2Psz( AsString(oDCOCX_Exontrol1:
[valuesFromICalendar,AsString(i),p:GuessType,"D"]) ))
OutputDebugString(String2Psz( "hour:" ))
OutputDebugString(String2Psz( AsString(oDCOCX_Exontrol1:
[valuesFromICalendar,AsString(i),p:GuessType,"H"]) ))
OutputDebugString(String2Psz( "min:" ))
OutputDebugString(String2Psz( AsString(oDCOCX_Exontrol1:
[valuesFromICalendar,AsString(i),p:GuessType,"M"]) ))
OutputDebugString(String2Psz( "sec:" ))
OutputDebugString(String2Psz( AsString(oDCOCX_Exontrol1:
[valuesFromICalendar,AsString(i),p:GuessType,"S"]) ))
```

PowerBuilder

```
OleObject oICalendar,p,var_Component
any i

oICalendar = CREATE OLEObject
oICalendar.ConnectToNewObject("Exontrol.ICalendar.1")

var_Component = oICalendar.Content.Components.Add("VCALENDAR")
   var_Component.Properties.Add("Duration",oICalendar.toICalendar(3.325,6))
p = oICalendar.Root.Properties.Item("Duration")
i = oICalendar.toICalendar(p.Value,p.GuessType)
MessageBox("Information",string( "icalendar:" ))
MessageBox("Information",string( String(i) ))
MessageBox("Information",string( "all:" ))
MessageBox("Information",string(
String(oICalendar.valuesFromICalendar(String(i),p.GuessType,"")) ))
MessageBox("Information",string( "duration:" ))
MessageBox("Information",string(
String(oICalendar.valuesFromICalendar(String(i),p.GuessType,"Duration")) ))
```

```
MessageBox("Information",string( "weeks:" ))
MessageBox("Information",string(
String(oICalendar.valuesFromICalendar(String(i),p.GuessType,"W")) ))
MessageBox("Information",string( "days:" ))
MessageBox("Information",string(
String(oICalendar.valuesFromICalendar(String(i),p.GuessType,"D")) ))
MessageBox("Information",string( "hour:" ))
MessageBox("Information",string(
String(oICalendar.valuesFromICalendar(String(i),p.GuessType,"H")) ))
MessageBox("Information",string( "min:" ))
MessageBox("Information",string(
String(oICalendar.valuesFromICalendar(String(i),p.GuessType,"M")) ))
MessageBox("Information",string( "sec:" ))
MessageBox("Information",string(
String(oICalendar.valuesFromICalendar(String(i),p.GuessType,"S")) ))
```

Visual DataFlex

```
Procedure OnCreate
    Forward Send OnCreate
    Variant oComICalendar1
    Get ComCreateObject "Exontrol.ICalendar.1" to oComICalendar1

    Variant voComponent
    Get ComContent to voComponent
    Handle hoComponent
    Get Create (RefClass(cComComponent)) to hoComponent
    Set pvComObject of hoComponent to voComponent
        Variant voComponents
        Get ComComponents of hoComponent to voComponents
        Handle hoComponents
        Get Create (RefClass(cComComponents)) to hoComponents
        Set pvComObject of hoComponents to voComponents
            Variant voComponent1
            Get ComAdd of hoComponents "VCALENDAR" to voComponent1
            Handle hoComponent1
            Get Create (RefClass(cComComponent)) to hoComponent1
            Set pvComObject of hoComponent1 to voComponent1
                Variant voProperties
                Get ComProperties of hoComponent1 to voProperties
                Handle hoProperties
                Get Create (RefClass(cComProperties)) to hoProperties
                Set pvComObject of hoProperties to voProperties
                    Variant vValue
                        Get ComtoICalendar 3.325 OLEexPropertyTypeDuration to vValue
                    Get ComAdd of hoProperties "Duration" vValue to Nothing
                Send Destroy to hoProperties
            Send Destroy to hoComponent1
        Send Destroy to hoComponents
    Send Destroy to hoComponent
    Variant v
    Variant voComponent2
    Get ComRoot to voComponent2
    Handle hoComponent2
    Get Create (RefClass(cComComponent)) to hoComponent2
    Set pvComObject of hoComponent2 to voComponent2
        Variant voProperties1
        Get ComProperties of hoComponent2 to voProperties1
        Handle hoProperties1
        Get Create (RefClass(cComProperties)) to hoProperties1
        Set pvComObject of hoProperties1 to voProperties1
            Get ComItem of hoProperties1 "Duration" to v
        Send Destroy to hoProperties1
    Send Destroy to hoComponent2
    Variant p
    Move v to p
    Variant i
    Get ComtoICalendar p p to i
    Showln "icalendar:" i
    Showln "all:" (ComvaluesFromICalendar(Self,i,p,""))
    Showln "duration:" (ComvaluesFromICalendar(Self,i,p,"Duration"))
    Showln "weeks:" (ComvaluesFromICalendar(Self,i,p,"W"))
    Showln "days:" (ComvaluesFromICalendar(Self,i,p,"D"))
```

```
   Showln "hour:" (ComvaluesFromICalendar(Self,i,p,"H"))
   Showln "min:" (ComvaluesFromICalendar(Self,i,p,"M"))
   Showln "sec:" (ComvaluesFromICalendar(Self,i,p,"S"))
End_Procedure
```

XBase++

```
#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
   LOCAL oForm
   LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
   LOCAL oICalendar
   LOCAL oComponent
   LOCAL p
   LOCAL i

   oForm := XbpDialog():new( AppDesktop() )
   oForm:drawingArea:clipChildren := .T.
   oForm:create( ,,{100,100}, {640,480},, .F. )
   oForm:close  := {|| PostAppEvent( xbeP_Quit )}

   oICalendar := XbpActiveXControl():new( oForm:drawingArea )
   oICalendar:CLSID  := "Exontrol.ICalendar.1" /*{D6C87100-38E2-4ABB-8AC2-
4C0097AEE2D6}*/
   oICalendar:create(,, {10,60},{610,370} )

      oComponent := oICalendar:Content():Components():Add("VCALENDAR")

oComponent:Properties():Add("Duration",oICalendar:toICalendar(3.325,6/*exPropertyT

      p := oICalendar:Root:Properties:Item("Duration")
      i := oICalendar:toICalendar(p:Value(),p:GuessType())
      DevOut( "icalendar:" )
      DevOut( Transform(i,"") )
      DevOut( "all:" )
```

```
        DevOut(
Transform(oICalendar:valuesFromICalendar(Transform(i,""),p:GuessType(),""),"") )
        DevOut( "duration:" )
        DevOut(
Transform(oICalendar:valuesFromICalendar(Transform(i,""),p:GuessType(),"Duration"),"
 )
        DevOut( "weeks:" )
        DevOut(
Transform(oICalendar:valuesFromICalendar(Transform(i,""),p:GuessType(),"W"),"") )
        DevOut( "days:" )
        DevOut(
Transform(oICalendar:valuesFromICalendar(Transform(i,""),p:GuessType(),"D"),"") )
        DevOut( "hour:" )
        DevOut(
Transform(oICalendar:valuesFromICalendar(Transform(i,""),p:GuessType(),"H"),"") )
        DevOut( "min:" )
        DevOut(
Transform(oICalendar:valuesFromICalendar(Transform(i,""),p:GuessType(),"M"),"") )
        DevOut( "sec:" )
        DevOut(
Transform(oICalendar:valuesFromICalendar(Transform(i,""),p:GuessType(),"S"),"") )

    oForm:Show()
    DO WHILE nEvent != xbeP_Quit
        nEvent := AppEvent( @mp1, @mp2, @oXbp )
        oXbp:handleEvent( nEvent, mp1, mp2 )
    ENDDO
RETURN
```

property ICalendar.valuesToICalendar (Values as String, Type as PropertyTypeEnum) as String

Converts the values to a value of ICalendar format. For instance, valuesToICalendar("Duration=15.5",exPropertyTypeDuration) returns "P15DT12H", which indicates 15 days and 12 hours.

Type	Description
Values as String	A String expression that specifies the Parameter=Value[;Parameter=Value] that indicates the expression to be converted to iCalendar format
Type as PropertyTypeEnum	A PropertyTypeEnum expression that indicates the type of the Value to be converted.
String	A String expression that specifies the iCalendar format

The valuesToICalendar property converts the known parameter to iCalendar format. The valuesFromICalendar property returns values for known parameters of the giving expression (available for exPropertyTypeDuration, exPropertyTypePeriod and exPropertyTypeRecur types as explained bellow). The fromICalendar property converts the ICalendar value to a VARIANT expression. The toICalendar property is the reverse function, so it converts a VARIANT expression back to iCalendar format.

The valuesToICalendar property is available for following types:

- exPropertyTypeDuration, The exPropertyTypeDuration expression supports the following parameters:
  - "-", specifies whether the duration is negative or positive
  - "W", indicates the number of weeks
  - "D", indicates the number of days
  - "H", indicates the number of hours
  - "M", indicates the number of minutes
  - "S", indicates the number of seconds
  - "Duration", indicates the duration in days for integer part, while the fractional part specifies hours, minutes and seconds.

- exPropertyTypePeriod, The exPropertyTypePeriod expression supports the following parameters:
  - "Start", indicates the date to start the period
  - "End", indicates the date to end the period
  - "-", specifies whether the duration of the period is negative or positive
  - "W", indicates the number of weeks within period
  - "D", indicates the number of days within period
  - "H", indicates the number of hours within period

    - "M", indicates the number of minutes within period
    - "S", indicates the number of seconds within period
    - "Duration", indicates the duration of the period in days for integer part, while the fractional part specifies hours, minutes and seconds.

- exPropertyTypeRecur, The exPropertyTypeRecur expression supports the following parameters:
    - "FREQ", identifies the type of recurrence rule, and could by any of the following: "SECONDLY", "MINUTELY", "HOURLY", "DAILY", "WEEKLY", "MONTHLY", "YEARLY"
    - "UNTIL", defines a DATE value that bounds the recurrence rule in an inclusive manner.
    - "COUNT", defines the number of occurrences at which to range-bound the recurrence.
    - "INTERVAL", contains a positive integer representing at which intervals the recurrence rule repeats.
    - "BYSECOND", specifies a COMMA-separated list of seconds within a minute.
    - "BYMINUTE", specifies a COMMA-separated list of minutes within an hour
    - "BYHOUR", specifies a COMMA-separated list of hours of the day
    - "BYDAY", specifies a COMMA-separated list of days of the week; SU indicates Sunday; MO indicates Monday; TU indicates Tuesday; WE indicates Wednesday; TH indicates Thursday; FR indicates Friday; and SA indicates Saturday. Each BYDAY value can also be preceded by a positive (+n) or negative (-n) integer. If present, this indicates the nth occurrence of a specific day within the MONTHLY or YEARLY "RRULE".
    - "BYMONTHDAY", specifies a COMMA-separated list of days of the month. Valid values are 1 to 31 or -31 to -1.
    - "BYYEARDAY", specifies a COMMA-separated list of days of the year. Valid values are 1 to 366 or -366 to -1.
    - "BYWEEKNO", specifies a COMMA-separated list of ordinals specifying weeks of the year. Valid values are 1 to 53 or -53 to -1.
    - "BYMONTH", specifies a COMMA-separated list of months of the year. Valid values are 1 to 12.
    - "BYSETPOS", specifies a COMMA-separated list of values that corresponds to the nth occurrence within the set of recurrence instances specified by the rule.
    - "WKST", specifies the day on which the workweek starts. Valid values are MO, TU, WE, TH, FR, SA, and SU. The default value is MO.

The following samples show how you can add a property of duration type:

VBA (MS Access, Excell...)

```
Set ICalendar1 = CreateObject("Exontrol.ICalendar.1")
With ICalendar1
  With .Content.Components.Add("VCALENDAR")
    .Properties.Add "Duration1",ICalendar1.toICalendar(2.5,6)
    With .Properties.Add("Duration2")
      .Value = 2.5
      .Type = 6
    End With
    .Properties.Add "Duration3",ICalendar1.valuesToICalendar("D=2;H=12",6)
  End With
  Debug.Print( .Save )
End With
```

VB6

```
Set ICalendar1 = CreateObject("Exontrol.ICalendar.1")
With ICalendar1
  With .Content.Components.Add("VCALENDAR")
    .Properties.Add
"Duration1",ICalendar1.toICalendar(2.5,exPropertyTypeDuration)
    With .Properties.Add("Duration2")
      .Value = 2.5
      .Type = exPropertyTypeDuration
    End With
    .Properties.Add
"Duration3",ICalendar1.valuesToICalendar("D=2;H=12",exPropertyTypeDuration)
  End With
  Debug.Print( .Save )
End With
```

VB.NET

```
' Add 'exontrol.exicalendar.dll(ExICalendar.dll)' reference to your project.
Exicalendar1 = New exontrol.EXICALENDARLib.exicalendar()
With Exicalendar1
  With .Content.Components.Add("VCALENDAR")

.Properties.Add("Duration1",Exicalendar1.get_toICalendar(2.5,exontrol.EXICALENDARI
        With .Properties.Add("Duration2")
            .Value = 2.5
            .Type =
exontrol.EXICALENDARLib.PropertyTypeEnum.exPropertyTypeDuration
        End With

.Properties.Add("Duration3",Exicalendar1.get_valuesToICalendar("D=2;H=12",exontro

    End With
    Debug.Print( .Save() )
End With
```

VB.NET for /COM

```
AxICalendar1 = CreateObject("Exontrol.ICalendar.1")
With AxICalendar1
    With .Content.Components.Add("VCALENDAR")

.Properties.Add("Duration1",AxICalendar1.toICalendar(2.5,EXICALENDARLib.PropertyT

        With .Properties.Add("Duration2")
            .Value = 2.5
            .Type = EXICALENDARLib.PropertyTypeEnum.exPropertyTypeDuration
        End With

.Properties.Add("Duration3",AxICalendar1.valuesToICalendar("D=2;H=12",EXICALEND

    End With
    Debug.Print( .Save() )
End With
```

C++

```
/*
    Includes the definition for CreateObject function like follows:

    #include <comdef.h>
    IUnknownPtr CreateObject( BSTR Object )
    {
        IUnknownPtr spResult;
        spResult.CreateInstance( Object );
        return spResult;
    };

*/
/*
    Copy and paste the following directives to your header file as
    it defines the namespace 'EXICALENDARLib' for the library: 'ICalendar 1.0 Type
Library'

    #import <ExICalendar.dll>
    using namespace EXICALENDARLib;
*/
EXICALENDARLib::IICalendarPtr spICalendar1 =
::CreateObject(L"Exontrol.ICalendar.1");
EXICALENDARLib::IComponentPtr var_Component = spICalendar1->GetContent()-
>GetComponents()->Add(L"VCALENDAR");
   var_Component->GetProperties()->Add(L"Duration1",spICalendar1-
>GettoICalendar(double(2.5),EXICALENDARLib::exPropertyTypeDuration));
   EXICALENDARLib::IPropertyPtr var_Property = var_Component->GetProperties()-
>Add(L"Duration2",vtMissing);
      var_Property->PutValue(double(2.5));
      var_Property->PutType(EXICALENDARLib::exPropertyTypeDuration);
   var_Component->GetProperties()->Add(L"Duration3",spICalendar1-
>GetvaluesToICalendar(L"D=2;H=12",EXICALENDARLib::exPropertyTypeDuration));
OutputDebugStringW( spICalendar1->Save() );
```

C++ Builder

```
Exicalendarlib_tlb::IICalendarPtr ICalendar1 =
Variant::CreateObject(L"Exontrol.ICalendar.1");
Exicalendarlib_tlb::IComponentPtr var_Component = ICalendar1->Content-
>Components->Add(L"VCALENDAR");
```

```
    var_Component->Properties->Add(L"Duration1",TVariant(ICalendar1-
>get_toICalendar(TVariant(2.5),Exicalendarlib_tlb::PropertyTypeEnum::exPropertyType

    Exicalendarlib_tlb::IPropertyPtr var_Property = var_Component->Properties-
>Add(L"Duration2",TNoParam());
        var_Property->set_Value(TVariant(2.5));
        var_Property->Type =
Exicalendarlib_tlb::PropertyTypeEnum::exPropertyTypeDuration;
    var_Component->Properties->Add(L"Duration3",TVariant(ICalendar1-
>get_valuesToICalendar(L"D=2;H=12",Exicalendarlib_tlb::PropertyTypeEnum::exPrope

OutputDebugString( ICalendar1->Save() );
```

C#

```
// Add 'exontrol.exicalendar.dll(ExICalendar.dll)' reference to your project.
exontrol.EXICALENDARLib.exicalendar exicalendar1 = new
exontrol.EXICALENDARLib.exicalendar();
exontrol.EXICALENDARLib.Component var_Component =
exicalendar1.Content.Components.Add("VCALENDAR");

var_Component.Properties.Add("Duration1",exicalendar1.get_toICalendar(2.5,exontro

    exontrol.EXICALENDARLib.Property var_Property =
var_Component.Properties.Add("Duration2",null);
        var_Property.Value = 2.5;
        var_Property.Type =
exontrol.EXICALENDARLib.PropertyTypeEnum.exPropertyTypeDuration;

var_Component.Properties.Add("Duration3",exicalendar1.get_valuesToICalendar("D=

System.Diagnostics.Debug.Print( exicalendar1.Save() );
```

JScript/JavaScript

```
<BODY onload="Init()">
```

```
<OBJECT CLASSID="clsid:D6C87100-38E2-4ABB-8AC2-4C0097AEE2D6"
id="ICalendar1"></OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Component = ICalendar1.Content.Components.Add("VCALENDAR");
        var_Component.Properties.Add("Duration1",ICalendar1.toICalendar(2.5,6));
        var var_Property = var_Component.Properties.Add("Duration2",null);
            var_Property.Value = 2.5;
            var_Property.Type = 6;

var_Component.Properties.Add("Duration3",ICalendar1.valuesToICalendar("D=2;H=1

    alert( ICalendar1.Save() );
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:D6C87100-38E2-4ABB-8AC2-4C0097AEE2D6"
id="ICalendar1"></OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With ICalendar1
        With .Content.Components.Add("VCALENDAR")
            .Properties.Add "Duration1",ICalendar1.toICalendar(2.5,6)
            With .Properties.Add("Duration2")
                .Value = 2.5
                .Type = 6
            End With
            .Properties.Add "Duration3",ICalendar1.valuesToICalendar("D=2;H=12",6)
        End With
        alert( .Save )
    End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
EXICALENDARLib.ICalendar axICalendar1 = new EXICALENDARLib.ICalendar();
EXICALENDARLib.Component var_Component =
axICalendar1.Content.Components.Add("VCALENDAR");

var_Component.Properties.Add("Duration1",axICalendar1.get_toICalendar(2.5,EXICAL

    EXICALENDARLib.Property var_Property =
var_Component.Properties.Add("Duration2",null);
        var_Property.Value = 2.5;
        var_Property.Type =
EXICALENDARLib.PropertyTypeEnum.exPropertyTypeDuration;

var_Component.Properties.Add("Duration3",axICalendar1.get_valuesToICalendar("D=

System.Diagnostics.Debug.Print( axICalendar1.Save() );
```

X++ (Dynamics Ax 2009)

```
public void init()
{
    COM com_Component,com_Properties,com_Property,com_exicalendar1;
    anytype exicalendar1,var_Component,var_Properties,var_Property;
    ;

    super();

    // Add 'exicalendar.dll(ExICalendar.dll)' reference to your project.
    exicalendar1 = COM::createFromObject(new EXICALENDARLib.exicalendar());
```

```
com_exicalendar1 = exicalendar1;
    var_Component =
COM::createFromObject(com_exicalendar1.Content().Components()).Add("VCALENDAR
 com_Component = var_Component;
        var_Properties = COM::createFromObject(com_Component.Properties());
com_Properties = var_Properties;

com_Properties.Add("Duration1",COMVariant::createFromStr(com_exicalendar1.toICale

        var_Properties = COM::createFromObject(com_Component.Properties());
com_Properties = var_Properties;
        var_Property = COM::createFromObject(com_Properties).Add("Duration2");
com_Property = var_Property;
            com_Property.Value(COMVariant::createFromReal(2.5));
            com_Property.Type(6/*exPropertyTypeDuration*/);
        var_Properties = COM::createFromObject(com_Component.Properties());
com_Properties = var_Properties;

com_Properties.Add("Duration3",COMVariant::createFromStr(com_exicalendar1.values

    print( com_exicalendar1.Save() );
}
```

Delphi 8 (.NET only)

```
AxICalendar1 :=
(ComObj.CreateComObject(ComObj.ProgIDToClassID('Exontrol.ICalendar.1')) as
EXICALENDARLib.ICalendar);
with AxICalendar1 do
begin
    with Content.Components.Add('VCALENDAR') do
    begin

Properties.Add('Duration1',TObject(AxICalendar1.toICalendar[TObject(2.5),EXICALEND

        with Properties.Add('Duration2',Nil) do
        begin
```

```
            Value := TObject(2.5);
            Type := EXICALENDARLib.PropertyTypeEnum.exPropertyTypeDuration;
        end;

Properties.Add('Duration3',TObject(AxICalendar1.valuesToICalendar['D=2;H=12',EXIC

    end;
    OutputDebugString( Save() );
end
```

Delphi (standard)

```
ICalendar1 :=
(IUnknown(ComObj.CreateComObject(ComObj.ProgIDToClassID('Exontrol.ICalendar.1'))
 as EXICALENDARLib_TLB.ICalendar);
with ICalendar1 do
begin
    with Content.Components.Add('VCALENDAR') do
    begin

Properties.Add('Duration1',OleVariant(ICalendar1.toICalendar[OleVariant(2.5),EXICALE

        with Properties.Add('Duration2',Null) do
        begin
            Value := OleVariant(2.5);
            Type := EXICALENDARLib_TLB.exPropertyTypeDuration;
        end;

Properties.Add('Duration3',OleVariant(ICalendar1.valuesToICalendar['D=2;H=12',EXIC

    end;
    OutputDebugString( Save() );
end
```

VFP

```
thisform.ICalendar1 = CreateObject("Exontrol.ICalendar.1")
with thisform.ICalendar1
```

```
    with .Content.Components.Add("VCALENDAR")
        .Properties.Add("Duration1",thisform.ICalendar1.toICalendar(2.5,6))
        with .Properties.Add("Duration2")
            .Value = 2.5
            .Type = 6
        endwith

.Properties.Add("Duration3",thisform.ICalendar1.valuesToICalendar("D=2;H=12",6))
    endwith
    DEBUGOUT( .Save )
endwith
```

dBASE Plus

```
local oICalendar,var_Component,var_Property

oICalendar = new OleAutoClient("Exontrol.ICalendar.1")

var_Component = oICalendar.Content.Components.Add("VCALENDAR")
    var_Component.Properties.Add("Duration1",oICalendar.toICalendar(2.5,6))
    var_Property = var_Component.Properties.Add("Duration2")
        var_Property.Value = 2.5
        var_Property.Type = 6

var_Component.Properties.Add("Duration3",oICalendar.valuesToICalendar("D=2;H=1

? oICalendar.Save()
```

XBasic (Alpha Five)

```
Dim oICalendar as P
Dim var_Component as P
Dim var_Property as P

oICalendar = OLE.Create("Exontrol.ICalendar.1")

var_Component = oICalendar.Content.Components.Add("VCALENDAR")
```

```
   var_Component.Properties.Add("Duration1",oICalendar.toICalendar(2.5,6))
   var_Property = var_Component.Properties.Add("Duration2")
      var_Property.Value = 2.5
      var_Property.Type = 6

var_Component.Properties.Add("Duration3",oICalendar.valuesToICalendar("D=2;H=1

? oICalendar.Save()
```

Visual Objects

```
local var_Component as IComponent
local var_Property as IProperty

oDCOCX_Exontrol1 := IICalendar{"Exontrol.ICalendar.1"}
var_Component := oDCOCX_Exontrol1:Content:Components:Add("VCALENDAR")
   var_Component:Properties:Add("Duration1",oDCOCX_Exontrol1:
[toICalendar,2.5,exPropertyTypeDuration])
   var_Property := var_Component:Properties:Add("Duration2",nil)
      var_Property:Value := 2.5
      var_Property:Type := exPropertyTypeDuration
   var_Component:Properties:Add("Duration3",oDCOCX_Exontrol1:
[valuesToICalendar,"D=2;H=12",exPropertyTypeDuration])
OutputDebugString(String2Psz( oDCOCX_Exontrol1:Save() ))
```

PowerBuilder

```
OleObject oICalendar,var_Component,var_Property

oICalendar = CREATE OLEObject
oICalendar.ConnectToNewObject("Exontrol.ICalendar.1")

var_Component = oICalendar.Content.Components.Add("VCALENDAR")
   var_Component.Properties.Add("Duration1",oICalendar.toICalendar(2.5,6))
   var_Property = var_Component.Properties.Add("Duration2")
      var_Property.Value = 2.5
```

```
        var_Property.Type = 6

var_Component.Properties.Add("Duration3",oICalendar.valuesToICalendar("D=2;H=1

MessageBox("Information",string( oICalendar.Save() ))
```

Visual DataFlex

```
Procedure OnCreate
    Forward Send OnCreate
    Variant oComICalendar1
    Get ComCreateObject "Exontrol.ICalendar.1" to oComICalendar1

    Variant voComponent
    Get ComContent to voComponent
    Handle hoComponent
    Get Create (RefClass(cComComponent)) to hoComponent
    Set pvComObject of hoComponent to voComponent
        Variant voComponents
        Get ComComponents of hoComponent to voComponents
        Handle hoComponents
        Get Create (RefClass(cComComponents)) to hoComponents
        Set pvComObject of hoComponents to voComponents
            Variant voComponent1
            Get ComAdd of hoComponents "VCALENDAR" to voComponent1
            Handle hoComponent1
            Get Create (RefClass(cComComponent)) to hoComponent1
            Set pvComObject of hoComponent1 to voComponent1
                Variant voProperties
                Get ComProperties of hoComponent1 to voProperties
                Handle hoProperties
                Get Create (RefClass(cComProperties)) to hoProperties
                Set pvComObject of hoProperties to voProperties
                    Variant vValue
                        Get ComtoICalendar 2.5 OLEexPropertyTypeDuration to vValue
                    Get ComAdd of hoProperties "Duration1" vValue to Nothing
                Send Destroy to hoProperties
                Variant voProperties1
                Get ComProperties of hoComponent1 to voProperties1
                Handle hoProperties1
                Get Create (RefClass(cComProperties)) to hoProperties1
                Set pvComObject of hoProperties1 to voProperties1
                    Variant voProperty
                    Get ComAdd of hoProperties1 "Duration2" Nothing to voProperty
                    Handle hoProperty
                    Get Create (RefClass(cComProperty)) to hoProperty
                    Set pvComObject of hoProperty to voProperty
                        Set ComValue of hoProperty to 2.5
                        Set ComType of hoProperty to OLEexPropertyTypeDuration
                    Send Destroy to hoProperty
                Send Destroy to hoProperties1
                Variant voProperties2
                Get ComProperties of hoComponent1 to voProperties2
                Handle hoProperties2
                Get Create (RefClass(cComProperties)) to hoProperties2
                Set pvComObject of hoProperties2 to voProperties2
                    Variant vValue1
                        Get ComvaluesToICalendar "D=2;H=12" OLEexPropertyTypeDuration
to vValue1
                    Get ComAdd of hoProperties2 "Duration3" vValue1 to Nothing
                Send Destroy to hoProperties2
            Send Destroy to hoComponent1
        Send Destroy to hoComponents
    Send Destroy to hoComponent
    Showln (ComSave(Self))
End_Procedure
```

XBase++

```
#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
```

```
   LOCAL oForm
   LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
   LOCAL oICalendar
   LOCAL oComponent
   LOCAL oProperty

   oForm := XbpDialog():new( AppDesktop() )
   oForm:drawingArea:clipChildren := .T.
   oForm:create( ,,{100,100}, {640,480},, .F. )
   oForm:close  := {|| PostAppEvent( xbeP_Quit )}

   oICalendar := XbpActiveXControl():new( oForm:drawingArea )
   oICalendar:CLSID  := "Exontrol.ICalendar.1" /*{D6C87100-38E2-4ABB-8AC2-
4C0097AEE2D6}*/
   oICalendar:create(,, {10,60},{610,370} )

      oComponent := oICalendar:Content():Components():Add("VCALENDAR")

oComponent:Properties():Add("Duration1",oICalendar:toICalendar(2.5,6/*exPropertyT

         oProperty := oComponent:Properties():Add("Duration2")
            oProperty:Value := 2.5
            oProperty:Type := 6/*exPropertyTypeDuration*/

oComponent:Properties():Add("Duration3",oICalendar:valuesToICalendar("D=2;H=12

      DevOut( oICalendar:Save() )

   oForm:Show()
   DO WHILE nEvent != xbeP_Quit
      nEvent := AppEvent( @mp1, @mp2, @oXbp )
      oXbp:handleEvent( nEvent, mp1, mp2 )
   ENDDO
RETURN
```

property ICalendar.Version as String

Retrieves the control's version.

Type	Description
String	A String expression that indicates the Version of the component you are running.

The Version property specifies the Version of the control is running.

Parameter object

The Parameter object holds information about a parameter of a property. The Parameters property of the Property accesses the Parameters collection.

The following is an example of a parameter:

RDATE;**VALUE**=DATE:19970304,19970504,19970704,19970904

The VALUE indicates the parameter of the RDATE property.

The Parameter object supports the following properties and methods:

Name	Description
Name	Indicates the parameter's name.
Property	Retrieves the parent property of the parameter.
toICalendar	Gets the iCalendar representation of the parameter.
UserData	Indicates any extra data associated with the parameter.
Value	Indicates the parameter's value(or values if a safe array is used).

property Parameter.Name as String

Indicates the parameter's name.

Type	Description
String	A String expression that specifies the name of the Parameter.

The Name property specifies the name of the Parameter. The Value property specifies the value/values of the parameter. The UserData property holds any extra data associated with the parameter object.

The following is an example of a parameter:

RDATE;**VALUE**=DATE:19970304,19970504,19970704,19970904

The VALUE specifies the name of the parameter.

property Parameter.Property as Property

Retrieves the parent property of the parameter.

Type	Description
Property	A Property object that specifies the parent property.

The Parent property indicates the parent property of the parameter. The Name property specifies the name of the Parameter. The Value property specifies the value/values of the parameter. The UserData property holds any extra data associated with the parameter object.

The following is an example of a parameter:

RDATE;VALUE=DATE:19970304,19970504,19970704,19970904

The RDATE indicates the parent property of the VALUE parameter.

property Parameter.toICalendar as String

Gets the iCalendar representation of the parameter.

Type	Description
String	A String expression that specifies the iCalendar format of the current parameter

The toICalendar property retrieves the iCalendar representation of the parameter. You can use the Load / LoadFile / LoadFileFromUnicode methods load iCalendar format, while Save / SaveFile / SaveFileAsUnicode methods save the control's content as iCalendar format.

property Parameter.UserData as Variant

Indicates any extra data associated with the parameter.

Type	Description
Variant	A VARIANT expression that specifies any extra data associated with the parameter.

The UserData property holds any extra data associated with the parameter object. The Value property specifies the value/values of the parameter. The Name property specifies the name of the Parameter. The AddParameter event notifies your application once a new Parameter object is added to the Parameters collection.

property Parameter.Value as Variant

Indicates the parameter's value(or values if a safe array is used).

Type	Description
Variant	A VARIANT expression that specifies the value of the current parameter, or a safe array of VARIANT if it contains multiple values.

The Value property specifies the value/values of the parameter. The Name property specifies the name of the Parameter. The UserData property holds any extra data associated with the parameter object.

The following is an example of a parameter:

RDATE;VALUE=**DATE**:19970304,19970504,19970704,19970904

The DATE specifies the value of the parameter.

Parameters object

The Parameters object holds a collection of Parameter objects. The Parameters property of the Property give access to the parameters of the property.

The following is an example of a parameter:

RDATE;**VALUE**=DATE:19970304,19970504,19970704,19970904

The VALUE indicates the parameter of the RDATE property.

The Parameters object supports the following properties and method:

Name	Description
Add	Adds a Parameter object to the collection and returns a reference to the newly created object.
Clear	Removes all objects in a collection.
Count	Returns the number of parameters in the collection.
Enumerate	Enumerates the parameters in the collection whose name matches the giving mask.
Item	Returns a specific Parameter of the Parameters collection, giving its name.
Remove	Removes a specific member from the Parameters collection, giving its name.

method Parameters.Add (Name as String, [Value as Variant])

Adds a Parameter object to the collection and returns a reference to the newly created object.

Type	Description
Name as String	A String expression that indicates the name of the parameter to be added.
Value as Variant	A VARIANT expression that specifies the value of the parameter to be added.

Return	Description
Parameter	A Parameter object being added.

The Add method adds a new parameter to the current property. The Name property specifies the name of the parameter. The Value property specifies the value of the parameter. The control fires the AddParameter event once a new parameter is added, if the control's FireEvents property is True.

The following is a simple example of an iCalendar format:

```
BEGIN:VCALENDAR
BEGIN:VEVENT
SUMMARY;LANGUAGE=en-US:Company Holiday Party
DATE:20010101
END:VEVENT
END:VCALENDAR
```

The LANGUAGE indicates the name of the parameter, of the property SUMMARY.

The following samples show how you can add parameters to a property.

VBA (MS Access, Excell...)

```
Set ICalendar1 = CreateObject("Exontrol.ICalendar.1")
With ICalendar1
  With .Content.Components.Add("VCALENDAR")
    With .Components.Add("VEVENT").Properties
      .Add("SUMMARY","Company Holiday Party").Parameters.Add
"LANGUAGE","en-US"
      .Add "DATE",#1/1/2001#
    End With
    End With
    Debug.Print( .Save )
End With
```

VB6

```
Set ICalendar1 = CreateObject("Exontrol.ICalendar.1")
With ICalendar1
    With .Content.Components.Add("VCALENDAR")
        With .Components.Add("VEVENT").Properties
            .Add("SUMMARY","Company Holiday Party").Parameters.Add
"LANGUAGE","en-US"
            .Add "DATE",#1/1/2001#
        End With
    End With
    Debug.Print( .Save )
End With
```

VB.NET

```
' Add 'exontrol.exicalendar.dll(ExICalendar.dll)' reference to your project.
Exicalendar1 = New exontrol.EXICALENDARLib.exicalendar()
With Exicalendar1
    With .Content.Components.Add("VCALENDAR")
        With .Components.Add("VEVENT").Properties
            .Add("SUMMARY","Company Holiday
Party").Parameters.Add("LANGUAGE","en-US")
            .Add("DATE",#1/1/2001#)
        End With
    End With
    Debug.Print( .Save() )
End With
```

VB.NET for /COM

```
AxICalendar1 = CreateObject("Exontrol.ICalendar.1")
With AxICalendar1
    With .Content.Components.Add("VCALENDAR")
```

```
        With .Components.Add("VEVENT").Properties
            .Add("SUMMARY","Company Holiday
Party").Parameters.Add("LANGUAGE","en-US")
            .Add("DATE",#1/1/2001#)
        End With
    End With
    Debug.Print( .Save() )
End With
```

C++

```
/*
    Includes the definition for CreateObject function like follows:

    #include <comdef.h>
    IUnknownPtr CreateObject( BSTR Object )
    {
        IUnknownPtr spResult;
        spResult.CreateInstance( Object );
        return spResult;
    };

*/
/*
    Copy and paste the following directives to your header file as
    it defines the namespace 'EXICALENDARLib' for the library: 'ICalendar 1.0 Type
Library'

    #import <ExICalendar.dll>
    using namespace EXICALENDARLib;
*/
EXICALENDARLib::IICalendarPtr spICalendar1 =
::CreateObject(L"Exontrol.ICalendar.1");
EXICALENDARLib::IComponentPtr var_Component = spICalendar1->GetContent()-
>GetComponents()->Add(L"VCALENDAR");
    EXICALENDARLib::IPropertiesPtr var_Properties = var_Component-
>GetComponents()->Add(L"VEVENT")->GetProperties();
```

```
	var_Properties->Add(L"SUMMARY","Company Holiday Party")-
>GetParameters()->Add(L"LANGUAGE","en-US");
	var_Properties->Add(L"DATE",COleDateTime(2001,1,1,0,00,00).operator DATE());
OutputDebugStringW( splCalendar1->Save() );
```

C++ Builder

```
Exicalendarlib_tlb::IICalendarPtr ICalendar1 =
Variant::CreateObject(L"Exontrol.ICalendar.1");
Exicalendarlib_tlb::IComponentPtr var_Component = ICalendar1->Content-
>Components->Add(L"VCALENDAR");
  Exicalendarlib_tlb::IPropertiesPtr var_Properties = var_Component->Components-
>Add(L"VEVENT")->Properties;
	var_Properties->Add(L"SUMMARY",TVariant("Company Holiday Party"))-
>Parameters->Add(L"LANGUAGE",TVariant("en-US"));
	var_Properties->Add(L"DATE",TVariant(TDateTime(2001,1,1).operator double()));
OutputDebugString( ICalendar1->Save() );
```

C#

```
// Add 'exontrol.exicalendar.dll(ExICalendar.dll)' reference to your project.
exontrol.EXICALENDARLib.exicalendar exicalendar1 = new
exontrol.EXICALENDARLib.exicalendar();
exontrol.EXICALENDARLib.Component var_Component =
exicalendar1.Content.Components.Add("VCALENDAR");
  exontrol.EXICALENDARLib.Properties var_Properties =
var_Component.Components.Add("VEVENT").Properties;
	var_Properties.Add("SUMMARY","Company Holiday
Party").Parameters.Add("LANGUAGE","en-US");

var_Properties.Add("DATE",Convert.ToDateTime("1/1/2001",System.Globalization.Cultu
US")));
System.Diagnostics.Debug.Print( exicalendar1.Save() );
```

JScript/JavaScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:D6C87100-38E2-4ABB-8AC2-4C0097AEE2D6"
id="ICalendar1"></OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Component = ICalendar1.Content.Components.Add("VCALENDAR");
        var var_Properties = var_Component.Components.Add("VEVENT").Properties;
            var_Properties.Add("SUMMARY","Company Holiday
Party").Parameters.Add("LANGUAGE","en-US");
            var_Properties.Add("DATE","1/1/2001");
    alert( ICalendar1.Save() );
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:D6C87100-38E2-4ABB-8AC2-4C0097AEE2D6"
id="ICalendar1"></OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With ICalendar1
        With .Content.Components.Add("VCALENDAR")
            With .Components.Add("VEVENT").Properties
                .Add("SUMMARY","Company Holiday Party").Parameters.Add
"LANGUAGE","en-US"
                .Add "DATE",#1/1/2001#
            End With
        End With
        alert( .Save )
    End With
End Function
```

```
</SCRIPT>
</BODY>
```

C# for /COM

```
EXICALENDARLib.ICalendar axICalendar1 = new EXICALENDARLib.ICalendar();
EXICALENDARLib.Component var_Component =
axICalendar1.Content.Components.Add("VCALENDAR");
    EXICALENDARLib.Properties var_Properties =
var_Component.Components.Add("VEVENT").Properties;
        var_Properties.Add("SUMMARY","Company Holiday
Party").Parameters.Add("LANGUAGE","en-US");

var_Properties.Add("DATE",Convert.ToDateTime("1/1/2001",System.Globalization.Cultu
US")));
System.Diagnostics.Debug.Print( axICalendar1.Save() );
```

X++ (Dynamics Ax 2009)

```
public void init()
{
    COM
com_Component,com_Component1,com_Components,com_Parameters,com_Propertie

    anytype
exicalendar1,var_Component,var_Component1,var_Components,var_Parameters,var_Pr

    ;

    super();

    // Add 'exicalendar.dll(ExICalendar.dll)' reference to your project.
    exicalendar1 = COM::createFromObject(new EXICALENDARLib.exicalendar());
com_exicalendar1 = exicalendar1;
    var_Component =
COM::createFromObject(com_exicalendar1.Content().Components()).Add("VCALENDAR
 com_Component = var_Component;
       var_Components = COM::createFromObject(com_Component.Components());
com_Components = var_Components;
       var_Component1 = COM::createFromObject(com_Components).Add("VEVENT");
com_Component1 = var_Component1;
       var_Properties = com_Component1.Properties(); com_Properties =
var_Properties;
          var_Property =
COM::createFromObject(com_Properties.Add("SUMMARY","Company Holiday
Party")); com_Property = var_Property;
          var_Parameters = COM::createFromObject(com_Property).Parameters();
com_Parameters = var_Parameters;
          com_Parameters.Add("LANGUAGE","en-US");

com_Properties.Add("DATE",COMVariant::createFromDate(str2Date("1/1/2001",213)));

   print( com_excalendar1.Save() );
}
```

Delphi 8 (.NET only)

```
AxICalendar1 :=
(ComObj.CreateComObject(ComObj.ProgIDToClassID('Exontrol.ICalendar.1')) as
EXICALENDARLib.ICalendar);
with AxICalendar1 do
begin
   with Content.Components.Add('VCALENDAR') do
   begin
      with Components.Add('VEVENT').Properties do
      begin
         Add('SUMMARY','Company Holiday Party').Parameters.Add('LANGUAGE','en-
US');
         Add('DATE','1/1/2001');
      end;
   end;
   OutputDebugString( Save() );
end
```

Delphi (standard)

```
ICalendar1 :=
(IUnknown(ComObj.CreateComObject(ComObj.ProgIDToClassID('Exontrol.ICalendar.1'))
 as EXICALENDARLib_TLB.ICalendar);
with ICalendar1 do
begin
    with Content.Components.Add('VCALENDAR') do
    begin
        with Components.Add('VEVENT').Properties do
        begin
            Add('SUMMARY','Company Holiday Party').Parameters.Add('LANGUAGE','en-
US');
            Add('DATE','1/1/2001');
        end;
    end;
    OutputDebugString( Save() );
end
```

VFP

```
thisform.ICalendar1 = CreateObject("Exontrol.ICalendar.1")
with thisform.ICalendar1
    with .Content.Components.Add("VCALENDAR")
        with .Components.Add("VEVENT").Properties
            .Add("SUMMARY","Company Holiday
Party").Parameters.Add("LANGUAGE","en-US")
            .Add("DATE",{^2001-1-1})
        endwith
    endwith
    DEBUGOUT( .Save )
endwith
```

dBASE Plus

```
local oICalendar,var_Component,var_Properties

oICalendar = new OleAutoClient("Exontrol.ICalendar.1")
```

```
var_Component = oICalendar.Content.Components.Add("VCALENDAR")
    var_Properties = var_Component.Components.Add("VEVENT").Properties
        var_Properties.Add("SUMMARY","Company Holiday
Party").Parameters.Add("LANGUAGE","en-US")
        var_Properties.Add("DATE","01/01/2001")
? oICalendar.Save()
```

XBasic (Alpha Five)

```
Dim oICalendar as P
Dim var_Component as P
Dim var_Properties as P

oICalendar = OLE.Create("Exontrol.ICalendar.1")

var_Component = oICalendar.Content.Components.Add("VCALENDAR")
    var_Properties = var_Component.Components.Add("VEVENT").Properties
        var_Properties.Add("SUMMARY","Company Holiday
Party").Parameters.Add("LANGUAGE","en-US")
        var_Properties.Add("DATE",{01/01/2001})
? oICalendar.Save()
```

Visual Objects

```
local var_Component as IComponent
local var_Properties as IProperties

oDCOCX_Exontrol1 := IICalendar{"Exontrol.ICalendar.1"}
var_Component := oDCOCX_Exontrol1:Content:Components:Add("VCALENDAR")
    var_Properties := var_Component:Components:Add("VEVENT"):Properties
        var_Properties:Add("SUMMARY","Company Holiday
Party"):Parameters:Add("LANGUAGE","en-US")
        var_Properties:Add("DATE",SToD("20010101"))
OutputDebugString(String2Psz( oDCOCX_Exontrol1:Save() ))
```

PowerBuilder

```
OleObject oICalendar,var_Component,var_Properties

oICalendar = CREATE OLEObject
oICalendar.ConnectToNewObject("Exontrol.ICalendar.1")

var_Component = oICalendar.Content.Components.Add("VCALENDAR")
    var_Properties = var_Component.Components.Add("VEVENT").Properties
        var_Properties.Add("SUMMARY","Company Holiday
Party").Parameters.Add("LANGUAGE","en-US")
        var_Properties.Add("DATE",2001-01-01)
MessageBox("Information",string( oICalendar.Save() ))
```

Visual DataFlex

```
Procedure OnCreate
    Forward Send OnCreate
    Variant oComICalendar1
    Get ComCreateObject "Exontrol.ICalendar.1" to oComICalendar1

    Variant voComponent
    Get ComContent to voComponent
    Handle hoComponent
    Get Create (RefClass(cComComponent)) to hoComponent
    Set pvComObject of hoComponent to voComponent
        Variant voComponents
        Get ComComponents of hoComponent to voComponents
        Handle hoComponents
        Get Create (RefClass(cComComponents)) to hoComponents
        Set pvComObject of hoComponents to voComponents
            Variant voComponent1
            Get ComAdd of hoComponents "VCALENDAR" to voComponent1
            Handle hoComponent1
            Get Create (RefClass(cComComponent)) to hoComponent1
            Set pvComObject of hoComponent1 to voComponent1
                Variant voComponents1
            Get ComComponents of hoComponent1 to voComponents1
            Handle hoComponents1
            Get Create (RefClass(cComComponents)) to hoComponents1
            Set pvComObject of hoComponents1 to voComponents1
                Variant voComponent2
                Get ComAdd of hoComponents1 "VEVENT" to voComponent2
                Handle hoComponent2
                Get Create (RefClass(cComComponent)) to hoComponent2
                Set pvComObject of hoComponent2 to voComponent2
                    Variant voProperties
                    Get ComProperties of hoComponent2 to voProperties
                    Handle hoProperties
                    Get Create (RefClass(cComProperties)) to hoProperties
                    Set pvComObject of hoProperties to voProperties
                        Variant voProperty
                        Get ComAdd of hoProperties "SUMMARY" "Company Holiday Party"
to voProperty
                        Handle hoProperty
                        Get Create (RefClass(cComProperty)) to hoProperty
                        Set pvComObject of hoProperty to voProperty
                            Variant voParameters
                            Get ComParameters of hoProperty to voParameters
                            Handle hoParameters
                            Get Create (RefClass(cComParameters)) to hoParameters
                            Set pvComObject of hoParameters to voParameters
                                Get ComAdd of hoParameters "LANGUAGE" "en-US" to Nothing
                            Send Destroy to hoParameters
                        Send Destroy to hoProperty
                        Get ComAdd of hoProperties "DATE" "1/1/2001" to Nothing
                    Send Destroy to hoProperties
                Send Destroy to hoComponent2
            Send Destroy to hoComponents1
        Send Destroy to hoComponent1
    Send Destroy to hoComponents
  Send Destroy to hoComponent
  Showln (ComSave(Self))
End_Procedure
```

XBase++

```
#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
   LOCAL oForm
   LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
   LOCAL oICalendar
   LOCAL oComponent
   LOCAL oProperties

   oForm := XbpDialog():new( AppDesktop() )
   oForm:drawingArea:clipChildren := .T.
   oForm:create( ,,{100,100}, {640,480},, .F. )
   oForm:close  := {|| PostAppEvent( xbeP_Quit )}

   oICalendar := XbpActiveXControl():new( oForm:drawingArea )
   oICalendar:CLSID  := "Exontrol.ICalendar.1" /*{D6C87100-38E2-4ABB-8AC2-
4C0097AEE2D6}*/
   oICalendar:create(,, {10,60},{610,370} )

      oComponent := oICalendar:Content():Components():Add("VCALENDAR")
         oProperties := oComponent:Components():Add("VEVENT"):Properties()
            oProperties:Add("SUMMARY","Company Holiday
Party"):Parameters():Add("LANGUAGE","en-US")
            oProperties:Add("DATE","01/01/2001")
      DevOut( oICalendar:Save() )

   oForm:Show()
   DO WHILE nEvent != xbeP_Quit
      nEvent := AppEvent( @mp1, @mp2, @oXbp )
      oXbp:handleEvent( nEvent, mp1, mp2 )
   ENDDO
RETURN
```

method Parameters.Clear ()

Removes all objects in a collection.

Type	Description

The Clear method clears the Parameters collection. The Remove method removes a parameter giving its name. The Item property accesses the Parameter giving its name. The Count property indicates the number of parameters in the collection. The Item / Count properties can be used to enumerate the Parameters collection as well as **for each** statement. The Enumerate method enumerates the parameters in the collection whose name matches the giving mask.

property Parameters.Count as Long

Returns the number of parameters in the collection.

Type	Description
Long	A Long expression that specifies the number of Parameter objects in the Parameters collection.

The Count property indicates the number of parameters in the collection. The Item property accesses the Parameter giving its name. The Item / Count properties can be used to enumerate the Parametes collection as well as **for each** statement. The Enumerate method enumerates the parameters in the collection whose name matches the giving mask. The Remove method removes a parameter from the Parameters collection. The Clear method clears the Parameters collection.

property Parameters.Enumerate (Mask as String) as Variant

Enumerates the parameters in the collection whose name matches the giving mask.

Type	Description
Mask as String	A String expression that specifies the mask of the parameters to be requested. The Mask parameter can include: • '?' for any single character • '*' for zero or more occurrences of any character • '#' for any digit character
Variant	A safe array of Parameter objects whose name matches the giving mask.

Use the Enumerate property to enumerate the parameters giving a mask. The Item / Count properties can be used to enumerate the Components collection as well as **for each** statement.

property Parameters.Item (Name as Variant) as Parameter

Returns a specific Parameter of the Parameters collection, giving its name.

Type	Description
Name as Variant	A String expression that specifies the name of the parameter to be requested
Parameter	A Parameter object being requested.

The Item property accesses the Parameter giving its name. The Count property indicates the number of parameters in the collection. The Item / Count properties can be used to enumerate the Parametes collection as well as **for each** statement. The Enumerate method enumerates the parameters in the collection whose name matches the giving mask. The Remove method removes a parameter from the Parameters collection. The Clear method clears the Parameters collection.

method Parameters.Remove (Name as Variant)

Removes a specific member from the Parameters collection, giving its name.

Type	Description
Name as Variant	A String expression that specifies the name of the parameter to be removed

The Remove method removes a parameter from the Parameters collection. The Clear method clears the Parameters collection. The Item property accesses the Parameter giving its name. The Count property indicates the number of parameters in the collection. The Item / Count properties can be used to enumerate the Parameters collection as well as **for each** statement. The Enumerate method enumerates the parameters in the collection whose name matches the giving mask.

Properties object

The Properties object holds a collection of Property objects. The Properties property of the Component object gives access to the property's properties collection.

The following is a simple example of an iCalendar format:

BEGIN:VCALENDAR
Version:2.0
PRODID:-//hacksw/handcal//NONSGML v1.0//EN
END:VCALENDAR

The Version, PRODID are properties of the VCALENDAR object.

The Properties collection supports the following properties and method:

Name	Description
Add	Adds a Property object to the collection and returns a reference to the newly created object.
Clear	Removes all objects in a collection.
Count	Returns the number of properties in the collection.
Enumerate	Enumerates the properties in the collection whose name matches the giving mask.
Item	Returns a specific Property of the Properties collection, giving its index or name.
Remove	Removes a specific member from the Properties collection, giving its index or name.

method Properties.Add (Name as String, [Value as Variant])

Adds a Property object to the collection and returns a reference to the newly created object.

Type	Description
Name as String	A String expression that specifies the name of the property to be added.
Value as Variant	A VARIANT expression that specifies the value of the property to be added.
Return	**Description**
Property	A Property object being created.

The Add method adds a Property object to the collection and returns a reference to the newly created object. The Name property specifies the name of the property. The Value property specifies the value of the property. The control fires the AddProperty event once a new property is added, if the control's FireEvents property is True. Use the UserData property to associate any extra-data to the property object. The Parameters property specifies the Parameters collection of the current property.

The following is a simple example of an iCalendar format:

BEGIN:VCALENDAR
Version:2.0
PRODID:-//hacksw/handcal//NONSGML v1.0//EN
END:VCALENDAR

The Version, PRODID are properties of the VCALENDAR object .

The following sample shows how you can add new properties to the component:

VBA (MS Access, Excell...)

```
Set ICalendar1 = CreateObject("Exontrol.ICalendar.1")
With ICalendar1
	With .Content.Components.Add("VCALENDAR").Properties
		.Add "Version","2.0"
		.Add "PRODID","-//hacksw/handcal//NONSGML v1.0//EN"
	End With
	Debug.Print( .Save )
End With
```

VB6

```
Set ICalendar1 = CreateObject("Exontrol.ICalendar.1")
With ICalendar1
   With .Content.Components.Add("VCALENDAR").Properties
      .Add "Version","2.0"
      .Add "PRODID","-//hacksw/handcal//NONSGML v1.0//EN"
   End With
   Debug.Print( .Save )
End With
```

VB.NET

```
' Add 'exontrol.exicalendar.dll(ExICalendar.dll)' reference to your project.
Exicalendar1 = New exontrol.EXICALENDARLib.exicalendar()
With Exicalendar1
   With .Content.Components.Add("VCALENDAR").Properties
      .Add("Version","2.0")
      .Add("PRODID","-//hacksw/handcal//NONSGML v1.0//EN")
   End With
   Debug.Print( .Save() )
End With
```

VB.NET for /COM

```
AxICalendar1 = CreateObject("Exontrol.ICalendar.1")
With AxICalendar1
   With .Content.Components.Add("VCALENDAR").Properties
      .Add("Version","2.0")
      .Add("PRODID","-//hacksw/handcal//NONSGML v1.0//EN")
   End With
   Debug.Print( .Save() )
End With
```

C++

```
/*
   Includes the definition for CreateObject function like follows:
```

```
	#include <comdef.h>
	IUnknownPtr CreateObject( BSTR Object )
	{
		IUnknownPtr spResult;
		spResult.CreateInstance( Object );
		return spResult;
	};

*/
/*
	Copy and paste the following directives to your header file as
	it defines the namespace 'EXICALENDARLib' for the library: 'ICalendar 1.0 Type
Library'

	#import <ExICalendar.dll>
	using namespace EXICALENDARLib;
*/
EXICALENDARLib::IICalendarPtr spICalendar1 =
::CreateObject(L"Exontrol.ICalendar.1");
EXICALENDARLib::IPropertiesPtr var_Properties = spICalendar1->GetContent()-
>GetComponents()->Add(L"VCALENDAR")->GetProperties();
   var_Properties->Add(L"Version","2.0");
   var_Properties->Add(L"PRODID","-//hacksw/handcal//NONSGML v1.0//EN");
OutputDebugStringW( spICalendar1->Save() );
```

C++ Builder

```
Exicalendarlib_tlb::IICalendarPtr ICalendar1 =
Variant::CreateObject(L"Exontrol.ICalendar.1");
Exicalendarlib_tlb::IPropertiesPtr var_Properties = ICalendar1->Content-
>Components->Add(L"VCALENDAR")->Properties;
   var_Properties->Add(L"Version",TVariant("2.0"));
   var_Properties->Add(L"PRODID",TVariant("-//hacksw/handcal//NONSGML
v1.0//EN"));
OutputDebugString( ICalendar1->Save() );
```

C#

```
// Add 'exontrol.exicalendar.dll(ExICalendar.dll)' reference to your project.
exontrol.EXICALENDARLib.exicalendar exicalendar1 = new
exontrol.EXICALENDARLib.exicalendar();
exontrol.EXICALENDARLib.Properties var_Properties =
exicalendar1.Content.Components.Add("VCALENDAR").Properties;
    var_Properties.Add("Version","2.0");
    var_Properties.Add("PRODID","-//hacksw/handcal//NONSGML v1.0//EN");
System.Diagnostics.Debug.Print( exicalendar1.Save() );
```

JScript/JavaScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:D6C87100-38E2-4ABB-8AC2-4C0097AEE2D6"
id="ICalendar1"></OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Properties =
ICalendar1.Content.Components.Add("VCALENDAR").Properties;
        var_Properties.Add("Version","2.0");
        var_Properties.Add("PRODID","-//hacksw/handcal//NONSGML v1.0//EN");
    alert( ICalendar1.Save() );
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:D6C87100-38E2-4ABB-8AC2-4C0097AEE2D6"
id="ICalendar1"></OBJECT>

<SCRIPT LANGUAGE="VBScript">
```

```
Function Init()
    With ICalendar1
        With .Content.Components.Add("VCALENDAR").Properties
            .Add "Version","2.0"
            .Add "PRODID","-//hacksw/handcal//NONSGML v1.0//EN"
        End With
        alert( .Save )
    End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
EXICALENDARLib.ICalendar axICalendar1 = new EXICALENDARLib.ICalendar();
EXICALENDARLib.Properties var_Properties =
axICalendar1.Content.Components.Add("VCALENDAR").Properties;
    var_Properties.Add("Version","2.0");
    var_Properties.Add("PRODID","-//hacksw/handcal//NONSGML v1.0//EN");
System.Diagnostics.Debug.Print( axICalendar1.Save() );
```

X++ (Dynamics Ax 2009)

```
public void init()
{
    COM com_Component,com_Properties,com_exicalendar1;
    anytype exicalendar1,var_Component,var_Properties;
    ;

    super();

    // Add 'exicalendar.dll(ExICalendar.dll)' reference to your project.
    exicalendar1 = COM::createFromObject(new EXICALENDARLib.exicalendar());
com_exicalendar1 = exicalendar1;
    var_Component =
COM::createFromObject(com_exicalendar1.Content().Components()).Add("VCALENDA
  com_Component = var_Component;
    var_Properties = com_Component.Properties(); com_Properties = var_Properties;
       com_Properties.Add("Version","2.0");
       com_Properties.Add("PRODID","-//hacksw/handcal//NONSGML v1.0//EN");
    print( com_exicalendar1.Save() );
}
```

Delphi 8 (.NET only)

```
AxICalendar1 :=
(ComObj.CreateComObject(ComObj.ProgIDToClassID('Exontrol.ICalendar.1')) as
EXICALENDARLib.ICalendar);
with AxICalendar1 do
begin
   with Content.Components.Add('VCALENDAR').Properties do
   begin
      Add('Version','2.0');
      Add('PRODID','-//hacksw/handcal//NONSGML v1.0//EN');
   end;
   OutputDebugString( Save() );
end
```

Delphi (standard)

```
ICalendar1 :=
(IUnknown(ComObj.CreateComObject(ComObj.ProgIDToClassID('Exontrol.ICalendar.1'))
 as EXICALENDARLib_TLB.ICalendar);
with ICalendar1 do
begin
   with Content.Components.Add('VCALENDAR').Properties do
   begin
      Add('Version','2.0');
      Add('PRODID','-//hacksw/handcal//NONSGML v1.0//EN');
   end;
   OutputDebugString( Save() );
end
```

VFP

```
thisform.ICalendar1 = CreateObject("Exontrol.ICalendar.1")
with thisform.ICalendar1
   with .Content.Components.Add("VCALENDAR").Properties
      .Add("Version","2.0")
      .Add("PRODID","-//hacksw/handcal//NONSGML v1.0//EN")
   endwith
   DEBUGOUT( .Save )
endwith
```

dBASE Plus

```
local oICalendar,var_Properties

oICalendar = new OleAutoClient("Exontrol.ICalendar.1")

var_Properties = oICalendar.Content.Components.Add("VCALENDAR").Properties
   var_Properties.Add("Version","2.0")
   var_Properties.Add("PRODID","-//hacksw/handcal//NONSGML v1.0//EN")
? oICalendar.Save()
```

XBasic (Alpha Five)

```
Dim oICalendar as P
Dim var_Properties as P

oICalendar = OLE.Create("Exontrol.ICalendar.1")

var_Properties = oICalendar.Content.Components.Add("VCALENDAR").Properties
   var_Properties.Add("Version","2.0")
   var_Properties.Add("PRODID","-//hacksw/handcal//NONSGML v1.0//EN")
? oICalendar.Save()
```

Visual Objects

```
local var_Properties as IProperties
```

```
oDCOCX_Exontrol1 := IICalendar{"Exontrol.ICalendar.1"}
var_Properties :=
oDCOCX_Exontrol1:Content:Components:Add("VCALENDAR"):Properties
   var_Properties:Add("Version","2.0")
   var_Properties:Add("PRODID","-//hacksw/handcal//NONSGML v1.0//EN")
OutputDebugString(String2Psz( oDCOCX_Exontrol1:Save() ))
```

PowerBuilder

```
OleObject oICalendar,var_Properties

oICalendar = CREATE OLEObject
oICalendar.ConnectToNewObject("Exontrol.ICalendar.1")

var_Properties = oICalendar.Content.Components.Add("VCALENDAR").Properties
   var_Properties.Add("Version","2.0")
   var_Properties.Add("PRODID","-//hacksw/handcal//NONSGML v1.0//EN")
MessageBox("Information",string( oICalendar.Save() ))
```

Visual DataFlex

```
Procedure OnCreate
   Forward Send OnCreate
   Variant oComICalendar1
   Get ComCreateObject "Exontrol.ICalendar.1" to oComICalendar1

   Variant voComponent
   Get ComContent to voComponent
   Handle hoComponent
   Get Create (RefClass(cComComponent)) to hoComponent
   Set pvComObject of hoComponent to voComponent
      Variant voComponents
      Get ComComponents of hoComponent to voComponents
      Handle hoComponents
      Get Create (RefClass(cComComponents)) to hoComponents
      Set pvComObject of hoComponents to voComponents
```

```
        Variant voComponent1
        Get ComAdd of hoComponents "VCALENDAR" to voComponent1
        Handle hoComponent1
        Get Create (RefClass(cComComponent)) to hoComponent1
        Set pvComObject of hoComponent1 to voComponent1
          Variant voProperties
          Get ComProperties of hoComponent1 to voProperties
          Handle hoProperties
          Get Create (RefClass(cComProperties)) to hoProperties
          Set pvComObject of hoProperties to voProperties
            Get ComAdd of hoProperties "Version" "2.0" to Nothing
            Get ComAdd of hoProperties "PRODID" "-//hacksw/handcal//NONSGML
v1.0//EN" to Nothing
          Send Destroy to hoProperties
        Send Destroy to hoComponent1
      Send Destroy to hoComponents
    Send Destroy to hoComponent
    Showln (ComSave(Self))
End_Procedure
```

XBase++

```
#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
  LOCAL oForm
  LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
  LOCAL oICalendar
  LOCAL oProperties

  oForm := XbpDialog():new( AppDesktop() )
  oForm:drawingArea:clipChildren := .T.
  oForm:create( ,,{100,100}, {640,480},, .F. )
  oForm:close  := {|| PostAppEvent( xbeP_Quit )}

  oICalendar := XbpActiveXControl():new( oForm:drawingArea )
```

```
    oICalendar:CLSID  := "Exontrol.ICalendar.1" /*{D6C87100-38E2-4ABB-8AC2-
4C0097AEE2D6}*/
    oICalendar:create(,, {10,60},{610,370} )

        oProperties :=
oICalendar:Content():Components():Add("VCALENDAR"):Properties()
            oProperties:Add("Version","2.0")
            oProperties:Add("PRODID","-//hacksw/handcal//NONSGML v1.0//EN")
        DevOut( oICalendar:Save() )

    oForm:Show()
    DO WHILE nEvent != xbeP_Quit
        nEvent := AppEvent( @mp1, @mp2, @oXbp )
        oXbp:handleEvent( nEvent, mp1, mp2 )
    ENDDO
RETURN
```

method Properties.Clear ()

Removes all objects in a collection.

Type	Description

The Clear method removes all properties of the Property object. The Clear method of the Component property empties the component, by removing the name, properties and components. The Remove method removes a property giving its index or name. The Item property accesses the Property giving its index / 0 - based or name. The Count property indicates the number of properties in the collection. The Item / Count properties can be used to enumerate the Properties collection as well as **for each** statement.

property Properties.Count as Long

Returns the number of properties in the collection.

Type	Description
Long	A Long expression that specifies the number of properties in the collection.

The Count property indicates the number of properties in the collection. The Item property accesses the Property giving its index / 0 - based or name. The Item / Count properties can be used to enumerate the Properties collection as well as **for each** statement. The Clear method removes all properties of the Property object. The Clear method of the Component property empties the component, by removing the name, properties and components. The Remove method removes a property giving its index or name.

The following code enumerates the properties of the root component:

```
Dim c As Variant
For Each c In ICalendar1.Root.Properties
   Debug.Print "Name " & c.Name
Next
```

and it's equivalent with the following snippet:

```
Dim i As Long
With ICalendar1.Root.Properties
   For i = 0 To .Count - 1
      Debug.Print .Item(i).Name
   Next
End With
```

property Properties.Enumerate (Mask as String) as Variant

Enumerates the properties in the collection whose name matches the giving mask.

Type	Description
Mask as String	A String expression that specifies the mask of the properties to be requested. The Mask parameter can include: • '?' for any single character • '*' for zero or more occurrences of any character • '#' for any digit character
Variant	A safe array of Property objects whose name matches the giving mask.

Use the Enumerate property to enumerate the properties giving a mask. The Item / Count properties can be used to enumerate the Properties collection as well as **for each** statement.

The following code enumerates the Properties of the root component, that starts with Ve:

```
Dim c As Variant
For Each c In ICalendar1.Root.Properties.Enumerate("Ve*")
   Debug.Print "Name " & c.Name
Next
```

property Properties.Item (Index as Variant) as Property

Returns a specific Property of the Properties collection, giving its index or name.

Type	Description
Index as Variant	A Long expression that specifies the index of the property to be requested, or a String expression that specifies the name of the property to be requested
Property	A Property being requested.

The Item property accesses the Property giving its index / 0 - based or name. The Count property indicates the number of properties in the collection. The Item / Count properties can be used to enumerate the Properties collection as well as **for each** statement. The Clear method removes all properties of the Property object. The Clear method of the Component property empties the component, by removing the name, properties and components. The Remove method removes a property giving its index or name.

The following code enumerates the properties of the root component:

```
Dim c As Variant
For Each c In ICalendar1.Root.Properties
    Debug.Print "Name " & c.Name
Next
```

and it's equivalent with the following snippet:

```
Dim i As Long
With ICalendar1.Root.Properties
    For i = 0 To .Count - 1
        Debug.Print .Item(i).Name
    Next
End With
```

method Properties.Remove (Index as Variant)

Removes a specific member from the Properties collection, giving its index or name.

Type	Description
Index as Variant	A Long expression that specifies the index of the property to be requested, or a String expression that specifies the name of the property to be removed

The Remove method removes a property giving its index or name. The Clear method of the Component property empties the component, by removing the name, properties and components. The Item property accesses the Property giving its index / 0 - based or name. The Count property indicates the number of properties in the collection. The Item / Count properties can be used to enumerate the Properties collection as well as **for each** statement.

Property object

The Property object holds information about a property within a component.

The following is a simple example of an iCalendar format:

```
BEGIN:VCALENDAR
Version:2.0
PRODID:-//hacksw/handcal//NONSGML v1.0//EN
END:VCALENDAR
```

The Version and PRODID represent properties of the VCALENDAR object.

The Property object supports the following properties and methods:

Name	Description
Component	Retrieves the parent component of the property.
GuessType	Guesses the property's type, from its value
Name	Indicates the property's name.
Parameters	Retrieves the parameters of the current property.
toICalendar	Gets the iCalendar representation of the property.
Type	Indicates the property's type.
UserData	Indicates any extra data associated with the property.
Value	Indicates the property's value(or values if a safe array is used).

property Property.Component as Component

Retrieves the parent component of the property.

Type	Description
Component	A Component object that specifies the parent component.

The Component property specifies the parent component of the property. The Name property specifies the name of the property. The Value property indicates the value of the property. The Type / GuessType property specifies the type of the property.

The following is a simple example of an iCalendar format:

BEGIN:VCALENDAR
Version:2.0
PRODID:-//hacksw/handcal//NONSGML v1.0//EN
END:VCALENDAR

The VCALENDAR object is the parent of the Version and PRODID properties.

property Property.GuessType as PropertyTypeEnum

Guesses the property's type, from its value

Type	Description
PropertyTypeEnum	A PropertyTypeEnum expression that specifies the type of the property, by getting it from Value property

The GuessType property guesses the property's type, from its value. The Value property specifies the value of the property. The Type property specifies the type of the property from the Value parameter of the property. The Name property specifies the name of the property. The valuesFromICalendar property extracts all values or specified value of the giving value/type in ICalendar format. The toICalendar property converts the giving value to a specified type.

The following samples show how Type and GuessType properties get the type of the Duration1 and Duration2 property .

In the following iCalendar format the Duration2 has a Value parameter that determines the type of the property, while the Duration1 has no Value parameter:

```
BEGIN:VCALENDAR
Duration1:P2DT12H
Duration2;VALUE=DURATION:P2DT12H
END:VCALENDAR
```

If running any of the following samples, you should get an output like:

```
Duration1 Guess 6
Duration1 Type 0
Duration2 Guess 6
Duration2 Type 6
```

VBA (MS Access, Excell...)

```
Set ICalendar1 = CreateObject("Exontrol.ICalendar.1")
With ICalendar1
  With .Content.Components.Add("VCALENDAR")
    .Properties.Add "Duration1",ICalendar1.toICalendar(2.5,6)
    With .Properties.Add("Duration2")
      .Value = 2.5
```

```
            .Type = 6
        End With
    End With
    With .Root.Properties.Item("Duration1")
        Debug.Print( .Name )
        Debug.Print( "Guess" )
        Debug.Print( .GuessType )
        Debug.Print( .Name )
        Debug.Print( "Type" )
        Debug.Print( .Type )
    End With
    With .Root.Properties.Item("Duration2")
        Debug.Print( .Name )
        Debug.Print( "Guess" )
        Debug.Print( .GuessType )
        Debug.Print( .Name )
        Debug.Print( "Type" )
        Debug.Print( .Type )
    End With
End With
```

VB6

```
Set ICalendar1 = CreateObject("Exontrol.ICalendar.1")
With ICalendar1
    With .Content.Components.Add("VCALENDAR")
        .Properties.Add "Duration1",ICalendar1.toICalendar(2.5,exPropertyTypeDuration)
        With .Properties.Add("Duration2")
            .Value = 2.5
            .Type = exPropertyTypeDuration
        End With
    End With
    With .Root.Properties.Item("Duration1")
        Debug.Print( .Name )
        Debug.Print( "Guess" )
        Debug.Print( .GuessType )
        Debug.Print( .Name )
        Debug.Print( "Type" )
        Debug.Print( .Type )
    End With
    With .Root.Properties.Item("Duration2")
        Debug.Print( .Name )
        Debug.Print( "Guess" )
        Debug.Print( .GuessType )
        Debug.Print( .Name )
        Debug.Print( "Type" )
        Debug.Print( .Type )
    End With
End With
```

VB.NET

```
' Add 'exontrol.exicalendar.dll(ExICalendar.dll)' reference to your project.
Exicalendar1 = New exontrol.EXICALENDARLib.exicalendar()
With Exicalendar1
    With .Content.Components.Add("VCALENDAR")

.Properties.Add("Duration1",Exicalendar1.get_toICalendar(2.5,exontrol.EXICALENDARLi

        With .Properties.Add("Duration2")
            .Value = 2.5
            .Type =
exontrol.EXICALENDARLib.PropertyTypeEnum.exPropertyTypeDuration
        End With
    End With
    With .Root.Properties.Item("Duration1")
        Debug.Print( .Name )
        Debug.Print( "Guess" )
        Debug.Print( .GuessType )
        Debug.Print( .Name )
        Debug.Print( "Type" )
        Debug.Print( .Type )
    End With
    With .Root.Properties.Item("Duration2")
```

```
        Debug.Print( .Name )
        Debug.Print( "Guess" )
        Debug.Print( .GuessType )
        Debug.Print( .Name )
        Debug.Print( "Type" )
        Debug.Print( .Type )
    End With
End With
```

VB.NET for /COM

```
AxICalendar1 = CreateObject("Exontrol.ICalendar.1")
With AxICalendar1
    With .Content.Components.Add("VCALENDAR")

.Properties.Add("Duration1",AxICalendar1.toICalendar(2.5,EXICALENDARLib.PropertyTy

        With .Properties.Add("Duration2")
            .Value = 2.5
            .Type = EXICALENDARLib.PropertyTypeEnum.exPropertyTypeDuration
        End With
    End With
    With .Root.Properties.Item("Duration1")
        Debug.Print( .Name )
        Debug.Print( "Guess" )
        Debug.Print( .GuessType )
        Debug.Print( .Name )
        Debug.Print( "Type" )
        Debug.Print( .Type )
    End With
    With .Root.Properties.Item("Duration2")
        Debug.Print( .Name )
        Debug.Print( "Guess" )
        Debug.Print( .GuessType )
        Debug.Print( .Name )
        Debug.Print( "Type" )
        Debug.Print( .Type )
```

```
   End With
End With
```

C++

```
/*
    Includes the definition for CreateObject function like follows:

    #include <comdef.h>
    IUnknownPtr CreateObject( BSTR Object )
    {
        IUnknownPtr spResult;
        spResult.CreateInstance( Object );
        return spResult;
    };

*/
/*
    Copy and paste the following directives to your header file as
    it defines the namespace 'EXICALENDARLib' for the library: 'ICalendar 1.0 Type
Library'

    #import <ExICalendar.dll>
    using namespace EXICALENDARLib;
*/
EXICALENDARLib::IICalendarPtr spICalendar1 =
::CreateObject(L"Exontrol.ICalendar.1");
EXICALENDARLib::IComponentPtr var_Component = spICalendar1->GetContent()-
>GetComponents()->Add(L"VCALENDAR");
   var_Component->GetProperties()->Add(L"Duration1",spICalendar1-
>GettoICalendar(double(2.5),EXICALENDARLib::exPropertyTypeDuration));
   EXICALENDARLib::IPropertyPtr var_Property = var_Component->GetProperties()-
>Add(L"Duration2",vtMissing);
      var_Property->PutValue(double(2.5));
      var_Property->PutType(EXICALENDARLib::exPropertyTypeDuration);
EXICALENDARLib::IPropertyPtr var_Property1 = spICalendar1->GetRoot()-
>GetProperties()->GetItem("Duration1");
```

```
   OutputDebugStringW( var_Property1->GetName() );
   OutputDebugStringW( L"Guess" );
   OutputDebugStringW( _bstr_t(var_Property1->GetGuessType()) );
   OutputDebugStringW( var_Property1->GetName() );
   OutputDebugStringW( L"Type" );
   OutputDebugStringW( _bstr_t(var_Property1->GetType()) );
EXICALENDARLib::IPropertyPtr var_Property2 = spICalendar1->GetRoot()-
>GetProperties()->GetItem("Duration2");
   OutputDebugStringW( var_Property2->GetName() );
   OutputDebugStringW( L"Guess" );
   OutputDebugStringW( _bstr_t(var_Property2->GetGuessType()) );
   OutputDebugStringW( var_Property2->GetName() );
   OutputDebugStringW( L"Type" );
   OutputDebugStringW( _bstr_t(var_Property2->GetType()) );
```

C++ Builder

```
Exicalendarlib_tlb::IICalendarPtr ICalendar1 =
Variant::CreateObject(L"Exontrol.ICalendar.1");
Exicalendarlib_tlb::IComponentPtr var_Component = ICalendar1->Content-
>Components->Add(L"VCALENDAR");
   var_Component->Properties->Add(L"Duration1",TVariant(ICalendar1-
>get_toICalendar(TVariant(2.5),Exicalendarlib_tlb::PropertyTypeEnum::exPropertyTypeD

   Exicalendarlib_tlb::IPropertyPtr var_Property = var_Component->Properties-
>Add(L"Duration2",TNoParam());
      var_Property->set_Value(TVariant(2.5));
      var_Property->Type =
Exicalendarlib_tlb::PropertyTypeEnum::exPropertyTypeDuration;
Exicalendarlib_tlb::IPropertyPtr var_Property1 = ICalendar1->Root->Properties-
>get_Item(TVariant("Duration1"));
   OutputDebugString( var_Property1->Name );
   OutputDebugString( L"Guess" );
   OutputDebugString( PChar(var_Property1->GuessType) );
   OutputDebugString( var_Property1->Name );
   OutputDebugString( L"Type" );
```

```
    OutputDebugString( PChar(var_Property1->Type) );
Exicalendarlib_tlb::IPropertyPtr var_Property2 = ICalendar1->Root->Properties-
>get_Item(TVariant("Duration2"));
    OutputDebugString( var_Property2->Name );
    OutputDebugString( L"Guess" );
    OutputDebugString( PChar(var_Property2->GuessType) );
    OutputDebugString( var_Property2->Name );
    OutputDebugString( L"Type" );
    OutputDebugString( PChar(var_Property2->Type) );
```

C#

```
// Add 'exontrol.exicalendar.dll(ExICalendar.dll)' reference to your project.
exontrol.EXICALENDARLib.exicalendar exicalendar1 = new
exontrol.EXICALENDARLib.exicalendar();
exontrol.EXICALENDARLib.Component var_Component =
exicalendar1.Content.Components.Add("VCALENDAR");

var_Component.Properties.Add("Duration1",exicalendar1.get_toICalendar(2.5,exontrol.

    exontrol.EXICALENDARLib.Property var_Property =
var_Component.Properties.Add("Duration2",null);
        var_Property.Value = 2.5;
        var_Property.Type =
exontrol.EXICALENDARLib.PropertyTypeEnum.exPropertyTypeDuration;
exontrol.EXICALENDARLib.Property var_Property1 =
exicalendar1.Root.Properties["Duration1"];
    System.Diagnostics.Debug.Print( var_Property1.Name );
    System.Diagnostics.Debug.Print( "Guess" );
    System.Diagnostics.Debug.Print( var_Property1.GuessType.ToString() );
    System.Diagnostics.Debug.Print( var_Property1.Name );
    System.Diagnostics.Debug.Print( "Type" );
    System.Diagnostics.Debug.Print( var_Property1.Type.ToString() );
exontrol.EXICALENDARLib.Property var_Property2 =
exicalendar1.Root.Properties["Duration2"];
    System.Diagnostics.Debug.Print( var_Property2.Name );
    System.Diagnostics.Debug.Print( "Guess" );
    System.Diagnostics.Debug.Print( var_Property2.GuessType.ToString() );
    System.Diagnostics.Debug.Print( var_Property2.Name );
    System.Diagnostics.Debug.Print( "Type" );
    System.Diagnostics.Debug.Print( var_Property2.Type.ToString() );
```

JScript/JavaScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:D6C87100-38E2-4ABB-8AC2-4C0097AEE2D6"
id="ICalendar1"></OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Component = ICalendar1.Content.Components.Add("VCALENDAR");
        var_Component.Properties.Add("Duration1",ICalendar1.toICalendar(2.5,6));
        var var_Property = var_Component.Properties.Add("Duration2",null);
            var_Property.Value = 2.5;
            var_Property.Type = 6;
    var var_Property1 = ICalendar1.Root.Properties.Item("Duration1");
        alert( var_Property1.Name );
        alert( "Guess" );
        alert( var_Property1.GuessType );
        alert( var_Property1.Name );
        alert( "Type" );
        alert( var_Property1.Type );
    var var_Property2 = ICalendar1.Root.Properties.Item("Duration2");
        alert( var_Property2.Name );
        alert( "Guess" );
        alert( var_Property2.GuessType );
        alert( var_Property2.Name );
        alert( "Type" );
        alert( var_Property2.Type );
}
</SCRIPT>
```

```
</BODY>
```

VBScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:D6C87100-38E2-4ABB-8AC2-4C0097AEE2D6"
id="ICalendar1"></OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With ICalendar1
        With .Content.Components.Add("VCALENDAR")
            .Properties.Add "Duration1",ICalendar1.toICalendar(2.5,6)
            With .Properties.Add("Duration2")
                .Value = 2.5
                .Type = 6
            End With
        End With
        With .Root.Properties.Item("Duration1")
            alert( .Name )
            alert( "Guess" )
            alert( .GuessType )
            alert( .Name )
            alert( "Type" )
            alert( .Type )
        End With
        With .Root.Properties.Item("Duration2")
            alert( .Name )
            alert( "Guess" )
            alert( .GuessType )
            alert( .Name )
            alert( "Type" )
            alert( .Type )
        End With
    End With
End Function
```

```
</SCRIPT>
</BODY>
```

C# for /COM

```
EXICALENDARLib.ICalendar axICalendar1 = new EXICALENDARLib.ICalendar();
EXICALENDARLib.Component var_Component =
axICalendar1.Content.Components.Add("VCALENDAR");

var_Component.Properties.Add("Duration1",axICalendar1.get_toICalendar(2.5,EXICALE

    EXICALENDARLib.Property var_Property =
var_Component.Properties.Add("Duration2",null);
        var_Property.Value = 2.5;
        var_Property.Type =
EXICALENDARLib.PropertyTypeEnum.exPropertyTypeDuration;
EXICALENDARLib.Property var_Property1 =
axICalendar1.Root.Properties["Duration1"];
    System.Diagnostics.Debug.Print( var_Property1.Name );
    System.Diagnostics.Debug.Print( "Guess" );
    System.Diagnostics.Debug.Print( var_Property1.GuessType.ToString() );
    System.Diagnostics.Debug.Print( var_Property1.Name );
    System.Diagnostics.Debug.Print( "Type" );
    System.Diagnostics.Debug.Print( var_Property1.Type.ToString() );
EXICALENDARLib.Property var_Property2 =
axICalendar1.Root.Properties["Duration2"];
    System.Diagnostics.Debug.Print( var_Property2.Name );
    System.Diagnostics.Debug.Print( "Guess" );
    System.Diagnostics.Debug.Print( var_Property2.GuessType.ToString() );
    System.Diagnostics.Debug.Print( var_Property2.Name );
    System.Diagnostics.Debug.Print( "Type" );
    System.Diagnostics.Debug.Print( var_Property2.Type.ToString() );
```

X++ (Dynamics Ax 2009)

```
public void init()
```

```
{
    COM
com_Component,com_Properties,com_Property,com_Property1,com_Property2,com_e

    anytype
exicalendar1,var_Component,var_Properties,var_Property,var_Property1,var_Property2,

    ;

    super();

    // Add 'exicalendar.dll(ExICalendar.dll)' reference to your project.
    exicalendar1 = COM::createFromObject(new EXICALENDARLib.exicalendar());
com_exicalendar1 = exicalendar1;
    var_Component =
COM::createFromObject(com_exicalendar1.Content().Components()).Add("VCALENDAR
 com_Component = var_Component;
        var_Properties = COM::createFromObject(com_Component.Properties());
com_Properties = var_Properties;

com_Properties.Add("Duration1",COMVariant::createFromStr(com_exicalendar1.toICale

        var_Properties = COM::createFromObject(com_Component.Properties());
com_Properties = var_Properties;
        var_Property = COM::createFromObject(com_Properties).Add("Duration2");
com_Property = var_Property;
            com_Property.Value(COMVariant::createFromReal(2.5));
            com_Property.Type(6/*exPropertyTypeDuration*/);
    var_Property1 =
COM::createFromObject(com_exicalendar1.Root().Properties()).Item("Duration1");
com_Property1 = var_Property1;
        print( com_Property1.Name() );
        print( "Guess" );
        print( com_Property1.GuessType() );
        print( com_Property1.Name() );
        print( "Type" );
        print( com_Property1.Type() );
```

```
   var_Property2 =
COM::createFromObject(com_exicalendar1.Root().Properties()).Item("Duration2");
com_Property2 = var_Property2;
      print( com_Property2.Name() );
      print( "Guess" );
      print( com_Property2.GuessType() );
      print( com_Property2.Name() );
      print( "Type" );
      print( com_Property2.Type() );
}
```

Delphi 8 (.NET only)

```
AxICalendar1 :=
(ComObj.CreateComObject(ComObj.ProgIDToClassID('Exontrol.ICalendar.1')) as
EXICALENDARLib.ICalendar);
with AxICalendar1 do
begin
   with Content.Components.Add('VCALENDAR') do
   begin

Properties.Add('Duration1',TObject(AxICalendar1.toICalendar[TObject(2.5),EXICALEND

      with Properties.Add('Duration2',Nil) do
      begin
         Value := TObject(2.5);
         Type := EXICALENDARLib.PropertyTypeEnum.exPropertyTypeDuration;
      end;
   end;
   with Root.Properties.Item['Duration1'] do
   begin
      OutputDebugString( Name );
      OutputDebugString( 'Guess' );
      OutputDebugString( GuessType );
      OutputDebugString( Name );
      OutputDebugString( 'Type' );
      OutputDebugString( Type );
```

```
    end;
    with Root.Properties.Item['Duration2'] do
    begin
        OutputDebugString( Name );
        OutputDebugString( 'Guess' );
        OutputDebugString( GuessType );
        OutputDebugString( Name );
        OutputDebugString( 'Type' );
        OutputDebugString( Type );
    end;
end
```

Delphi (standard)

```
ICalendar1 :=
(IUnknown(ComObj.CreateComObject(ComObj.ProgIDToClassID('Exontrol.ICalendar.1'))
 as EXICALENDARLib_TLB.ICalendar);
with ICalendar1 do
begin
    with Content.Components.Add('VCALENDAR') do
    begin

Properties.Add('Duration1',OleVariant(ICalendar1.toICalendar[OleVariant(2.5),EXICALEN

        with Properties.Add('Duration2',Null) do
        begin
            Value := OleVariant(2.5);
            Type := EXICALENDARLib_TLB.exPropertyTypeDuration;
        end;
    end;
    with Root.Properties.Item['Duration1'] do
    begin
        OutputDebugString( Name );
        OutputDebugString( 'Guess' );
        OutputDebugString( GuessType );
        OutputDebugString( Name );
        OutputDebugString( 'Type' );
        OutputDebugString( Type );
    end;
    with Root.Properties.Item['Duration2'] do
    begin
        OutputDebugString( Name );
        OutputDebugString( 'Guess' );
        OutputDebugString( GuessType );
        OutputDebugString( Name );
        OutputDebugString( 'Type' );
        OutputDebugString( Type );
    end;
end
```

VFP

```
thisform.ICalendar1 = CreateObject("Exontrol.ICalendar.1")
with thisform.ICalendar1
    with .Content.Components.Add("VCALENDAR")
        .Properties.Add("Duration1",thisform.ICalendar1.toICalendar(2.5,6))
        with .Properties.Add("Duration2")
            .Value = 2.5
            .Type = 6
        endwith
    endwith
    with .Root.Properties.Item("Duration1")
        DEBUGOUT( .Name )
        DEBUGOUT( "Guess" )
        DEBUGOUT( .GuessType )
        DEBUGOUT( .Name )
        DEBUGOUT( "Type" )
        DEBUGOUT( .Type )
    endwith
    with .Root.Properties.Item("Duration2")
        DEBUGOUT( .Name )
        DEBUGOUT( "Guess" )
        DEBUGOUT( .GuessType )
        DEBUGOUT( .Name )
        DEBUGOUT( "Type" )
        DEBUGOUT( .Type )
    endwith
endwith
```

dBASE Plus

```
local oICalendar,var_Component,var_Property,var_Property1,var_Property2

oICalendar = new OleAutoClient("Exontrol.ICalendar.1")

var_Component = oICalendar.Content.Components.Add("VCALENDAR")
    var_Component.Properties.Add("Duration1",oICalendar.toICalendar(2.5,6))
    var_Property = var_Component.Properties.Add("Duration2")
        var_Property.Value = 2.5
        var_Property.Type = 6
var_Property1 = oICalendar.Root.Properties.Item("Duration1")
    ? var_Property1.Name
    ? "Guess"
    ? Str(var_Property1.GuessType)
    ? var_Property1.Name
    ? "Type"
    ? Str(var_Property1.Type)
var_Property2 = oICalendar.Root.Properties.Item("Duration2")
    ? var_Property2.Name
    ? "Guess"
    ? Str(var_Property2.GuessType)
    ? var_Property2.Name
    ? "Type"
    ? Str(var_Property2.Type)
```

XBasic (Alpha Five)

```
Dim oICalendar as P
Dim var_Component as P
Dim var_Property as P
Dim var_Property1 as P
Dim var_Property2 as P

oICalendar = OLE.Create("Exontrol.ICalendar.1")

var_Component = oICalendar.Content.Components.Add("VCALENDAR")
   var_Component.Properties.Add("Duration1",oICalendar.toICalendar(2.5,6))
   var_Property = var_Component.Properties.Add("Duration2")
      var_Property.Value = 2.5
      var_Property.Type = 6
var_Property1 = oICalendar.Root.Properties.Item("Duration1")
   ? var_Property1.Name
   ? "Guess"
   ? var_Property1.GuessType
   ? var_Property1.Name
   ? "Type"
   ? var_Property1.Type
var_Property2 = oICalendar.Root.Properties.Item("Duration2")
   ? var_Property2.Name
   ? "Guess"
   ? var_Property2.GuessType
   ? var_Property2.Name
   ? "Type"
   ? var_Property2.Type
```

Visual Objects

```
local var_Component as IComponent
local var_Property,var_Property1,var_Property2 as IProperty

oDCOCX_Exontrol1 := IICalendar{"Exontrol.ICalendar.1"}
var_Component := oDCOCX_Exontrol1:Content:Components:Add("VCALENDAR")
   var_Component:Properties:Add("Duration1",oDCOCX_Exontrol1:
[toICalendar,2.5,exPropertyTypeDuration])
   var_Property := var_Component:Properties:Add("Duration2",nil)
      var_Property:Value := 2.5
      var_Property:Type := exPropertyTypeDuration
```

```
var_Property1 := oDCOCX_Exontrol1:Root:Properties:[Item,"Duration1"]
   OutputDebugString(String2Psz( var_Property1:Name ))
   OutputDebugString(String2Psz( "Guess" ))
   OutputDebugString(String2Psz( AsString(var_Property1:GuessType) ))
   OutputDebugString(String2Psz( var_Property1:Name ))
   OutputDebugString(String2Psz( "Type" ))
   OutputDebugString(String2Psz( AsString(var_Property1:Type) ))
var_Property2 := oDCOCX_Exontrol1:Root:Properties:[Item,"Duration2"]
   OutputDebugString(String2Psz( var_Property2:Name ))
   OutputDebugString(String2Psz( "Guess" ))
   OutputDebugString(String2Psz( AsString(var_Property2:GuessType) ))
   OutputDebugString(String2Psz( var_Property2:Name ))
   OutputDebugString(String2Psz( "Type" ))
   OutputDebugString(String2Psz( AsString(var_Property2:Type) ))
```

PowerBuilder

```
OleObject oICalendar,var_Component,var_Property,var_Property1,var_Property2

oICalendar = CREATE OLEObject
oICalendar.ConnectToNewObject("Exontrol.ICalendar.1")

var_Component = oICalendar.Content.Components.Add("VCALENDAR")
   var_Component.Properties.Add("Duration1",oICalendar.toICalendar(2.5,6))
   var_Property = var_Component.Properties.Add("Duration2")
      var_Property.Value = 2.5
      var_Property.Type = 6
var_Property1 = oICalendar.Root.Properties.Item("Duration1")
   MessageBox("Information",string( var_Property1.Name ))
   MessageBox("Information",string( "Guess" ))
   MessageBox("Information",string( String(var_Property1.GuessType) ))
   MessageBox("Information",string( var_Property1.Name ))
   MessageBox("Information",string( "Type" ))
   MessageBox("Information",string( String(var_Property1.Type) ))
var_Property2 = oICalendar.Root.Properties.Item("Duration2")
   MessageBox("Information",string( var_Property2.Name ))
```

```
    MessageBox("Information",string( "Guess" ))
    MessageBox("Information",string( String(var_Property2.GuessType) ))
    MessageBox("Information",string( var_Property2.Name ))
    MessageBox("Information",string( "Type" ))
    MessageBox("Information",string( String(var_Property2.Type) ))
```

Visual DataFlex

```
Procedure OnCreate
    Forward Send OnCreate
    Variant oComICalendar1
    Get ComCreateObject "Exontrol.ICalendar.1" to oComICalendar1

    Variant voComponent
    Get ComContent to voComponent
    Handle hoComponent
    Get Create (RefClass(cComComponent)) to hoComponent
    Set pvComObject of hoComponent to voComponent
        Variant voComponents
        Get ComComponents of hoComponent to voComponents
        Handle hoComponents
        Get Create (RefClass(cComComponents)) to hoComponents
        Set pvComObject of hoComponents to voComponents
            Variant voComponent1
            Get ComAdd of hoComponents "VCALENDAR" to voComponent1
            Handle hoComponent1
            Get Create (RefClass(cComComponent)) to hoComponent1
            Set pvComObject of hoComponent1 to voComponent1
                Variant voProperties
                Get ComProperties of hoComponent1 to voProperties
                Handle hoProperties
                Get Create (RefClass(cComProperties)) to hoProperties
                Set pvComObject of hoProperties to voProperties
                    Variant vValue
                        Get ComtoICalendar 2.5 OLEexPropertyTypeDuration to vValue
                    Get ComAdd of hoProperties "Duration1" vValue to Nothing
```

```
			Send Destroy to hoProperties
			Variant voProperties1
			Get ComProperties of hoComponent1 to voProperties1
			Handle hoProperties1
			Get Create (RefClass(cComProperties)) to hoProperties1
			Set pvComObject of hoProperties1 to voProperties1
				Variant voProperty
				Get ComAdd of hoProperties1 "Duration2" Nothing to voProperty
				Handle hoProperty
				Get Create (RefClass(cComProperty)) to hoProperty
				Set pvComObject of hoProperty to voProperty
					Set ComValue of hoProperty to 2.5
					Set ComType of hoProperty to OLEexPropertyTypeDuration
				Send Destroy to hoProperty
			Send Destroy to hoProperties1
		Send Destroy to hoComponent1
	Send Destroy to hoComponents
Send Destroy to hoComponent
Variant voComponent2
Get ComRoot to voComponent2
Handle hoComponent2
Get Create (RefClass(cComComponent)) to hoComponent2
Set pvComObject of hoComponent2 to voComponent2
	Variant voProperties2
	Get ComProperties of hoComponent2 to voProperties2
	Handle hoProperties2
	Get Create (RefClass(cComProperties)) to hoProperties2
	Set pvComObject of hoProperties2 to voProperties2
		Variant voProperty1
		Get ComItem of hoProperties2 "Duration1" to voProperty1
		Handle hoProperty1
		Get Create (RefClass(cComProperty)) to hoProperty1
		Set pvComObject of hoProperty1 to voProperty1
			Variant voProperty2
			Get ComItem of hoProperty1 "Duration1" to voProperty2
			Handle hoProperty2
			Get Create (RefClass(cComProperty)) to hoProperty2
```

```
            Set pvComObject of hoProperty2 to voProperty2
                ShowIn (ComName(hoProperty2)) "Guess"
(ComGuessType(hoProperty2))
                ShowIn (ComName(hoProperty2)) "Type" (ComType(hoProperty2))
            Send Destroy to hoProperty2
        Send Destroy to hoProperty1
    Send Destroy to hoProperties2
Send Destroy to hoComponent2
Variant voComponent3
Get ComRoot to voComponent3
Handle hoComponent3
Get Create (RefClass(cComComponent)) to hoComponent3
Set pvComObject of hoComponent3 to voComponent3
    Variant voProperties3
    Get ComProperties of hoComponent3 to voProperties3
    Handle hoProperties3
    Get Create (RefClass(cComProperties)) to hoProperties3
    Set pvComObject of hoProperties3 to voProperties3
        Variant voProperty3
        Get ComItem of hoProperties3 "Duration2" to voProperty3
        Handle hoProperty3
        Get Create (RefClass(cComProperty)) to hoProperty3
        Set pvComObject of hoProperty3 to voProperty3
            Variant voProperty4
            Get ComItem of hoProperty3 "Duration2" to voProperty4
            Handle hoProperty4
            Get Create (RefClass(cComProperty)) to hoProperty4
            Set pvComObject of hoProperty4 to voProperty4
                ShowIn (ComName(hoProperty4)) "Guess"
(ComGuessType(hoProperty4))
                ShowIn (ComName(hoProperty4)) "Type" (ComType(hoProperty4))
            Send Destroy to hoProperty4
        Send Destroy to hoProperty3
    Send Destroy to hoProperties3
Send Destroy to hoComponent3
End_Procedure
```

XBase++

```
#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
   LOCAL oForm
   LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
   LOCAL oICalendar
   LOCAL oComponent
   LOCAL oProperty,oProperty1,oProperty2

   oForm := XbpDialog():new( AppDesktop() )
   oForm:drawingArea:clipChildren := .T.
   oForm:create( ,,{100,100}, {640,480},, .F. )
   oForm:close  := {|| PostAppEvent( xbeP_Quit )}

   oICalendar := XbpActiveXControl():new( oForm:drawingArea )
   oICalendar:CLSID  := "Exontrol.ICalendar.1" /*{D6C87100-38E2-4ABB-8AC2-
4C0097AEE2D6}*/
   oICalendar:create(,, {10,60},{610,370} )

      oComponent := oICalendar:Content():Components():Add("VCALENDAR")

oComponent:Properties():Add("Duration1",oICalendar:toICalendar(2.5,6/*exPropertyTy

         oProperty := oComponent:Properties():Add("Duration2")
            oProperty:Value := 2.5
            oProperty:Type := 6/*exPropertyTypeDuration*/
      oProperty1 := oICalendar:Root():Properties:Item("Duration1")
         DevOut( oProperty1:Name() )
         DevOut( "Guess" )
         DevOut( Transform(oProperty1:GuessType(),"") )
         DevOut( oProperty1:Name() )
         DevOut( "Type" )
         DevOut( Transform(oProperty1:Type(),"") )
      oProperty2 := oICalendar:Root():Properties:Item("Duration2")
```

```
            DevOut( oProperty2:Name() )
            DevOut( "Guess" )
            DevOut( Transform(oProperty2:GuessType(),"") )
            DevOut( oProperty2:Name() )
            DevOut( "Type" )
            DevOut( Transform(oProperty2:Type(),"") )

    oForm:Show()
    DO WHILE nEvent != xbeP_Quit
      nEvent := AppEvent( @mp1, @mp2, @oXbp )
      oXbp:handleEvent( nEvent, mp1, mp2 )
    ENDDO
RETURN
```

property Property.Name as String

Indicates the property's name.

Type	Description
String	A String expression that specifies the name of the property.

The Name property indicates the name of the property. The Name parameter of the Add property indicates the name of the property to be added. The Value property specifies the value of the property. The GuessType property guesses the property's type, from its value. The Type property specifies the type of the property from the Value parameter of the property. The Parameters property specifies the property's Parameters collection.

The following is a simple example of an iCalendar format:

BEGIN:VCALENDAR
BEGIN:VEVENT
SUMMARY;LANGUAGE=en-US:Company Holiday Party
DATE:20010101
END:VEVENT
END:VCALENDAR

The SUMMARY and DATE are the name of the properties of the VCALENDAR object.

property Property.Parameters as Parameters

Retrieves the parameters of the current property.

Type	Description
Parameters	A Parameters collection that holds Parameter objects.

The Parameters property gives access to the property's parameters. If property has no parameters, the Parameters collection is empty. The Add method adds a new parameter to the current property.

The following is an example of a parameter:

RDATE;**VALUE**=DATE:19970304,19970504,19970704,19970904

The VALUE indicates the parameter of the RDATE property.

property Property.toICalendar as String

Gets the iCalendar representation of the property.

Type	Description
String	A String expression that specifies the iCalendar format of the current property

The toICalendar property retrieves the iCalendar representation of the property. You can use the Load / LoadFile / LoadFileFromUnicode methods load iCalendar format, while Save / SaveFile / SaveFileAsUnicode methods save the control's content as iCalendar format.

property Property.Type as PropertyTypeEnum

Indicates the property's type.

Type	Description
PropertyTypeEnum	A PropertyTypeEnum expression that specifies the type of the property, by getting it from Value parameter of the property.

The Type property specifies the type of the property from the Value parameter of the property. The Parameters property specifies the property's Parameters collection. The GuessType property guesses the property's type, from its value. The Value property specifies the value of the property. The Name property specifies the name of the property. The valuesFromICalendar property extracts all values or specified value of the giving value/type in ICalendar format. The toICalendar property converts the giving value to a specified type.

The following samples show how Type and GuessType properties get the type of the Duration1 and Duration2 property .

In the following iCalendar format the Duration2 has a Value parameter that determines the type of the property, while the Duration1 has no Value parameter:

```
BEGIN:VCALENDAR
Duration1:P2DT12H
Duration2;VALUE=DURATION:P2DT12H
END:VCALENDAR
```

If running any of the following samples, you should get an output like:

```
Duration1 Guess 6
Duration1 Type 0
Duration2 Guess 6
Duration2 Type 6
```

VBA (MS Access, Excell...)

```
Set ICalendar1 = CreateObject("Exontrol.ICalendar.1")
With ICalendar1
    With .Content.Components.Add("VCALENDAR")
        .Properties.Add "Duration1",ICalendar1.toICalendar(2.5,6)
        With .Properties.Add("Duration2")
        .Value = 2.5
        .Type = 6
      End With
    End With
    With .Root.Properties.Item("Duration1")
      Debug.Print( .Name )
      Debug.Print( "Guess" )
      Debug.Print( .GuessType )
      Debug.Print( .Name )
      Debug.Print( "Type" )
      Debug.Print( .Type )
    End With
    With .Root.Properties.Item("Duration2")
      Debug.Print( .Name )
      Debug.Print( "Guess" )
      Debug.Print( .GuessType )
      Debug.Print( .Name )
      Debug.Print( "Type" )
      Debug.Print( .Type )
    End With
End With
```

VB6

```
Set ICalendar1 = CreateObject("Exontrol.ICalendar.1")
With ICalendar1
    With .Content.Components.Add("VCALENDAR")
      .Properties.Add "Duration1",ICalendar1.toICalendar(2.5,exPropertyTypeDuration)
      With .Properties.Add("Duration2")
        .Value = 2.5
        .Type = exPropertyTypeDuration
      End With
    End With
    With .Root.Properties.Item("Duration1")
      Debug.Print( .Name )
      Debug.Print( "Guess" )
      Debug.Print( .GuessType )
```

```
        Debug.Print( .Name )
        Debug.Print( "Type" )
        Debug.Print( .Type )
    End With
    With .Root.Properties.Item("Duration2")
        Debug.Print( .Name )
        Debug.Print( "Guess" )
        Debug.Print( .GuessType )
        Debug.Print( .Name )
        Debug.Print( "Type" )
        Debug.Print( .Type )
    End With
End With
```

VB.NET

```
' Add 'exontrol.exicalendar.dll(ExICalendar.dll)' reference to your project.
Exicalendar1 = New exontrol.EXICALENDARLib.exicalendar()
With Exicalendar1
    With .Content.Components.Add("VCALENDAR")

.Properties.Add("Duration1",Exicalendar1.get_toICalendar(2.5,exontrol.EXICALENDARLi

        With .Properties.Add("Duration2")
            .Value = 2.5
            .Type =
exontrol.EXICALENDARLib.PropertyTypeEnum.exPropertyTypeDuration
        End With
    End With
    With .Root.Properties.Item("Duration1")
        Debug.Print( .Name )
        Debug.Print( "Guess" )
        Debug.Print( .GuessType )
        Debug.Print( .Name )
        Debug.Print( "Type" )
        Debug.Print( .Type )
    End With
```

```
	With .Root.Properties.Item("Duration2")
		Debug.Print( .Name )
		Debug.Print( "Guess" )
		Debug.Print( .GuessType )
		Debug.Print( .Name )
		Debug.Print( "Type" )
		Debug.Print( .Type )
	End With
End With
```

VB.NET for /COM

```
AxICalendar1 = CreateObject("Exontrol.ICalendar.1")
With AxICalendar1
	With .Content.Components.Add("VCALENDAR")

.Properties.Add("Duration1",AxICalendar1.toICalendar(2.5,EXICALENDARLib.PropertyTy

		With .Properties.Add("Duration2")
			.Value = 2.5
			.Type = EXICALENDARLib.PropertyTypeEnum.exPropertyTypeDuration
		End With
	End With
	With .Root.Properties.Item("Duration1")
		Debug.Print( .Name )
		Debug.Print( "Guess" )
		Debug.Print( .GuessType )
		Debug.Print( .Name )
		Debug.Print( "Type" )
		Debug.Print( .Type )
	End With
	With .Root.Properties.Item("Duration2")
		Debug.Print( .Name )
		Debug.Print( "Guess" )
		Debug.Print( .GuessType )
		Debug.Print( .Name )
		Debug.Print( "Type" )
```

```
        Debug.Print( .Type )
    End With
End With
```

C++

```
/*
    Includes the definition for CreateObject function like follows:

    #include <comdef.h>
    IUnknownPtr CreateObject( BSTR Object )
    {
        IUnknownPtr spResult;
        spResult.CreateInstance( Object );
        return spResult;
    };

*/
/*
    Copy and paste the following directives to your header file as
    it defines the namespace 'EXICALENDARLib' for the library: 'ICalendar 1.0 Type
Library'

    #import <ExICalendar.dll>
    using namespace EXICALENDARLib;
*/
EXICALENDARLib::IICalendarPtr spICalendar1 =
::CreateObject(L"Exontrol.ICalendar.1");
EXICALENDARLib::IComponentPtr var_Component = spICalendar1->GetContent()-
>GetComponents()->Add(L"VCALENDAR");
    var_Component->GetProperties()->Add(L"Duration1",spICalendar1-
>GettoICalendar(double(2.5),EXICALENDARLib::exPropertyTypeDuration));
    EXICALENDARLib::IPropertyPtr var_Property = var_Component->GetProperties()-
>Add(L"Duration2",vtMissing);
        var_Property->PutValue(double(2.5));
        var_Property->PutType(EXICALENDARLib::exPropertyTypeDuration);
EXICALENDARLib::IPropertyPtr var_Property1 = spICalendar1->GetRoot()-
```

```
>GetProperties()->GetItem("Duration1");
   OutputDebugStringW( var_Property1->GetName() );
   OutputDebugStringW( L"Guess" );
   OutputDebugStringW( _bstr_t(var_Property1->GetGuessType()) );
   OutputDebugStringW( var_Property1->GetName() );
   OutputDebugStringW( L"Type" );
   OutputDebugStringW( _bstr_t(var_Property1->GetType()) );
EXICALENDARLib::IPropertyPtr var_Property2 = spICalendar1->GetRoot()-
>GetProperties()->GetItem("Duration2");
   OutputDebugStringW( var_Property2->GetName() );
   OutputDebugStringW( L"Guess" );
   OutputDebugStringW( _bstr_t(var_Property2->GetGuessType()) );
   OutputDebugStringW( var_Property2->GetName() );
   OutputDebugStringW( L"Type" );
   OutputDebugStringW( _bstr_t(var_Property2->GetType()) );
```

C++ Builder

```
Exicalendarlib_tlb::IICalendarPtr ICalendar1 =
Variant::CreateObject(L"Exontrol.ICalendar.1");
Exicalendarlib_tlb::IComponentPtr var_Component = ICalendar1->Content-
>Components->Add(L"VCALENDAR");
   var_Component->Properties->Add(L"Duration1",TVariant(ICalendar1-
>get_toICalendar(TVariant(2.5),Exicalendarlib_tlb::PropertyTypeEnum::exPropertyTypeD

   Exicalendarlib_tlb::IPropertyPtr var_Property = var_Component->Properties-
>Add(L"Duration2",TNoParam());
      var_Property->set_Value(TVariant(2.5));
      var_Property->Type =
Exicalendarlib_tlb::PropertyTypeEnum::exPropertyTypeDuration;
Exicalendarlib_tlb::IPropertyPtr var_Property1 = ICalendar1->Root->Properties-
>get_Item(TVariant("Duration1"));
   OutputDebugString( var_Property1->Name );
   OutputDebugString( L"Guess" );
   OutputDebugString( PChar(var_Property1->GuessType) );
   OutputDebugString( var_Property1->Name );
```

```
    OutputDebugString( L"Type" );
    OutputDebugString( PChar(var_Property1->Type) );
Exicalendarlib_tlb::IPropertyPtr var_Property2 = ICalendar1->Root->Properties-
>get_Item(TVariant("Duration2"));
    OutputDebugString( var_Property2->Name );
    OutputDebugString( L"Guess" );
    OutputDebugString( PChar(var_Property2->GuessType) );
    OutputDebugString( var_Property2->Name );
    OutputDebugString( L"Type" );
    OutputDebugString( PChar(var_Property2->Type) );
```

C#

```
// Add 'exontrol.exicalendar.dll(ExICalendar.dll)' reference to your project.
exontrol.EXICALENDARLib.exicalendar exicalendar1 = new
exontrol.EXICALENDARLib.exicalendar();
exontrol.EXICALENDARLib.Component var_Component =
exicalendar1.Content.Components.Add("VCALENDAR");

var_Component.Properties.Add("Duration1",exicalendar1.get_toICalendar(2.5,exontrol.

    exontrol.EXICALENDARLib.Property var_Property =
var_Component.Properties.Add("Duration2",null);
        var_Property.Value = 2.5;
        var_Property.Type =
exontrol.EXICALENDARLib.PropertyTypeEnum.exPropertyTypeDuration;
exontrol.EXICALENDARLib.Property var_Property1 =
exicalendar1.Root.Properties["Duration1"];
    System.Diagnostics.Debug.Print( var_Property1.Name );
    System.Diagnostics.Debug.Print( "Guess" );
    System.Diagnostics.Debug.Print( var_Property1.GuessType.ToString() );
    System.Diagnostics.Debug.Print( var_Property1.Name );
    System.Diagnostics.Debug.Print( "Type" );
    System.Diagnostics.Debug.Print( var_Property1.Type.ToString() );
exontrol.EXICALENDARLib.Property var_Property2 =
exicalendar1.Root.Properties["Duration2"];
```

```
    System.Diagnostics.Debug.Print( var_Property2.Name );
    System.Diagnostics.Debug.Print( "Guess" );
    System.Diagnostics.Debug.Print( var_Property2.GuessType.ToString() );
    System.Diagnostics.Debug.Print( var_Property2.Name );
    System.Diagnostics.Debug.Print( "Type" );
    System.Diagnostics.Debug.Print( var_Property2.Type.ToString() );
```

JScript/JavaScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:D6C87100-38E2-4ABB-8AC2-4C0097AEE2D6"
id="ICalendar1"></OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Component = ICalendar1.Content.Components.Add("VCALENDAR");
        var_Component.Properties.Add("Duration1",ICalendar1.toICalendar(2.5,6));
        var var_Property = var_Component.Properties.Add("Duration2",null);
            var_Property.Value = 2.5;
            var_Property.Type = 6;
    var var_Property1 = ICalendar1.Root.Properties.Item("Duration1");
        alert( var_Property1.Name );
        alert( "Guess" );
        alert( var_Property1.GuessType );
        alert( var_Property1.Name );
        alert( "Type" );
        alert( var_Property1.Type );
    var var_Property2 = ICalendar1.Root.Properties.Item("Duration2");
        alert( var_Property2.Name );
        alert( "Guess" );
        alert( var_Property2.GuessType );
        alert( var_Property2.Name );
        alert( "Type" );
        alert( var_Property2.Type );
}
```

```
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:D6C87100-38E2-4ABB-8AC2-4C0097AEE2D6"
id="ICalendar1"></OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With ICalendar1
        With .Content.Components.Add("VCALENDAR")
            .Properties.Add "Duration1",ICalendar1.toICalendar(2.5,6)
            With .Properties.Add("Duration2")
                .Value = 2.5
                .Type = 6
            End With
        End With
        With .Root.Properties.Item("Duration1")
            alert( .Name )
            alert( "Guess" )
            alert( .GuessType )
            alert( .Name )
            alert( "Type" )
            alert( .Type )
        End With
        With .Root.Properties.Item("Duration2")
            alert( .Name )
            alert( "Guess" )
            alert( .GuessType )
            alert( .Name )
            alert( "Type" )
            alert( .Type )
        End With
    End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
EXICALENDARLib.ICalendar axICalendar1 = new EXICALENDARLib.ICalendar();
EXICALENDARLib.Component var_Component =
axICalendar1.Content.Components.Add("VCALENDAR");

var_Component.Properties.Add("Duration1",axICalendar1.get_toICalendar(2.5,EXICALE

    EXICALENDARLib.Property var_Property =
var_Component.Properties.Add("Duration2",null);
        var_Property.Value = 2.5;
        var_Property.Type =
EXICALENDARLib.PropertyTypeEnum.exPropertyTypeDuration;
EXICALENDARLib.Property var_Property1 =
axICalendar1.Root.Properties["Duration1"];
    System.Diagnostics.Debug.Print( var_Property1.Name );
    System.Diagnostics.Debug.Print( "Guess" );
    System.Diagnostics.Debug.Print( var_Property1.GuessType.ToString() );
    System.Diagnostics.Debug.Print( var_Property1.Name );
    System.Diagnostics.Debug.Print( "Type" );
    System.Diagnostics.Debug.Print( var_Property1.Type.ToString() );
EXICALENDARLib.Property var_Property2 =
axICalendar1.Root.Properties["Duration2"];
    System.Diagnostics.Debug.Print( var_Property2.Name );
    System.Diagnostics.Debug.Print( "Guess" );
    System.Diagnostics.Debug.Print( var_Property2.GuessType.ToString() );
    System.Diagnostics.Debug.Print( var_Property2.Name );
    System.Diagnostics.Debug.Print( "Type" );
    System.Diagnostics.Debug.Print( var_Property2.Type.ToString() );
```

X++ (Dynamics Ax 2009)

```
public void init()
{
    COM
com_Component,com_Properties,com_Property,com_Property1,com_Property2,com_e

    anytype
exicalendar1,var_Component,var_Properties,var_Property,var_Property1,var_Property2,

    ;

    super();

    // Add 'exicalendar.dll(ExICalendar.dll)' reference to your project.
    exicalendar1 = COM::createFromObject(new EXICALENDARLib.exicalendar());
com_exicalendar1 = exicalendar1;
    var_Component =
COM::createFromObject(com_exicalendar1.Content().Components()).Add("VCALENDAR
 com_Component = var_Component;
        var_Properties = COM::createFromObject(com_Component.Properties());
com_Properties = var_Properties;

com_Properties.Add("Duration1",COMVariant::createFromStr(com_exicalendar1.toICale

        var_Properties = COM::createFromObject(com_Component.Properties());
com_Properties = var_Properties;
        var_Property = COM::createFromObject(com_Properties).Add("Duration2");
com_Property = var_Property;
            com_Property.Value(COMVariant::createFromReal(2.5));
            com_Property.Type(6/*exPropertyTypeDuration*/);
    var_Property1 =
COM::createFromObject(com_exicalendar1.Root().Properties()).Item("Duration1");
com_Property1 = var_Property1;
        print( com_Property1.Name() );
        print( "Guess" );
        print( com_Property1.GuessType() );
        print( com_Property1.Name() );
        print( "Type" );
        print( com_Property1.Type() );
    var_Property2 =
COM::createFromObject(com_exicalendar1.Root().Properties()).Item("Duration2");
com_Property2 = var_Property2;
        print( com_Property2.Name() );
        print( "Guess" );
        print( com_Property2.GuessType() );
        print( com_Property2.Name() );
        print( "Type" );
        print( com_Property2.Type() );
}
```

Delphi 8 (.NET only)

```
AxICalendar1 :=
(ComObj.CreateComObject(ComObj.ProgIDToClassID('Exontrol.ICalendar.1')) as
EXICALENDARLib.ICalendar);
with AxICalendar1 do
begin
    with Content.Components.Add('VCALENDAR') do
    begin

Properties.Add('Duration1',TObject(AxICalendar1.toICalendar[TObject(2.5),EXICALEND

        with Properties.Add('Duration2',Nil) do
        begin
            Value := TObject(2.5);
            Type := EXICALENDARLib.PropertyTypeEnum.exPropertyTypeDuration;
        end;
    end;
    with Root.Properties.Item['Duration1'] do
    begin
        OutputDebugString( Name );
        OutputDebugString( 'Guess' );
        OutputDebugString( GuessType );
        OutputDebugString( Name );
        OutputDebugString( 'Type' );
        OutputDebugString( Type );
    end;
    with Root.Properties.Item['Duration2'] do
    begin
        OutputDebugString( Name );
        OutputDebugString( 'Guess' );
        OutputDebugString( GuessType );
        OutputDebugString( Name );
        OutputDebugString( 'Type' );
        OutputDebugString( Type );
    end;
end
```

Delphi (standard)

```
ICalendar1 :=
(IUnknown(ComObj.CreateComObject(ComObj.ProgIDToClassID('Exontrol.ICalendar.1'))
 as EXICALENDARLib_TLB.ICalendar);
with ICalendar1 do
begin
    with Content.Components.Add('VCALENDAR') do
    begin

Properties.Add('Duration1',OleVariant(ICalendar1.toICalendar[OleVariant(2.5),EXICALEN

        with Properties.Add('Duration2',Null) do
        begin
            Value := OleVariant(2.5);
            Type := EXICALENDARLib_TLB.exPropertyTypeDuration;
        end;
    end;
    with Root.Properties.Item['Duration1'] do
    begin
        OutputDebugString( Name );
        OutputDebugString( 'Guess' );
        OutputDebugString( GuessType );
        OutputDebugString( Name );
        OutputDebugString( 'Type' );
        OutputDebugString( Type );
    end;
    with Root.Properties.Item['Duration2'] do
    begin
        OutputDebugString( Name );
        OutputDebugString( 'Guess' );
        OutputDebugString( GuessType );
        OutputDebugString( Name );
        OutputDebugString( 'Type' );
        OutputDebugString( Type );
    end;
end
```

VFP

```
thisform.ICalendar1 = CreateObject("Exontrol.ICalendar.1")
with thisform.ICalendar1
    with .Content.Components.Add("VCALENDAR")
        .Properties.Add("Duration1",thisform.ICalendar1.toICalendar(2.5,6))
        with .Properties.Add("Duration2")
            .Value = 2.5
            .Type = 6
        endwith
    endwith
    with .Root.Properties.Item("Duration1")
        DEBUGOUT( .Name )
        DEBUGOUT( "Guess" )
        DEBUGOUT( .GuessType )
        DEBUGOUT( .Name )
        DEBUGOUT( "Type" )
        DEBUGOUT( .Type )
    endwith
    with .Root.Properties.Item("Duration2")
        DEBUGOUT( .Name )
        DEBUGOUT( "Guess" )
        DEBUGOUT( .GuessType )
        DEBUGOUT( .Name )
        DEBUGOUT( "Type" )
        DEBUGOUT( .Type )
    endwith
endwith
```

dBASE Plus

```
local oICalendar,var_Component,var_Property,var_Property1,var_Property2

oICalendar = new OleAutoClient("Exontrol.ICalendar.1")

var_Component = oICalendar.Content.Components.Add("VCALENDAR")
    var_Component.Properties.Add("Duration1",oICalendar.toICalendar(2.5,6))
    var_Property = var_Component.Properties.Add("Duration2")
        var_Property.Value = 2.5
        var_Property.Type = 6
var_Property1 = oICalendar.Root.Properties.Item("Duration1")
    ? var_Property1.Name
    ? "Guess"
    ? Str(var_Property1.GuessType)
    ? var_Property1.Name
    ? "Type"
    ? Str(var_Property1.Type)
var_Property2 = oICalendar.Root.Properties.Item("Duration2")
    ? var_Property2.Name
    ? "Guess"
    ? Str(var_Property2.GuessType)
    ? var_Property2.Name
    ? "Type"
    ? Str(var_Property2.Type)
```

XBasic (Alpha Five)

```
Dim oICalendar as P
Dim var_Component as P
Dim var_Property as P
Dim var_Property1 as P
Dim var_Property2 as P

oICalendar = OLE.Create("Exontrol.ICalendar.1")

var_Component = oICalendar.Content.Components.Add("VCALENDAR")
    var_Component.Properties.Add("Duration1",oICalendar.toICalendar(2.5,6))
    var_Property = var_Component.Properties.Add("Duration2")
        var_Property.Value = 2.5
        var_Property.Type = 6
var_Property1 = oICalendar.Root.Properties.Item("Duration1")
    ? var_Property1.Name
    ? "Guess"
    ? var_Property1.GuessType
    ? var_Property1.Name
    ? "Type"
    ? var_Property1.Type
var_Property2 = oICalendar.Root.Properties.Item("Duration2")
    ? var_Property2.Name
    ? "Guess"
    ? var_Property2.GuessType
    ? var_Property2.Name
    ? "Type"
    ? var_Property2.Type
```

Visual Objects

```
local var_Component as IComponent
local var_Property,var_Property1,var_Property2 as IProperty

oDCOCX_Exontrol1 := IICalendar{"Exontrol.ICalendar.1"}
var_Component := oDCOCX_Exontrol1:Content:Components:Add("VCALENDAR")
    var_Component:Properties:Add("Duration1",oDCOCX_Exontrol1:
[toICalendar,2.5,exPropertyTypeDuration])
    var_Property := var_Component:Properties:Add("Duration2",nil)
        var_Property:Value := 2.5
        var_Property:Type := exPropertyTypeDuration
var_Property1 := oDCOCX_Exontrol1:Root:Properties:[Item,"Duration1"]
    OutputDebugString(String2Psz( var_Property1:Name ))
    OutputDebugString(String2Psz( "Guess" ))
    OutputDebugString(String2Psz( AsString(var_Property1:GuessType) ))
    OutputDebugString(String2Psz( var_Property1:Name ))
    OutputDebugString(String2Psz( "Type" ))
    OutputDebugString(String2Psz( AsString(var_Property1:Type) ))
var_Property2 := oDCOCX_Exontrol1:Root:Properties:[Item,"Duration2"]
    OutputDebugString(String2Psz( var_Property2:Name ))
    OutputDebugString(String2Psz( "Guess" ))
    OutputDebugString(String2Psz( AsString(var_Property2:GuessType) ))
    OutputDebugString(String2Psz( var_Property2:Name ))
    OutputDebugString(String2Psz( "Type" ))
    OutputDebugString(String2Psz( AsString(var_Property2:Type) ))
```

PowerBuilder

```
OleObject oICalendar,var_Component,var_Property,var_Property1,var_Property2

oICalendar = CREATE OLEObject
oICalendar.ConnectToNewObject("Exontrol.ICalendar.1")

var_Component = oICalendar.Content.Components.Add("VCALENDAR")
    var_Component.Properties.Add("Duration1",oICalendar.toICalendar(2.5,6))
    var_Property = var_Component.Properties.Add("Duration2")
        var_Property.Value = 2.5
        var_Property.Type = 6
var_Property1 = oICalendar.Root.Properties.Item("Duration1")
    MessageBox("Information",string( var_Property1.Name ))
    MessageBox("Information",string( "Guess" ))
    MessageBox("Information",string( String(var_Property1.GuessType) ))
    MessageBox("Information",string( var_Property1.Name ))
    MessageBox("Information",string( "Type" ))
    MessageBox("Information",string( String(var_Property1.Type) ))
var_Property2 = oICalendar.Root.Properties.Item("Duration2")
```

```
    MessageBox("Information",string( var_Property2.Name ))
    MessageBox("Information",string( "Guess" ))
    MessageBox("Information",string( String(var_Property2.GuessType) ))
    MessageBox("Information",string( var_Property2.Name ))
    MessageBox("Information",string( "Type" ))
    MessageBox("Information",string( String(var_Property2.Type) ))
```

Visual DataFlex

```
Procedure OnCreate
    Forward Send OnCreate
    Variant oComICalendar1
    Get ComCreateObject "Exontrol.ICalendar.1" to oComICalendar1

    Variant voComponent
    Get ComContent to voComponent
    Handle hoComponent
    Get Create (RefClass(cComComponent)) to hoComponent
    Set pvComObject of hoComponent to voComponent
        Variant voComponents
        Get ComComponents of hoComponent to voComponents
        Handle hoComponents
        Get Create (RefClass(cComComponents)) to hoComponents
        Set pvComObject of hoComponents to voComponents
            Variant voComponent1
            Get ComAdd of hoComponents "VCALENDAR" to voComponent1
            Handle hoComponent1
            Get Create (RefClass(cComComponent)) to hoComponent1
            Set pvComObject of hoComponent1 to voComponent1
                Variant voProperties
                Get ComProperties of hoComponent1 to voProperties
                Handle hoProperties
                Get Create (RefClass(cComProperties)) to hoProperties
                Set pvComObject of hoProperties to voProperties
                    Variant vValue
                        Get ComtoICalendar 2.5 OLEexPropertyTypeDuration to vValue
                    Get ComAdd of hoProperties "Duration1" vValue to Nothing
                Send Destroy to hoProperties
                Variant voProperties1
                Get ComProperties of hoComponent1 to voProperties1
                Handle hoProperties1
                Get Create (RefClass(cComProperties)) to hoProperties1
                Set pvComObject of hoProperties1 to voProperties1
                    Variant voProperty
                    Get ComAdd of hoProperties1 "Duration2" Nothing to voProperty
                    Handle hoProperty
                    Get Create (RefClass(cComProperty)) to hoProperty
                    Set pvComObject of hoProperty to voProperty
                        Set ComValue of hoProperty to 2.5
                        Set ComType of hoProperty to OLEexPropertyTypeDuration
                    Send Destroy to hoProperty
                Send Destroy to hoProperties1
            Send Destroy to hoComponent1
        Send Destroy to hoComponents
    Send Destroy to hoComponent
    Variant voComponent2
    Get ComRoot to voComponent2
    Handle hoComponent2
    Get Create (RefClass(cComComponent)) to hoComponent2
    Set pvComObject of hoComponent2 to voComponent2
        Variant voProperties2
        Get ComProperties of hoComponent2 to voProperties2
        Handle hoProperties2
        Get Create (RefClass(cComProperties)) to hoProperties2
        Set pvComObject of hoProperties2 to voProperties2
            Variant voProperty1
            Get ComItem of hoProperties2 "Duration1" to voProperty1
            Handle hoProperty1
            Get Create (RefClass(cComProperty)) to hoProperty1
            Set pvComObject of hoProperty1 to voProperty1
                Variant voProperty2
                Get ComItem of hoProperty1 "Duration1" to voProperty2
                Handle hoProperty2
                Get Create (RefClass(cComProperty)) to hoProperty2
                Set pvComObject of hoProperty2 to voProperty2
                    Showln (ComName(hoProperty2)) "Guess"
    (ComGuessType(hoProperty2))
                    Showln (ComName(hoProperty2)) "Type" (ComType(hoProperty2))
                Send Destroy to hoProperty2
            Send Destroy to hoProperty1
        Send Destroy to hoProperties2
      Send Destroy to hoComponent2
      Variant voComponent3
      Get ComRoot to voComponent3
      Handle hoComponent3
      Get Create (RefClass(cComComponent)) to hoComponent3
      Set pvComObject of hoComponent3 to voComponent3
        Variant voProperties3
        Get ComProperties of hoComponent3 to voProperties3
        Handle hoProperties3
        Get Create (RefClass(cComProperties)) to hoProperties3
        Set pvComObject of hoProperties3 to voProperties3
            Variant voProperty3
            Get ComItem of hoProperties3 "Duration2" to voProperty3
            Handle hoProperty3
            Get Create (RefClass(cComProperty)) to hoProperty3
            Set pvComObject of hoProperty3 to voProperty3
                Variant voProperty4
                Get ComItem of hoProperty3 "Duration2" to voProperty4
                Handle hoProperty4
                Get Create (RefClass(cComProperty)) to hoProperty4
                Set pvComObject of hoProperty4 to voProperty4
                    Showln (ComName(hoProperty4)) "Guess"
    (ComGuessType(hoProperty4))
                    Showln (ComName(hoProperty4)) "Type" (ComType(hoProperty4))
                Send Destroy to hoProperty4
            Send Destroy to hoProperty3
        Send Destroy to hoProperties3
      Send Destroy to hoComponent3
    End_Procedure
```

XBase++

```
#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
   LOCAL oForm
   LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
   LOCAL oICalendar
   LOCAL oComponent
   LOCAL oProperty,oProperty1,oProperty2

   oForm := XbpDialog():new( AppDesktop() )
   oForm:drawingArea:clipChildren := .T.
   oForm:create( ,,{100,100}, {640,480},, .F. )
   oForm:close  := {|| PostAppEvent( xbeP_Quit )}

   oICalendar := XbpActiveXControl():new( oForm:drawingArea )
   oICalendar:CLSID  := "Exontrol.ICalendar.1" /*{D6C87100-38E2-4ABB-8AC2-
4C0097AEE2D6}*/
   oICalendar:create(,, {10,60},{610,370} )

      oComponent := oICalendar:Content():Components():Add("VCALENDAR")

oComponent:Properties():Add("Duration1",oICalendar:toICalendar(2.5,6/*exPropertyTy

         oProperty := oComponent:Properties():Add("Duration2")
            oProperty:Value := 2.5
            oProperty:Type := 6/*exPropertyTypeDuration*/
      oProperty1 := oICalendar:Root():Properties:Item("Duration1")
         DevOut( oProperty1:Name() )
         DevOut( "Guess" )
         DevOut( Transform(oProperty1:GuessType(),"") )
         DevOut( oProperty1:Name() )
         DevOut( "Type" )
         DevOut( Transform(oProperty1:Type(),"") )
      oProperty2 := oICalendar:Root():Properties:Item("Duration2")
```

```
          DevOut( oProperty2:Name() )
          DevOut( "Guess" )
          DevOut( Transform(oProperty2:GuessType(),"") )
          DevOut( oProperty2:Name() )
          DevOut( "Type" )
          DevOut( Transform(oProperty2:Type(),"") )

   oForm:Show()
   DO WHILE nEvent != xbeP_Quit
      nEvent := AppEvent( @mp1, @mp2, @oXbp )
      oXbp:handleEvent( nEvent, mp1, mp2 )
   ENDDO
RETURN
```

property Property.UserData as Variant

Indicates any extra data associated with the property.

Type	Description
Variant	A VARIANT expression that specifies any extra data associated with the property.

The UserData property holds any extra data associated with the property object. The Value property specifies the value/values of the property. The Name property specifies the name of the Property. The AddProperty event notifies your application once a new Property object is added to the Properties collection.

property Property.Value as Variant

Indicates the property's value(or values if a safe array is used).

Type	Description
Variant	A VARIANT expression that specifies the value of the property.

The Value property specifies the value of the property. The Value parameter of the Add property indicates the value of the property to be added. The GuessType property guesses the property's type, from its value. The Type property specifies the type of the property from the Value parameter of the property. The Parameters property specifies the property's Parameters collection. The Name property indicates the name of the property. The valuesFromICalendar property extracts all values or specified value of the giving value/type in ICalendar format. The toICalendar property converts the giving value to a specified type.

If the property's type is exPropertyTypeRecur, you can use any of the following properties:

- RecurAll property to return all occurrences of the specified recurrence rule as a safe array of DATEs
- RecurAllAsString property to return all occurrences of the specified recurrence rule as a string
- RecurCheck property to determine whether the giving date fits the giving recurrence rule
- RecurRange property to return all occurrences between specified range, of giving recurrence rule as a safe array of DATEs
- RecurRangeAsString property to return all occurrences between specified range, of giving recurrence rule as a string

The following is a simple example of an iCalendar format:

BEGIN:VCALENDAR
BEGIN:VEVENT
SUMMARY;LANGUAGE=en-US:**Company Holiday Party**
DATE:**20010101**
END:VEVENT
END:VCALENDAR

The 20010101 and Company Holiday Party are the values of the properties DATE and SUMMARY of the VCALENDAR object.

ExICalendar events

The order of the events for Load, LoadFile methods is:

- StartLoad event, notifies that the Load, LoadFile methods begins
- AddComponent(NewComponent) event, notifies that a new Component object is added to the Components collection.
- AddProperty(NewPropery) event, notifies that a new Property object is added to the Properties collection. The Component property of the Property object specifies owner/parent component of the newly added property.
- AddParameter(NewParameter) event, notifies that a new Parameter object is added to the Parameters collection. The Property property of the Parameter object specifies owner/parent property of the newly added parameter.
- EndLoad event, notifies that the Load, LoadFile methods ends.

The ExICalendar component supports the following events:

Name	Description
AddComponent	Occurs when a new component is added.
AddParameter	Occurs when a new parameter is added.
AddProperty	Occurs when a new property is added.
EndLoad	Notifies your application that the control's Load ends.
Event	Notifies the application once the control fires an event.
StartLoad	Notifies your application that the control's Load starts.

event AddComponent (NewComponent as Component)

Occurs when a new component is added.

Type	Description
NewComponent as Component	A Component object being created and added to the Components collection.

The AddComponent event notifies your application once a new Component object is added to the Components collection. The AddComponent event is fired only if the control's FireEvents property is True. The body of the iCalendar object consists of a sequence of calendar properties and one or more calendar components. The calendar properties are attributes that apply to the calendar object as a whole. The calendar components are collections of properties that express a particular calendar semantic. For example, the calendar component can specify an event, a to-do, a journal entry, time zone information, free/busy time information, or an alarm. For instance, the AddComponent event occurs when the LoadFile method encounters a "BEGIN" ":" iana-token sequence. Use the UserData property to associate any extra-data to the component object.

The order of the events for Load, LoadFile methods is:

- StartLoad event, notifies that the Load, LoadFile methods begins
- AddComponent(NewComponent) event, notifies that a new Component object is added to the Components collection.
- AddProperty(NewPropery) event, notifies that a new Property object is added to the Properties collection. The Component property of the Property object specifies owner/parent component of the newly added property.
- AddParameter(NewParameter) event, notifies that a new Parameter object is added to the Parameters collection. The Property property of the Parameter object specifies owner/parent property of the newly added parameter.
- EndLoad event, notifies that the Load, LoadFile methods ends.

The following is a simple example of an iCalendar component:

BEGIN:**VEVENT**
UID:19970610T172345Z-AF23B2@example.com
DTSTAMP:19970610T172345Z
DTSTART:19970714T170000Z
DTEND:19970715T040000Z
SUMMARY:Bastille Day Party
END:**VEVENT**

Syntax for AddComponent event, **/NET** version, on:

C#

```
private void AddComponent(object sender,exontrol.EXICALENDARLib.Component
NewComponent)
{
}
```

VB

```
Private Sub AddComponent(ByVal sender As System.Object,ByVal NewComponent
As exontrol.EXICALENDARLib.Component) Handles AddComponent
End Sub
```

Syntax for AddComponent event, **/COM** version, on:

C#

```
private void AddComponent(object sender,
AxEXICALENDARLib._IICalendarEvents_AddComponentEvent e)
{
}
```

C++

```
void OnAddComponent(LPDISPATCH NewComponent)
{
}
```

C++ Builder

```
void __fastcall AddComponent(TObject *Sender,Exicalendarlib_tlb::IComponent
*NewComponent)
{
}
```

Delphi

```
procedure AddComponent(ASender: TObject; NewComponent : IComponent);
begin
end;
```

Delphi 8 (.NET only)

```
procedure AddComponent(sender: System.Object; e:
AxEXICALENDARLib._IICalendarEvents_AddComponentEvent);
begin
end;
```

Powe…

```
begin event AddComponent(oleobject NewComponent)
end event AddComponent
```

VB.NET
```
Private Sub AddComponent(ByVal sender As System.Object, ByVal e As
AxEXICALENDARLib._IICalendarEvents_AddComponentEvent) Handles
AddComponent
End Sub
```

VB6
```
Private Sub AddComponent(ByVal NewComponent As
EXICALENDARLibCtl.IComponent)
End Sub
```

VBA
```
Private Sub AddComponent(ByVal NewComponent As Object)
End Sub
```

VFP
```
LPARAMETERS NewComponent
```

Xbas…
```
PROCEDURE OnAddComponent(oICalendar,NewComponent)
RETURN
```

Syntax for AddComponent event, **/COM** version (others), on:

Java…
```
<SCRIPT EVENT="AddComponent(NewComponent)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function AddComponent(NewComponent)
End Function
</SCRIPT>
```

Visual Data…
```
Procedure OnComAddComponent Variant llNewComponent
    Forward Send OnComAddComponent llNewComponent
End_Procedure
```

Visual Objects
```
METHOD OCX_AddComponent(NewComponent) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_AddComponent(COM _NewComponent)
{
}
```

XBasic

```
function AddComponent as v (NewComponent as
OLE::Exontrol.ICalendar.1::IComponent)
end function
```

dBASE

```
function nativeObject_AddComponent(NewComponent)
return
```

event AddParameter (NewParameter as Parameter)

Occurs when a new parameter is added.

Type	Description
NewParameter as Parameter	A Parameter object being created and added to the Parameters collection.

The AddParameter event notifies your application once a new Parameter object is added to the Parameters collection. The AddParameter event is fired only if the control's FireEvents property is True. Property parameters with values containing a COLON character, a SEMICOLON character or a COMMA character MUST be placed in quoted text. Use the UserData property to associate any extra-data to the parameter object.

The order of the events for Load, LoadFile methods is:

- StartLoad event, notifies that the Load, LoadFile methods begins
- AddComponent(NewComponent) event, notifies that a new Component object is added to the Components collection.
- AddProperty(NewPropery) event, notifies that a new Property object is added to the Properties collection. The Component property of the Property object specifies owner/parent component of the newly added property.
- AddParameter(NewParameter) event, notifies that a new Parameter object is added to the Parameters collection. The Property property of the Parameter object specifies owner/parent property of the newly added parameter.
- EndLoad event, notifies that the Load, LoadFile methods ends.

The following is an example of a parameter:

RDATE;**VALUE**=DATE:19970304,19970504,19970704,19970904

Syntax for AddParameter event, **/NET** version, on:

C#

```
private void AddParameter(object sender,exontrol.EXICALENDARLib.Parameter
NewParameter)
{
}
```

VB

```
Private Sub AddParameter(ByVal sender As System.Object,ByVal NewParameter As
exontrol.EXICALENDARLib.Parameter) Handles AddParameter
End Sub
```

Syntax for AddParameter event, **/COM** version, on:

C#
```
private void AddParameter(object sender,
AxEXICALENDARLib._ICalendarEvents_AddParameterEvent e)
{
}
```

C++
```
void OnAddParameter(LPDISPATCH NewParameter)
{
}
```

C++ Builder
```
void __fastcall AddParameter(TObject *Sender,Exicalendarlib_tlb::IParameter
*NewParameter)
{
}
```

Delphi
```
procedure AddParameter(ASender: TObject; NewParameter : IParameter);
begin
end;
```

Delphi 8 (.NET only)
```
procedure AddParameter(sender: System.Object; e:
AxEXICALENDARLib._ICalendarEvents_AddParameterEvent);
begin
end;
```

Powe…
```
begin event AddParameter(oleobject NewParameter)
end event AddParameter
```

VB.NET
```
Private Sub AddParameter(ByVal sender As System.Object, ByVal e As
AxEXICALENDARLib._ICalendarEvents_AddParameterEvent) Handles
AddParameter
End Sub
```

VB6
```
Private Sub AddParameter(ByVal NewParameter As
EXICALENDARLibCtl.IParameter)
End Sub
```

VBA
```
Private Sub AddParameter(ByVal NewParameter As Object)
End Sub
```

VFP

```
LPARAMETERS NewParameter
```

Xbas...

```
PROCEDURE OnAddParameter(oICalendar,NewParameter)
RETURN
```

Syntax for AddParameter event, **/COM** version (others), on:

Java...

```
<SCRIPT EVENT="AddParameter(NewParameter)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc...

```
<SCRIPT LANGUAGE="VBScript">
Function AddParameter(NewParameter)
End Function
</SCRIPT>
```

Visual Data...

```
Procedure OnComAddParameter Variant llNewParameter
    Forward Send OnComAddParameter llNewParameter
End_Procedure
```

Visual Objects

```
METHOD OCX_AddParameter(NewParameter) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_AddParameter(COM _NewParameter)
{
}
```

XBasic

```
function AddParameter as v (NewParameter as
OLE::Exontrol.ICalendar.1::IParameter)
end function
```

dBASE

```
function nativeObject_AddParameter(NewParameter)
return
```

event AddProperty (NewPropery as Property)

Occurs when a new property is added.

Type	Description
NewPropery as Property	A Property object being created and added to the Properties collection.

The AddProperty event notifies your application once a new Property object is added to the Properties collection. The AddProperty event is fired only if the control's FireEvents property is True. A property is the definition of an individual attribute describing a calendar object or a calendar component. Use the UserData property to associate any extra-data to the property object.

The order of the events for Load, LoadFile methods is:

- StartLoad event, notifies that the Load, LoadFile methods begins
- AddComponent(NewComponent) event, notifies that a new Component object is added to the Components collection.
- AddProperty(NewPropery) event, notifies that a new Property object is added to the Properties collection. The Component property of the Property object specifies owner/parent component of the newly added property.
- AddParameter(NewParameter) event, notifies that a new Parameter object is added to the Parameters collection. The Property property of the Parameter object specifies owner/parent property of the newly added parameter.
- EndLoad event, notifies that the Load, LoadFile methods ends.

The following is an example of a property:

DTSTAMP:19970610T172345Z

Syntax for AddProperty event, **/NET** version, on:

C#

```
private void AddProperty(object sender,exontrol.EXICALENDARLib.Property
NewPropery)
{
}
```

VB

```
Private Sub AddProperty(ByVal sender As System.Object,ByVal NewPropery As
exontrol.EXICALENDARLib.Property) Handles AddProperty
End Sub
```

Syntax for AddProperty event, **/COM** version, on:

C#
```
private void AddProperty(object sender,
AxEXICALENDARLib._IICalendarEvents_AddPropertyEvent e)
{
}
```

C++
```
void OnAddProperty(LPDISPATCH NewPropery)
{
}
```

C++ Builder
```
void __fastcall AddProperty(TObject *Sender,Exicalendarlib_tlb::IProperty
*NewPropery)
{
}
```

Delphi
```
procedure AddProperty(ASender: TObject; NewPropery : IProperty);
begin
end;
```

Delphi 8 (.NET only)
```
procedure AddProperty(sender: System.Object; e:
AxEXICALENDARLib._IICalendarEvents_AddPropertyEvent);
begin
end;
```

Powe…
```
begin event AddProperty(oleobject NewPropery)
end event AddProperty
```

VB.NET
```
Private Sub AddProperty(ByVal sender As System.Object, ByVal e As
AxEXICALENDARLib._IICalendarEvents_AddPropertyEvent) Handles AddProperty
End Sub
```

VB6
```
Private Sub AddProperty(ByVal NewPropery As EXICALENDARLibCtl.IProperty)
End Sub
```

VBA
```
Private Sub AddProperty(ByVal NewPropery As Object)
End Sub
```

VFP
```
LPARAMETERS NewPropery
```

Xbas…
```
PROCEDURE OnAddProperty(oICalendar,NewPropery)
RETURN
```

Syntax for AddProperty event, **/COM** version (others), on:

Java…
```
<SCRIPT EVENT="AddProperty(NewPropery)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function AddProperty(NewPropery)
End Function
</SCRIPT>
```

Visual Data…
```
Procedure OnComAddProperty Variant llNewPropery
    Forward Send OnComAddProperty llNewPropery
End_Procedure
```

Visual Objects
```
METHOD OCX_AddProperty(NewPropery) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_AddProperty(COM _NewPropery)
{
}
```

XBasic
```
function AddProperty as v (NewPropery as OLE::Exontrol.ICalendar.1::IProperty)
end function
```

dBASE
```
function nativeObject_AddProperty(NewPropery)
return
```

event EndLoad ()

Notifies your application that the control's Load ends.

Type	Description

The EndLoad event, notifies that the Load, LoadFile methods ends.

The order of the events for Load, LoadFile methods is:

- StartLoad event, notifies that the Load, LoadFile methods begins
- AddComponent(NewComponent) event, notifies that a new Component object is added to the Components collection.
- AddProperty(NewPropery) event, notifies that a new Property object is added to the Properties collection. The Component property of the Property object specifies owner/parent component of the newly added property.
- AddParameter(NewParameter) event, notifies that a new Parameter object is added to the Parameters collection. The Property property of the Parameter object specifies owner/parent property of the newly added parameter.
- EndLoad event, notifies that the Load, LoadFile methods ends.

Syntax for EndLoad event, **/NET** version, on:

C#

```
private void EndLoad(object sender)
{
}
```

VB

```
Private Sub EndLoad(ByVal sender As System.Object) Handles EndLoad
End Sub
```

Syntax for EndLoad event, **/COM** version, on:

C#

```
private void EndLoad(object sender, EventArgs e)
{
}
```

C++

```
void OnEndLoad()
{
}
```

C++ Builder

```
void __fastcall EndLoad(TObject *Sender)
{
}
```

Delphi

```
procedure EndLoad(ASender: TObject; );
begin
end;
```

Delphi 8 (.NET only)

```
procedure EndLoad(sender: System.Object; e: System.EventArgs);
begin
end;
```

Powe...

```
begin event EndLoad()
end event EndLoad
```

VB.NET

```
Private Sub EndLoad(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles EndLoad
End Sub
```

VB6

```
Private Sub EndLoad()
End Sub
```

VBA

```
Private Sub EndLoad()
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas...

```
PROCEDURE OnEndLoad(oICalendar)
RETURN
```

Syntax for EndLoad event, **/COM** version (others), on:

Java...

```
<SCRIPT EVENT="EndLoad()" LANGUAGE="JScript">
</SCRIPT>
```

VBSc...

```
<SCRIPT LANGUAGE="VBScript">
Function EndLoad()
End Function
</SCRIPT>
```

Visual Data...

```
Procedure OnComEndLoad
   Forward Send OnComEndLoad
End_Procedure
```

Visual Objects

```
METHOD OCX_EndLoad() CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_EndLoad()
{
}
```

XBasic

```
function EndLoad as v ()
end function
```

dBASE

```
function nativeObject_EndLoad()
return
```

event Event (EventID as Long)

Notifies the application once the control fires an event.

Type	Description
EventID as Long	A Long expression that specifies the identifier of the event. Each internal event of the control has an unique identifier. Use the EventParam(-2) to display entire information about fired event (such as name, identifier, and properties). The EventParam(-1) retrieves the number of parameters of fired event

The Event notification occurs ANY time the control fires an event. *This is useful for X++, which does not support event with parameters passed by reference. Also, this could be useful for C++ Builder or Delphi, which does not handle properly the events with parameters of VARIANT type.*

In X++ the "Error executing code: FormActiveXControl (data source), method ... called with invalid parameters" occurs when handling events that have parameters passed by reference. Passed by reference, means that in the event handler, you can change the value for that parameter, and so the control will takes the new value, and use it. The X++ is NOT able to handle properly events with parameters by reference, so we have the solution.

The solution is using and handling the Event notification and EventParam method., instead handling the event that gives the "invalid parameters" error executing code.

If you are not familiar with what a type library means just handle the Event of the control as follows:

```
// Notifies the application once the control fires an event.
void onEvent_Event(int _EventID)
{
   print exgrid1.EventParam(-2).toString();
}
```

This code allows you to display the information for each event of the control being fired as in the list bellow:

```
StartLoad/1
AddComponent/2( [Object] )
AddProperty/3( [Object] )
AddProperty/3( [Object] )
```

```
AddComponent/2( [Object] )
AddProperty/3( [Object] )
AddParameter/4( [Object] )
AddParameter/4( [Object] )
AddParameter/4( [Object] )
AddProperty/3( [Object] )
AddParameter/4( [Object] )
AddParameter/4( [Object] )
AddProperty/3( [Object] )
AddParameter/4( [Object] )
EndLoad/5
```

Each line indicates an event, and the following information is provided: the name of the event, its identifier, and the list of parameters being passed to the event. The parameters that starts with = character, indicates a parameter by reference, in other words one that can changed during the event handler.

In conclusion, anytime the X++ fires the "invalid parameters." while handling an event, you can use and handle the Event notification and EventParam methods of the control

Syntax for Event event, **/NET** version, on:

C#

```
private void Event(object sender,int EventID)
{
}
```

VB

```
Private Sub Event(ByVal sender As System.Object,ByVal EventID As Integer)
Handles Event
End Sub
```

Syntax for Event event, **/COM** version, on:

C#

```
private void Event(object sender,
AxEXICALENDARLib._ICalendarEvents_EventEvent e)
{
}
```

C++

```
void OnEvent(long EventID)
{
}
```

C++ Builder

```
void __fastcall Event(TObject *Sender,long EventID)
{
}
```

Delphi

```
procedure Event(ASender: TObject; EventID : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure Event(sender: System.Object; e:
AxEXICALENDARLib._IICalendarEvents_EventEvent);
begin
end;
```

Powe…

```
begin event Event(long EventID)
end event Event
```

VB.NET

```
Private Sub Event(ByVal sender As System.Object, ByVal e As
AxEXICALENDARLib._IICalendarEvents_EventEvent) Handles Event
End Sub
```

VB6

```
Private Sub Event(ByVal EventID As Long)
End Sub
```

VBA

```
Private Sub Event(ByVal EventID As Long)
End Sub
```

VFP

```
LPARAMETERS EventID
```

Xbas…

```
PROCEDURE OnEvent(oICalendar,EventID)
RETURN
```

Syntax for Event event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="Event(EventID)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
```

```
Function Event(EventID)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComEvent Integer llEventID
    Forward Send OnComEvent llEventID
End_Procedure
```

Visual Objects

```
METHOD OCX_Event(EventID) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_Event(int _EventID)
{
}
```

XBasic

```
function Event as v (EventID as N)
end function
```

dBASE

```
function nativeObject_Event(EventID)
return
```

event StartLoad ()

Notifies your application that the control's Load starts.

Type	Description

The StartLoad event, notifies that the Load, LoadFile, LoadFileFromUnicode methods begins.

The order of the events for Load, LoadFile, LoadFileFromUnicode methods is:

- StartLoad event, notifies that the Load, LoadFile, LoadFileFromUnicode methods begins
- AddComponent(NewComponent) event, notifies that a new Component object is added to the Components collection.
- AddProperty(NewPropery) event, notifies that a new Property object is added to the Properties collection. The Component property of the Property object specifies owner/parent component of the newly added property.
- AddParameter(NewParameter) event, notifies that a new Parameter object is added to the Parameters collection. The Property property of the Parameter object specifies owner/parent property of the newly added parameter.
- EndLoad event, notifies that the Load, LoadFile, LoadFileFromUnicode methods ends.

Syntax for StartLoad event, **/NET** version, on:

C#
```
private void StartLoad(object sender)
{
}
```

VB
```
Private Sub StartLoad(ByVal sender As System.Object) Handles StartLoad
End Sub
```

Syntax for StartLoad event, **/COM** version, on:

C#
```
private void StartLoad(object sender, EventArgs e)
{
}
```

C++
```
void OnStartLoad()
{
}
```

C++ Builder

```
void __fastcall StartLoad(TObject *Sender)
{
}
```

Delphi

```
procedure StartLoad(ASender: TObject; );
begin
end;
```

Delphi 8 (.NET only)

```
procedure StartLoad(sender: System.Object; e: System.EventArgs);
begin
end;
```

Powe...

```
begin event StartLoad()
end event StartLoad
```

VB.NET

```
Private Sub StartLoad(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles StartLoad
End Sub
```

VB6

```
Private Sub StartLoad()
End Sub
```

VBA

```
Private Sub StartLoad()
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas...

```
PROCEDURE OnStartLoad(oICalendar)
RETURN
```

Syntax for StartLoad event, **/COM** version (others), on:

Java...

```
<SCRIPT EVENT="StartLoad()" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function StartLoad()
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComStartLoad
    Forward Send OnComStartLoad
End_Procedure
```

Visual Objects

```
METHOD OCX_StartLoad() CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_StartLoad()
{
}
```

XBasic

```
function StartLoad as v ()
end function
```

dBASE

```
function nativeObject_StartLoad()
return
```