



# Youseful (Version 1.2) Help Contents

A Bill White Software Production

[Installing this software into the Delphi VCL](#)

[Merging the Youseful help into the Delphi Help System](#)



[Registering this software](#) (Unless this is the registered version)



[It just won't work](#)



[Version Info](#)

## TINSTALL DEPENDENT COMPONENTS



[TInstall](#)



[TInstallFileGroup](#)

[TCopyFileDlg](#)



[TInstallFileGroupsDlg](#)



[TInstallODBC](#) { Work in progress. Does not work. }



[TGoodInstallationDlg](#)



[TComponentConflictDlg](#)

## STAND-ALONE COMPONENTS



TInstallAlias



TInstallINIFile



TProgramItem



TInstallRSConnection



TSelectWindowGroupDlg



TInstallRSConnectionDlg



TColorFade



TModifiedReport



TReportManager

## **UNINSTALL COMPONENTS**



TUnInstall



TUnInstallingFileDialog

## **NON-VISUAL COMPONENTS**

TInstallFile

## **TECHNICAL SUPPORT**



**World Wide Web Site:** <http://www.destek.net/cybermkt/blwhite.htm>



[FTP Site: ftp.is.net/pub/csm/youseful.zip](ftp://ftp.is.net/pub/csm/youseful.zip)

### Author Information

#### **STANDARD LEGAL DISCLAIMER**

Youseful Components Version 1.0.

Copyright 1995

Bill White.

All rights reserved. I am not liable for any damage this might inflict on your computer, spouse, children, dog, or lawn. If you have the registered version, you may NOT DISTRIBUTE it. If you have the shareware version (un-registered version), please feel free to distribute like hell.



## TInstallFileGroup Component



[Properties](#)



[Methods](#)

[Events](#)

### Unit

Install

### Description

The TInstallFileGroup Component is really not a stand-alone component; its only real use comes if you hook it up to a TInstall component through the [InstallComponent](#) property. Basically, this component groups files together in your installation that are related--related meaning that you want these files to be copied to the same directory on the user's computer. Note: it does not matter necessarily which directory, the salient point here is that the files should be copied to the same directory, no matter which directory it may be. (e.g. this is a valid point because you can give the users the option to include/exclude file groups with the [Include](#) property and [AlwaysInstall](#) property). To always install the file group on the user's computer, set the AlwaysInstall property to True. To include a file group in the installation and allow the user to include/disclude that group, set the AlwaysInstall property to False and the Include property to True. If you always want the files in this group to be copied to a subdirectory of where another file group is copied, set the [AllowChangeDir](#) to False, the [ParentFileGroup](#) property to point to the other file group, and use the %f meta-variable in the DestDir property.

The [GroupName](#) property allows you to attach a descriptive name to the file group. The [Description](#) property allows you to attach a description of the file group to the component. The [TInstallFileGroupsDlg](#) component uses the Description property to show the users. This is also useful if you decide to make your own dialogs.

The [DestDir](#) property contains the directory to which the files that are owned by this component, in the [Files](#) property, will be installed. The number of files that the file group owns is contained in the run-time [FileCount](#) property. The run-time [Size](#) property contains the size of the file group in Bytes. This is calculated by summing the sizes of all the files owned by the file group.



## Properties

CheckForVCLConflict

CopyIfNewer

Disk

FileGroup

FileName

IncludeFromDir

Size



## TInstall Component (Version 3.0)



[Properties](#)



[Methods](#)

[Events](#)

### Unit

Install

### Component Editor

[How to build the installation files](#)

### Description

[Why TInstall?](#)

[Getting Started--BASIC](#)

[Getting Started--ADVANCED](#)

[Organization Of Setup Files](#)

The TInstall component is a component that provides installation services for applications written in Delphi. Services include: installing files, aliases, INI Files, ReportSmith connections, Program Items, and an Installation Builder program to build the installation disk(s)/files for you. Because the TInstall component is a native Delphi VCL component, if you want to do anything slightly different with your installation you can do it and most importantly you can do it using native Delphi code! No Scripts! The TInstall component also supports inheritance, polymorphism, and all the other object-oriented concepts that makes Delphi so powerfull. So you can create your own TInstall subclasses! You should refer to [Getting Started--BASIC](#) and [Getting Started--ADVANCED](#) for a step-by-step explanation of how to build your installation.

The TInstall allows you to combine the other powerfull components in this package to create a dazzling installation program (Of course, you can use these various components by themselves). Simply plopp down the components that you want to use onto your main installation form, assign values to the properties, and then call the [Install](#) method somewhere to start the installation rolling. **You may abort the installation process at anytime, (programmatically), by raising the EAbortInstallation exception.**

### IMPORTANT

To make your setup program smaller and load faster, go to the Options|Project. You should see a screen with the caption "Project Options". Click on the "Linker" tab at the bottom. Make sure the "Optimize for size and load time" check box is checked.



## TInstallAlias Component



[Properties](#)



[Methods](#)

[Events](#)

### Unit

Install

### Description

The InstallAlias component installs an alias into the Borland Database Engine(BDE) when you call the [Install](#) method. The name of the alias is determined by the [AliasName](#) property and the driver for the alias is determined by the [Driver](#) property.

For Standard aliases (e.g. ASCII, DBase, Paradox), the [Path](#) property will contain the path of the data files to which the alias points. The [SQLParams](#) property contains parameters for InterBase-type aliases (Driver property of INTRBASE). Refer to the Borland Database Engine Documentation for more information on aliases.

The TInstallAlias component is actually a stand-alone component. However, you can make the TInstallAlias component part of an installation by setting the [InstallComponent](#) property to any [TInstall component](#) on the form. If this component is linked to a [TInstall component](#), then the [TInstall component](#) will automatically install the alias, when the TInstall's [Install](#) method is called. If you specify a file group in the [FileGroup](#) property and you have also specified a value for the InstallComponent property, then the alias will only be installed if the file group's [Included](#) property is true.

Any [meta-variables](#) that are located in the [Path](#) property or [SQLParams](#) property will automatically be expanded at run-time. Note: the %f [meta-variable](#) will only be expanded if the FileGroup property is linked to a [TInstallFileGroup](#) component on the form.

## Meta-Variables

Meta variables are sequences of characters that are used to represent an internal value. Typically, meta-variables are expanded to their internal representation at run-time. I.e. if the user's window directory is c:\windows, then %w\temp will be expanded to c:\windows\temp.

|    |  |
|----|--|
| %f | Represents the path that the filegroup will be copied to on the user's computer. |
| %s | Represents the system directory for windows on the user's computer.              |
| %w | Represents the windows directory on the user's computer.                         |

Meta variables are case-sensitive. I.e. they must be lowercase.





## TInstallINIFile Component



Properties



Methods

Events

### **Unit**

Install

### **Description**

You should be somewhat familiar with INI files if you want to use this component. Please refer to the appropriate Window's documentation for further information on INI files. Basically, the FileName property specifies both a directory and file name into which the text in the Lines property will be copied with the Install method.

The TInstallINIFile component is actually a stand-alone component. However, you can make the TInstallINIFile component part of an installation by setting the InstallComponent property to any TInstall component on the form. If this component is linked to a TInstall component, then the TInstall component will automatically install the INI file, when its Install method is called. If you specify a file group in the FileGroup property and you have specified a value for the InstallComponent property, then the INI file will only be installed if the file group's Included property is true.

Any meta-variables that are located in the FileName property or Lines property will automatically be expanded at run-time. Note: the %f meta-variable will only be expanded if the FileGroup property is linked to a TInstallFileGroup component on the form.

WARNING:

In its current form, the file specified in the FileName property will be overwritten! This will be fixed.. However, for now, be careful!



## TProgramItem Component



[Properties](#)



[Methods](#)


[Events](#)

**Unit**

Install

### Description

You should be partly familiar with adding Program Manager Items and Groups if you are going to use this component. Please refer to the appropriate Window's documentation if you are unfamiliar with adding Program Manager Items and Groups. The TProgramItem component has 7 properties that are essentially "fill in the blank" for the following dialog. (Don't worry, you and your user will never see this dialog--it is here purely for pedagogical purposes to explain the relations of the properties to how the Program Item will be set-up).

|   |  |                |
|---|--|----------------|
| <b>Description:</b>   | Delphi                                 | OK             |
| <b>Command Line:</b>  | C:\DELPHI\BIN\DELPHI.EXE               | Cancel         |
| <b>Working Directory:</b>   | C:\DELPHI                              | Browse...      |
| <b>Shortcut Key:</b>  | None                                   | Change Icon... |
|  | <input type="checkbox"/> Run Minimized | Help           |

The Description property specifies the Description of the Program Item. The CommandLine property specifies the Command Line for the Program Item. The Directory property specifies the Working Directory of the Program Item. The IconFile property specifies a file that will hold any necessary icons for the Program Item. The IconIndex property specifies which icon in the file will be used as the icon for the Program Item. If you do not specify a value for the IconFile, Window's will display the first icon in the Executable. Any meta-variables in the CommandLine, Directory, and IconFile properties will be expanded at run-time by the Install method. You specify into which Window's group the Program Item will go with the WindowGroup property.

The TProgramItem component is actually a stand-alone component. However, you can make the TProgramItem component part of an installation by setting the InstallComponent property to any TInstall component on the form. If this component is linked to a TInstall component, then the TInstall component will automatically install the Program Item, when it's Install method is called. If you specify a file group in the FileGroup property and you have specified a value for the InstallComponent property, then the Program Item will only be installed if the file group's Included property is true. If the component is not linked to a TInstall component, execute the Install method to install the Program Item.



## TInstallRSConnection Component



[Properties](#)



[Methods](#)

[Events](#)

### Unit

Install

### Description

You should be partly familiar with adding ReportSmith Connections if you are going to use this component. Please refer to the appropriate ReportSmith documentation if you are unfamiliar with this procedure. The TInstallRSConnection component has 6 properties that are essentially "fill in the blanks" for the following dialog. Note: some connection types will prompt you for a User Id and Database. (Don't worry, you and your user will never see this dialog--it is here purely for pedagogical purposes to explain the relations of the properties to how the ReportSmith connections will be set-up). To use BDE aliases in ReportSmith, refer to the TModifiedReport component.

The screenshot shows a dialog box titled "Connection". At the top, there is a "Name:" text box and a "Type:" dropdown menu currently set to "DBASE (IDAPI)". Below this is a "Data file path:" text box with a "Browse" button underneath it. In the bottom left corner, there is a list box titled "Connections:" containing the items "FINANCE", "ROSTER", and "VIDEO". On the right side, there are four buttons: "New", "Delete", "Save", and "Help". At the bottom center, there are two buttons: "OK" and "Cancel".

The [ConnectionName](#) property identifies the name of the Connection. The [ConnectType](#) property identifies the Type of the Connection. The [DataFilePath](#) property identifies the Data File Path for the Connection (e.g. the path to the tables). The [Database](#) property identifies the Database for the connection (this is only valid for certain connection types). The [UserId](#) property identifies the User Id for

the connection (this is also only valid for certain connection types). Finally, the Password identifies the Password.

The TInstallRSConnection component is actually a stand-alone component. However, you can make the TInstallRSConnection component part of an installation by setting the InstallComponent property to any TInstall component on the form. If this component is linked to a TInstall component, then the TInstall component will automatically install the ReportSmith Connection, when its Install method is called. If you specify a file group in the FileGroup property and you have specified a value for the InstallComponent property, then the ReportSmith Connection will only be installed if the file group's Included property is true. If the component is not linked to a TInstall component, execute the Install method to install the Program Item.



## TInstallFileGroupsDlg Component



[Properties](#)



[Methods](#)

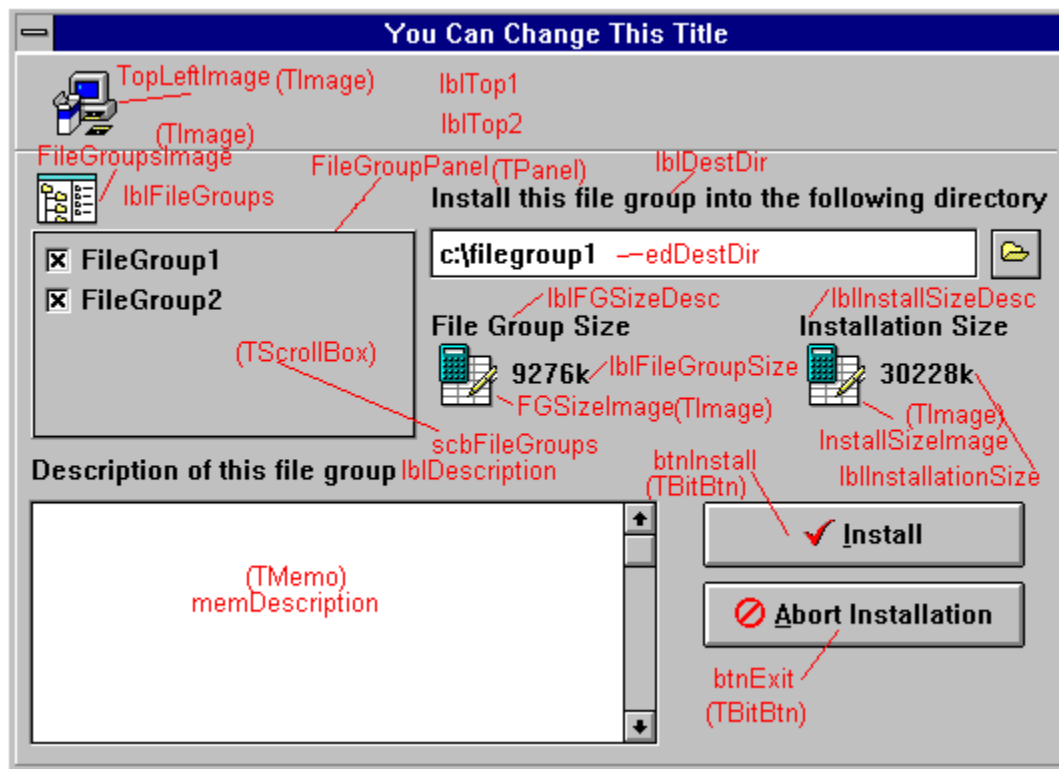
[Events](#)

**Unit**

IdFg

### Description

This component should be used in conjunction with a [TInstall](#) component with the [InstallComponent](#) property. Its main purpose is to allow the users the ability to change installation settings for the various file groups in your installation. You should call the [Execute](#) method in the [ChooseFileGroups](#) event of the TInstall component. The dialog that will be seen by the user is shown below. Note: You do NOT have to include this into your installation if you do not want this functionality.



You may change how the dialog looks at run-time by using the [DialogForm](#) property. This property references the form that will be seen by the user. You can reference the various components contained by their names, shown in the previous schematic. To change the text string "Install this file group into the following directory", you would execute the following code at run-time:

DialogForm.IblDestDir.Caption := 'Destination Directory';

You can use the Title property to change the caption of the dialog. The OnDestinationDirBtnClick event is invoked when the user clicks on the file folder. The OnFileGroupClick event is invoked when the user clicks on a file group. The OnMouseOverFileGroup event is invoked when the user moves the mouse over a file group checkbox. You can get access to the check boxes with the CheckBoxes property.



## TSelectWindowGroupDlg Component



[Properties](#)



[Methods](#)

[Events](#)

### Unit

IdWg

### Description

This component should be used in conjunction with a [TInstall](#) component with the [InstallComponent](#) property. Its main purpose is to allow the users the ability to change select the Window's group into which any Program Items will be installed ([TProgramItem](#)). You should call the [Execute](#) method in the [StartInstallingPMItems](#) event of the TInstall component. The dialog that will be seen by the user is shown below. Note: You do NOT have to include this into your installation if you do not want this functionality.



You may change how the dialog looks at run-time by using the [DialogForm](#) property. This property references the form that will be seen by the user. You can reference the various components contained by their names, shown in the previous schematic. To change the text string "A set of icons will be added to your", you would execute the following code at run-time:

```
DialogForm.Label1.Caption := 'Destination Directory';
```

You can use the [Title](#) property to change the caption of the dialog.

You also have the ability to add two custom buttons at design-time with the [CustomButton1Caption](#) and [CustomButton2Caption](#) properties by assigning them a value. The [OnCustomButton1Click](#) event will be triggered when the user clicks the first custom button and the [OnCustomButton2Click](#) event will be triggered when the user clicks the second custom button. The [ProgramManagerError](#) occurs when Program Manager was unable to be "linked" with.



## TInstallRSConnectionDlg Component



[Properties](#)



[Methods](#)

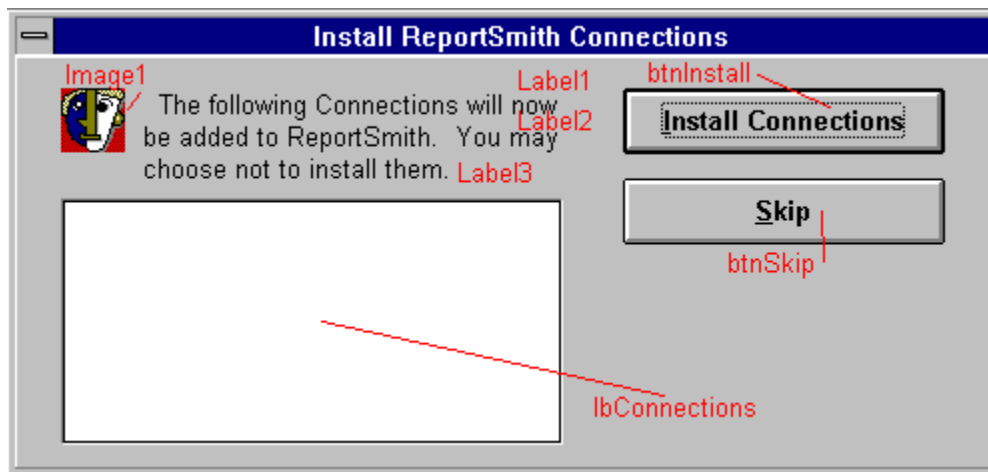
[Events](#)

### Unit

IdRs

### Description

This component should be used in conjunction with a [TInstall](#) component with the [InstallComponent](#) property. Its main purpose is to notify the users that ReportSmith Connections are going to be installed, and allow them to skip the installation of the Connections ([TInstallRSConnection](#)). You should call the [Execute](#) method in the [StartInstallingRSConns](#) event of the TInstall component. The dialog that will be seen by the user is shown below. Note: You do NOT have to include this into your installation if you do not want this functionality.



You may change how the dialog looks at run-time by using the [DialogForm](#) property. This property references the form that will be seen by the user. You can reference the various components contained by their names, shown in the previous schematic. To change the text string "The following Connections will now", you would execute the following code at run-time:

```
DialogForm.Label1.Caption := 'These are the connections that will';
```

You can use the [Title](#) property to change the caption of the dialog.





## Properties

AliasName

Driver

FileGroup

InstallComponent

Path

SQLParams



## Properties

|                         |                          |                     |
|-------------------------|--------------------------|---------------------|
| <u>AliasCount</u>       | <u>ProgramItemCount</u>  | <u>CompressType</u> |
| <u>Aliases</u>          | <u>ProgramItems</u>      |                     |
| <u>CopyingFile</u>      | <u>RSConnectionCount</u> |                     |
| <u>CopyingFileGroup</u> | <u>RSConnections</u>     |                     |
| <u>FileGroupCount</u>   | <u>Size</u>              |                     |
| <u>FileGroups</u>       | <u>SourceDir</u>         |                     |
| <u>INIFileCount</u>     | <u>TotalBytesCopied</u>  |                     |
| <u>INIFiles</u>         | <u>WindowGroupName</u>   |                     |



## Properties

AlwaysInstall

AllowChangeDir

Description

DestDir

FileCount

Files

GroupName

InstallComponent

Include

Size



## Properties

FileGroup

FileName

IncludeInstallInfo

InstallComponent

Lines



# Properties

CommandLine

Description

Directory

FileGroup

IconFile

IconIndex

InstallComponent

WindowGroup



## Properties

ConnectionName

ConnectType

Database

DataFilePath

FileGroup

InstallComponent

OverwriteDuplicate

Password

UserId



## Properties

CustomButton1

CustomButton2

CustomButton1Caption

CustomButton2Caption

DialogForm

InstallComponent

Title



## Properties

ConnectionNames

CustomButton1

CustomButton1Caption

CustomButton2

CustomButton2Caption

InstallComponent

Title





## Properties

CheckBoxes

DialogForm

InstallComponent

Title



## Methods

FileGroupByName

Install

InsertAlias

InsertFileGroup

InsertINIFile

InsertProgramItem

InsertRSConnection

RemoveAlias

RemoveFileGroup

RemoveINIFile

RemoveProgramItem

RemoveRSConnection



## Methods

[ExpandPaths](#)

[Install](#)



## Methods

[ClearFiles](#)

[FileByName](#)

[IndexOf](#)

[InsertFile](#)

[RemoveFile](#)



## Methods

[ExpandPaths](#)

[Install](#)



## Methods

[ExpandPaths](#)

[Install](#)



## Methods

[ExpandPaths](#)

[Install](#)



## Methods

Execute

GetGroups





## Methods

Execute



## Methods

Execute

## Events

|                           |                                    |                                |
|---------------------------|------------------------------------|--------------------------------|
| <u>AfterInstallAlias</u>  | <u>BeforeInstallPI</u>             | <u>OnDoneCopyingFiles</u>      |
| <u>AfterInstallFile</u>   | <u>BeforeInstallRSC</u>            | <u>OnNeedChangeDisk</u>        |
| <u>AfterInstallINI</u>    | <u>ChooseFileGroups</u>            | <u>ProgramManagerError</u>     |
| <u>AfterInstallPI</u>     | <u>DoneInstallingAliases</u>       | <u>StartInstallingAliases</u>  |
| <u>AfterInstallRSC</u>    | <u>DoneInstallingINIFiles</u>      | <u>StartInstallingINIFiles</u> |
| <u>AliasExists</u>        | <u>DoneInstallingPMItems</u>       | <u>StartInstallingPMItems</u>  |
| <u>BeforeInstallAlias</u> | <u>DoneInstallingRSConnections</u> | <u>StartInstallingRSConns</u>  |
| <u>BeforeInstallFile</u>  | <u>OnCopyingFileEvent</u>          | <u>BadInstallation</u>         |
| <u>BeforeInstallINI</u>   | <u>OnCannotOpenBDE</u>             | <u>GoodInstallation</u>        |

## Events

AfterInstall

AliasExists

BeforeInstall

CannotOpenBDE

## Events

# Events

AfterInstall

BeforeInstall

## Events

AfterInstall

BeforeInstall

ProgramManagerError

# Events

AfterInstall

BeforeInstall



## Events

[OnCustomButton1Click](#)

[OnCustomButton2Click](#)

[OnProgramManagerError](#)

## Events

OnCustomButton1Click

OnCustomButton2Click

## Events

[OnDestinationDirBtnClick](#)

[OnFileGroupClick](#)

[OnInitializeDialog](#)

[OnMouseOverFileGroup](#)

# AliasName Property

## Applies To

TInstallAlias

## Declaration

**property** AliasName: String;

## Description

This property determines the name of the alias, as it will appear in the Borland Database Engine.

# PropertyScreen

## Applies To

TSelectWindowGroupDlg, TInstallRSConnectionDlg

## Declaration

property xx

## Description

xx

# Driver Property

## Applies To

TInstallAlias

## Declaration

**property** Driver: String;

## Description

This property determines the driver for the alias. You cannot specify a value for the SQLParams property if the alias is a Standard alias. If the alias is an InterBase alias, then you cannot specify a value for the Path property.

## Type Drivers:

DBASE

ASCII

PARADOX

INTRBASE

# Path Property

## Applies To

TInstallAlias

## Declaration

**property** Path: String;

## Description

This property is only valid for Standard aliases (refer to the Borland Database Engine documentation for more information). Any meta-variables will be expanded when the Install method is called.



**Supports Meta-Variables**

# IconIndex Property

## Applies To

TProgramItem

## Declaration

**property** IconIndex: String;

## Description

Specifies the offset of the image in the file, as specified by the IconFile property, that will be the icon for the Program Item.



# SQLParams Property

## Applies To

TInstallAlias

## Declaration

**property** Path: String;

## Description

This property is only valid for InterBase aliases (refer to the Borland Database Engine documentation for more information). Any meta-variables will be expanded when the Install method is called.

This is a list defined as follows: "AliasOption: Option Data[:AliasOption: Option Data][;...]"

Example:

SERVER NAME: c:\iblocal\examples\employee.gdb;USER NAME: sysdba



**Supports Meta-Variables**

# Install Method

## Applies To

TInstallAlias

## Declaration

**procedure** Install; Virtual;

## Description

This method first calls the ExpandPaths method and then installs the alias into the Borland Database Engine(BDE). The BeforeInstall event is triggered before this method does anything. If there is already an alias with the name as specified in the AliasName property, then the AliasExists event will be triggered. If there is a problem with connecting to the BDE, then the CannotOpenBDE event will be triggered. Finally, if everything goes well, the AfterInstall event will be triggered.

# BeforeInstall Event

## Applies To

TInstallAlias, TInstallFileGroup, TInstallINIFile, TProgramItem, TInstallRSConnection, TInstallFile

## Declaration

**property** AfterInstall: TNotifyEvent;

## Description

This event is triggered before any code gets executed in the Install method of the component. To abort the operation of installing the component, execute the Abort command.

# AliasExists Event

## Applies To

TInstallAlias

## Declaration

**property** AliasExists: TNotifyEvent;

## Description

This event occurs if the alias name, as specified by the AliasName property, is already in the Borland Database Engine.

# CannotOpenBDE Event

## Applies To

TInstallAlias

## Declaration

**property** CannotOpenBDE: TNotifyEvent;

## Description

This event occurs if there is trouble with connecting to the Borland Database Engine. Usually there is trouble if the user does not have the BDE installed on his/her computer.

# InstallComponent Property

## Applies To

TInstallAlias, TInstallFileGroup, TInstallINIFile, TProgramItem, TInstallRSConnection,

(Dialog components)

TInstallRSConnectionDlg, TSelectWindowGroupDlg, TInstallFileGroupsDlg, TCopyFileDlg,

TComponentConflictDlg

## Declaration

**property** InstallComponent: TInstall;

## Description

Components that have this property are components that can run by themselves. However, this property allows you to use the component in conjunction with any TInstall component on the form. I.e. the Install component will essentially install the component for you when the time is right (Except for the dialog components, in which case you should call their execute methods in the appropriate event in the TInstall). If you have specified a value for both this property and the FileGroup property, then the component will only be installed if the file group's Include property is True (N/A for the dialog components). Note: you can selectively allow the user to install/not install file groups. If the user does not wish to install a file group, then the Include property should be set to false.

# FileGroup Property

## Applies To

TInstallAlias, TInstallFileGroup, TInstallINIFile, TProgramItem, TInstallRSConnection, TInstallFile

## Declaration

**property** FileGroup: TInstallFileGroup;

## Description

If you specify a value for both this property and the InstallComponent property, then the component will only be installed, by the TInstall component, if the file group's Included property is true when the TInstall component is run. You must also specify a value for this property if you plan to use the %f meta-variable in any of the properties for the component that support meta-variables.

# ExpandPaths Method

## Applies To

TInstallAlias, TInstallFileGroup, TInstallINIFile, TProgramItem, TInstallRSConnection

## Declaration

**procedure** ExpandPaths; virtual;

## Description

This method expands the meta-variables that are embedded in various properties for the component.



### TInstallAlias

Any meta-variables in the Path or SQLParams properties will be expanded.



### TInstallINIFile

Any meta-variables in the FileName of Lines properties will be expanded.



### TProgramItem

Any meta-variables in the CommandLine, Directory, or IconFile properties will be expanded.



### TInstallRSConnection

Any meta-variables in the DataFilePath, Database,Server, Blahh.. Blah.. will be expanded.

## Example:

|                                  |   |                          |
|----------------------------------|---|--------------------------|
| Path (with meta-variables)<br>%w | Expanded<br>Windows directory. Assume<br>(c:\window's) on users<br>machine.                         | c:\windows               |
| %s\foobar                        | Window's sytem directory.<br>Assume (d:\windows\system)<br>on user's machine.                       | c:\windows\system\foobar |
| %f\new                           | Assume this alias is linked to<br>a file group and that file group<br>gets installed to c:\foo\bar. | c:\foo\bar\new           |



# CommandLine Property

## Applies To

TProgramItem

## Declaration

**property** CommandLine: String;

## Description

Specifies the Command Line for the Program Item.



**Supports Meta-Variables**

# Description Property

## Applies To

TProgramItem

## Declaration

**property** Description: String;

## Description

Specifies the description of the Program Item, as it appears in the Program Manager.

# Directory Property

## Applies To

TProgramItem

## Declaration

`property` Directory: String;

## Description

Specifies the directory that the Program Item will run from.



**Supports Meta-Variables**

# IconFile Property

## Applies To

TProgramItem

## Declaration

**property** IconFile: String;

## Description

Specifies the file that will contain the icon for the Program Item.



**Supports Meta-Variables**

# WindowGroup Property

## Applies To

TProgramItem

## Declaration

**property** WindowGroup: String;

## Description

Specifies the Window's group in which to install the Program Item.

# Install Method

## Applies To

TProgramItem

## Declaration

**procedure** Install: Virtual;

## Description

Executes the ExpandPaths method and then installs the Program Item into the Window's group specified by the WindowGroup property. Before any code is executed, the BeforeInstall event is triggered. You may quit the installation of the Program Item by executing the Abort command in this event. The ProgramManagerError event is triggered if Program Manager cannot be found. The AfterInstall event is triggered if there are no glitches and the Program Item has been installed correctly.

# AfterInstall Event

## Applies To

TInstallAlias, TInstallFileGroup, TInstallINIFile, TProgramItem, TInstallRSConnection, TInstallFile

## Declaration

**property** AfterInstall: TNotifyEvent;

## Description

This event occurs if everything goes well with the installation of the component.

# ProgramManagerError Event

## Applies To

TProgramItem

## Declaration

**property** ProgramManagerError: TNotifyEvent;

## Description

This method is triggered if there is an error with trying to establish a link with Program Manager.



# ConnectionString Property

## Applies To

TInstallRSConnection

## Declaration

**property** ConnectionName: String;

## Description

Name of the ReportSmith connection.

# ConnectType Property

## Applies To

TInstallRSConnection

## Declaration

**property** ConnectType: TConnectType;

## Description

Type of the ReportSmith Connection. Possible values are:

DBASE\_IDAPI  
DBASE\_WINDOWS  
IBLOCAL\_IDAPI  
INTERBASE\_ODBC  
PARADOX\_IDAPI  
RSBTRIEVE\_ODBC  
RSDBASE\_ODBC  
RSEXCEL\_ODBC  
RSFOXPRO\_ODBC  
RSMACCESS\_ODBC  
RSPARADOX\_ODBC  
RSTEXT\_ODBC.

# Database Property

## Applies To

TInstallRSConnection

## Declaration

**property** Database: String;

## Description

Name of the ReportSmith database.



**Supports Meta-Variables**

# DataFilePath Property

## Applies To

TInstallRSConnection

## Declaration

**property** DataFilePath: String;;

## Description

Path to the data files for the ReportSmith Connection..



**Supports Meta-Variables**

# OverwriteDuplicates Property

## Applies To

TInstallRSConnection

## Declaration

**property** OverwriteDuplicates: Boolean;

## Description

If set to true, any ReportSmith connections that have the same name will be overwritten. If set to false and a Connection with the same name exists, the ReportSmith connection will not be installed, and the AfterInstall event will NOT be triggered.

# Password Property

## Applies To

TInstallRSConnection

## Declaration

**property** Password: String;

## Description

Password to the connection.

# Userld Property

## Applies To

TInstallRSConnection

## Declaration

**property** Userld: String;

## Description

Userld for the ReportSmith connection.

# Install Method

## Applies To

TInstallRSConnection

## Declaration

**procedure** Install; Virtual;

## Description.

Executes the ExpandPaths method and then installs the ReportSmith Connection into ReportSmith. Before any code is executed, the BeforeInstall event is triggered. You may quit the installation of the Connection by executing the Abort command in this event. The AfterInstall event is triggered if there are no glitches and the ReportSmith Connection has been installed correctly. If you have set OverwriteDuplicate to False and a duplicate is encountered, the AfterInstall event will NOT be triggered.



# AlwaysInstall Property

## Applies To

TInstallFileGroup

## Declaration

**property** AlwaysInstall: Boolean;

## Description

If set to True, the file group should always be installed. If set to False, the user should have the option of including/excluding the file group. The TInstallFileGroupsDlg complies with this behavior.

# Description Property

## Applies To

TInstallFileGroup

## Declaration

**property** Description: TStrings;

## Description

This property allows you to attach a description of the file group to the component. The TInstallFileGroupsDlg component uses this property to give a description of the file group to the user. If you design your own Dialogs, you will definitely find this property usefull.

# DestDir Property

## Applies To

[TInstallFileGroup](#)

## Declaration

**property** DestDir: String;

## Description

The DestDir property contains the directory to which the files that are owned by this component, in the Files property, will be installed when the Install method is executed. The [TInstallFileGroupsDlg](#) allows the users to change this at run-time. If you specify a parent file group in the [ParentFileGroup](#) property, then you can use the %f meta-variable, which will be expanded to the path where the parent file group is installed to.



## Supports Meta-Variables

# FileCount Property

## Applies To

TInstallFileGroup

## Declaration

**property** FileCount: Integer;

## Description

This property contains a count of the number of files owned by this file group. The files can be accessed through the Files property.

# FileByName Method

## Applies To

TInstallFileGroup

## Declaration

```
function FileByName(Name: String): TInstallFile; Virtual;
```

## Description

This method returns the file component with the file name of Name. If the file group does not own the file component, Nil will be returned.

# Files Property

## Applies To

TInstallFileGroup

## Declaration

**property** Files[l: Integer]: TInstallFile;

## Description

This property allows you to access the individual files owned by this file group.

# GroupName Property

## Applies To

TInstallFileGroup

## Declaration

**property** GroupName: String;

## Description

Attach a descriptive name of the file group with this property. The TInstallFileGroupsDlg uses this property to display the name of the file group to the user. If you create your own Dialogs, you should use this property instead of the Name property.

# Include Property

## Applies To

TInstallFileGroup

## Declaration

**property** Include: Boolean;;

## Description

If set to True, the file group should be slated for installation. However, if you set the AlwaysInstall property to false, the user should be able to exclude/include this file group from the installation. The TInstallFileGroupsDlg component complies with this behavior.



# Size Property

## Applies To

TInstallFileGroup

## Declaration

**property** Size: LongInt;

## Description

This property contains the size of the file group in Bytes. This property is calculated by summing the size of the all the files owned by this file group, which can be accessed through the Files property.

# ClearFiles Method

## Applies To

TInstallFileGroup

## Declaration

**procedure** ClearFiles; virtual;

## Description

This method removes all files that are owned by the file group.

# IndexOf Method

## Applies To

TInstallFileGroup

## Declaration

```
function IndexOf(InstallFile: TInstallFile): Integer; virtual;
```

## Description

This method returns the index into the Files property of the file component specified by the InstallFile parameter.

# InsertFile Method

## Applies To

[TInstallFileGroup](#)

## Declaration

**procedure** InsertFile(Value: [TInstallFile](#)); virtual;

## Description

This method adds the file component to the list of files owned by the file group (see [Files](#) property).

# RemoveFile Method

## Applies To

TInstallFileGroup

## Declaration

**procedure** RemoveFile(Value: TInstallFile);

## Description

This method removes the file component in the Value parameter, that is owned by this file group.

# Install Method

## Applies To

TInstallINIFile

## Declaration

**method** Install; virtual;

## Description

This method calls the ExpandPaths method and installs the INI file into the path given in the FileName property. The BeforeInstall event is triggered before any code executes, and the installation can be halted by executing the Abort command. The AfterInstall event is triggered if the INI file is installed correctly and there are no glitches.

# FileName Property

## Applies To

TInstallINIFile

## Declaration

**property** FileName: String;

## Description

The INI file will be written to the path specified in this property. You should include both a path and a file name.



## Supports Meta-Variables

# Lines Property

## Applies To

TInstallINIFile

## Declaration

`property` Lines: TStrings;

## Description

This is the text that will be written to the INI file when it is installed.



## Supports Meta-Variables



# DialogForm Property

## Applies To

TInstallFileGroupsDlg, TSelectWindowGroupDlg, TInstallRSConnectionDlg, TCopyFileDlg,  
TGoodInstallationDlg, TComponentConflictDlg

## Declaration

**property** DialogForm: TForm;

## Description

This run-time only property gives you access to the form that is displayed to the user. Basically, at run-time you can change how the dialog form looks by referencing the components on the form through this property.

TInstallFileGroupsDlg Examples:

To change the image at the top left (the little setup image), you would execute the following code (of course you will have to include the .bmp file in your installation):

```
DialogForm.TopLeftImage.LoadFromFile('newimage.bmp');
```

To change what happens when the user clicks the Abort Installation button:

```
Form1 = class(TForm)
```

```
...
```

```
public
```

```
    MyClickMethod(Sender: TObject);
```

```
...
```

```
procedure Form1.MyClickMethod(Sender: TObject)
```

```
begin
```

```
    ShowMessage('I cannot believe you canceled this beautiful installation.');
```

```
    ModalResult := mrCancel;
```

```
end;
```

```
DialogForm.btnExit.OnClick := MyClickMethod;
```

TSelectWindowGroupDlg Examples:

```
DialogForm.cbGroups.Text := 'NewerGroup';
```

# Title Property

## Applies To

TInstallFileGroupsDlg, TSelectWindowGroupDlg, TGoodInstallationDlg, TCopyFileDlg, TComponentConflictDlg

## Declaration

**property** Title: String;

## Description

You can change the Caption for the dialog form that the user sees with this property.

# Execute Method

## Applies To

TInstallFileGroupsDlg

## Declaration

**function** Execute: Boolean; virtual;

## Description

This method displays the dialog form. It returns true if the user clicked the Install button and False if the user clicks the Abort Installation button. If hooked up to a TInstall component, the Installation will automatically be aborted if the user hits the Abort Installation button.

# DestinationDirBtnClick Event

## Applies To

TInstallFileGroupsDlg

## Declaration

**property** DestinationDirBtnClick: TFileGroupsDlgNotifyEvent;

## Description

This event occurs when the user clicks the file folder.

# FileGroupsDlgNotifyEvent Type

## Unit

IdFg

## Applies To

TInstallFileGroupsDlg

## Declaration

### type

```
TFileGroupsDlgNotifyEvent = procedure(Sender: TObject;  
    CurrentFileGroup: TInstallFileGroup);
```

## Description

This event type returns the Sender and also the currently selected file group.

# OnFileGroupClick Event

## Applies To

TInstallFileGroupsDlg

## Declaration

**property** OnFileGroupClick: TFileGroupsDlgNotifyEvent;

## Description

This event occurs when the user clicks on a file group checkbox.

# OnMouseOverFileGroup Event

## Applies To

TInstallFileGroupsDlg

## Declaration

**property** OnMouseOverFileGroup: TFileGroupsDlgNotifyEvent;

## Description

This event occurs when the user moves the mouse over a file group checkbox.

# CustomButton1Caption Property

## Applies To

TSelectWindowGroupDlg, TInstallRSConnectionDlg

## Declaration

**property** CustomButton1Caption: String;

## Description

Assign a non-blank string to this property to provide the user with a custom button on the dialog form. The OnCustomButton1Click event will be triggered if the user clicks on this button.



# CustomButton2Caption Property

## Applies To

TSelectWindowGroupDlg, TInstallRSConnectionDlg

## Declaration

**property** CustomButton2Caption: String;

## Description

Assign a non-blank value to this property to give the user a second custom button. The OnCustomButton2Click event will be triggered if the user clicks the second custom button.

# Execute Method

## Applies To

TSelectWindowGroupDlg

## Declaration

**function** Execute: Boolean; virtual;

## Description

This method executes the select window group dialog. The WindowGroup property will hold the user's choice of window group. The function returns False if the user hit the Skip button and True if the user clicked the Install Icons button. If this component is hooked up to an Install Component through the InstallComponent property, then the TInstall's WindowGroupName property will contain the value of the window group that the user selected.

# GetGroups Method

## Applies To

TSelectWindowGroupDlg

## Declaration

```
procedure GetGroups(Value: TStrings);
```

## Description

The Value parameter will be cleared and a list of all the Window Group names will be in the Value parameter.

# CustomButton1 Property

## Applies To

TSelectWindowGroupDlg, TInstallRSConnectionDlg

## Declaration

**property** CustomButton1: TBitBtn;

## Description

Provides a run-time reference to the Custom Button #1.

# CustomButton2 Property

## Applies To

TSelectWindowGroupDlg, TInstallRSConnectionDlg

## Declaration

**property** CustomButton2: TBitBtn;

## Description

Provides a run-time reference to the Custom Button #2.

# OnCustomButton1Click Event

## Applies To

TSelectWindowGroupDlg, TInstallIRSConnectionDlg

## Declaration

**property** OnCustomButton1Click: TNotifyEvent;

## Description

Occurs when the user clicks on Custom Button #1.

.

# OnCustomButton2Click Event

## Applies To

TSelectWindowGroupDlg, TInstallIRSConnectionDlg

## Declaration

**property** OnCustomButton2Click: TNotifyEvent;

## Description

Occurs when the user clicks on Custom Button #2.

.

# OnProgramManagerError Event

## Applies To

TSelectWindowGroupDlg

## Declaration

**property** OnProgramManagerError: TNotifyEvent;

## Description

Occurs when there is a problem with "linking" to Program Manager.



# BeforeInstallFile Event

## Applies To

TInstall

## Declaration

**property** BeforeInstallFile: TInstallFileNotifyEvent;

## Description

This event is called before a File has been installed. Execute the Abort command to abort the installation of the file.

# Execute Method

## Applies To

TInstallRSConnectionDlg

## Declaration

**function** Execute: Boolean; virtual;;

## Description

If you have assigned a value to the InstallComponent property, then when this component is executed and the user hits the Skip button, the ReportSmith connections will not be installed. This method returns True if the user hits the Install Connections Button and False if the user hits the Skip button.

# ConnectionNames Property

## Applies To

TInstallRSConnectionDlg

## Declaration

**property** ConnectionNames: TStrings;

## Description

If you have assigned a value to the InstallComponent property, then when this component is executed, this property will contain the names of the ReportSmith Connections. If this component is not linked to a TInstall component, then use this property manually create a list of ReportSmith connections that will be installed. You will normally want to use this in conjunction with a TInstall component.

# AfterInstallRSC Event

## Applies To

TInstall

## Declaration

**property** AfterInstallRSC: TInstallRSCConnectionNotifyEvent;

## Description

This event is called after a ReportSmith connection has been installed.

# AfterInstallAlias Event

## Applies To

TInstall

## Declaration

**property** AfterInstallAlias: TInstallAliasNotifyEvent;

## Description

This event is called after an Alias has been installed.

# AfterInstallFile Event

## Applies To

TInstall

## Declaration

**property** AfterInstallFile: TInstallFileNotifyEvent;

## Description

This event is called after an File has been installed.

# AfterInstallINI Event

## Applies To

TInstall

## Declaration

**property** AfterInstallINI: TInstallININotifyEvent;

## Description

This event is called after an INI File has been installed.

# AfterInstallPI Event

## Applies To

TInstall

## Declaration

**property** AfterInstallPI: TProgramItemNotifyEvent;

## Description

This event is called after a Program Item has been installed.



# AliasExists Event

## Applies To

TInstall

## Declaration

**property** AliasExists: TInstallAliasNotifyEvent;

## Description

This event is called when there is an alias that already exists in the BDE on the user's machine. The alias will not be installed.

# BeforeInstallAlias Event

## Applies To

TInstall

## Declaration

**property** BeforeInstallAlias: TInstallAliasNotifyEvent;

## Description

This event is called before an Alias has been installed. Execute the Abort command to abort the installation of the alias.

# BeforeInstallINI Event

## Applies To

TInstall

## Declaration

**property** BeforeInstallINI: TInstallINIFileNotifyEvent;

## Description

This event is called before an INI File has been installed. Execute the Abort command to abort the installation of the INI File.

# BeforeInstallPI Event

## Applies To

TInstall

## Declaration

**property** BeforeInstallPI: TProgramItemNotifyEvent;

## Description

This event is called before a Program Item has been installed. Execute the Abort command to abort the installation of the Program Item.

# BeforeInstallRSC Event

## Applies To

TInstall

## Declaration

**property** BeforeInstallRSC: TInstallRSCConnectionNotifyEvent;

## Description

This event is called before a ReportSmith Connection has been installed. Execute the Abort command to abort the installation of the ReportSmith Connection.

# ChooseFileGroups Event

## Applies To

TInstall

## Declaration

**property** ChooseFileGroups: TNotifyEvent;

## Description

If you are going to use a TInstallFileGroupsDlg component in conjunction with this component, then you need to call the Execute method of the TInstallFileGroupsDlg component in this event.

```
InstallFileGroupsDlg1.Execute;
```

# DoneInstallingAliases Event

## Applies To

TInstall

## Declaration

**property** DoneInstallingAliases: TNotifyEvent;

## Description

This event is called after all the aliases have been installed.

# DoneInstallingINIFiles Event

## Applies To

TInstall

## Declaration

**property** DoneInstallingINIFiles: TNotifyEvent;

## Description

This event is called after all the INI Files have been installed.



# DoneInstallingPMItems Event

## Applies To

TInstall

## Declaration

**property** DoneInstallingPMItems: TNotifyEvent;

## Description

This event is called after all the Program Items have been installed.

# AliasCount Property

## Applies To

Install

## Declaration

**property** AliasCount: Integer;

## Description

Run-time only. This property contains the number of aliases in the Aliases property.

# DoneInstallingRSConnections Event

## Applies To

TInstall

## Declaration

**property** DoneInstallingRSConnections: TNotifyEvent;

## Description

This event is called after all the ReportSmith Connections have been installed.

# OnCopyingFileEvent Event

## Applies To

TInstall

## Declaration

**property** OnCopyingFileEvent: TCopyFileNotifyEvent;

## Type

TCopyFileNotifyEvent = procedure(Sender: TObject;InstallFile: TInstallFile) of object;

## Description

If you are going to use the TCopyFileDlg in conjunction with this component, then you should call the Update method of the TCopyFileDlg component in this event.

CopyFileDlg1.Update;

# OnCannotOpenBDE Event

## Applies To

TInstall

## Declaration

**property** OnCannotOpenBDE: TNotifyEvent;

## Description

This event is called when the Borland Database Engine cannot be found.

# OnDoneCopyingFiles Event

## Applies To

TInstall

## Declaration

**property** OnDoneCopyingFiles: TNotifyEvent;

## Description

This event is called after copying the files (successfully or non-successfully). If you are using a TCopyFileDlg component in conjunction with this component, you should call the TCopyFileDlg component's Hide method to hide the Copy File Dialog Box.

CopyFileDlg1.Hide;.

If you have a TComponentConflictDlg component that you are using, you should call its Execute method in this event.

# ProgramManagerError Event

## Applies To

TInstall

## Declaration

**property** ProgramManagerError: TNotifyEvent;

## Description

This event is called when Program Manager is not up and running. This event occurs each time a Program Item is installing itself and cannot link to the Program Manager.

# StartInstallingAliases Event

## Applies To

TInstall

## Declaration

**property** StartInstallingAliases: TNotifyEvent;

## Description

This event is called before the aliases are to be installed. Call the Abort procedure to abort installing the aliases.



# StartInstallingINIFiles Event

## Applies To

TInstall

## Declaration

**property** StartInstallingAliases: TNotifyEvent;

## Description

This event is called before the INI Files are to be installed. Call the Abort procedure to abort installing the INI Files.

# StartInstallingPMItems Event

## Applies To

TInstall

## Declaration

**property** StartInstallingPMItems: TNotifyEvent;

## Description

This event is called before the Program Items are to be installed. Call the Abort procedure to abort installing the Program Items. If you want to use the TSelectWindowGroupDlg component in conjunction with this component, then you will need to call the Execute method of the TSelectWindowGroupDlg component in this event.

SelectWindowGroupDlg1.Execute;

# StartInstallingRSConns Event

## Applies To

TInstall

## Declaration

**property** StartInstallingRSConns: TNotifyEvent;

## Description

This event is called before the ReportSmith connections are to be installed. Call the Abort procedure to abort installing the ReportSmith connections. If you want to use the TInstallRSConnectionDlg with this component, then you will want to call the Execute method of the TInstallRSConnectionDlg component in this event.

InstallRSConnectionDlg1.Execute;

# Aliases Property

## Applies To

TInstall

## Declaration

**property** Aliases[: Integer]: TInstallAlias;

## Description

Run-time only. This property contains references to all the aliases in the installation.

# CopyingFile Property

## Applies To

TInstall

## Declaration

**property** CopyingFile: TInstallFile;

## Description

Run-time only. This property contains a reference to the file that is currently being copied. It is Nil if there is no file being copied.

# CopyingFileGroup Property

## Applies To

TInstall

## Declaration

**property** CopyingFileGroup: TInstallFileGroup;

## Description

Run-time only. This property contains a reference to the file group that is currently being copied. It is Nil if there is no file group is being copied.

# FileGroupCount Property

## Applies To

TInstall

## Declaration

**property** FileGroupCount: Integer;

## Description

Run-time only. This property contains the number of file groups contained in the FileGroups property.

# FileGroups Property

## Applies To

TInstall

## Declaration

**property** FileGroups[!: Integer]: TInstallFileGroup;

## Description

Run-time only. This property contains references to all the file groups in the installation.



# INIFiles Property

## Applies To

TInstall

## Declaration

**property** INIFiles[l: Integer]: TInstallINIFiles;

## Description

Run-time only. This property contains references to all the INI Files in the installation.

# ProgramItems Property

## Applies To

TInstall

## Declaration

**property** ProgramItems[!: Integer]: TProgramItem;

## Description

Run-time only. This property contains references to all the Program Items in the installation.

# RSConnections Property

## Applies To

TInstall

## Declaration

**property** RSConnections[l: Integer]: TInstallRSConnection;

## Description

Run-time only. This property contains references to all the ReportSmith Connections in the installation.

## WHY TINSTALL?

Besides the obvious: installing files, program items, aliases, ReportSmith Connections, and INI files, and an Installation Builder to build the Installation disks for you, TInstall is powerful because it is a native Delphi VCL component. This means that there are no scripts to learn when you have to do a little something different--And you can change almost anything and everything about how your installation will run.

The TInstall publishes a slew of events for you to provide installation specific behavior--all in native Delphi code! The TInstall also supports polymorphism, inheritance and all the object oriented concepts that makes Delphi so great! For instance, if you want to make a change to the TInstall and you want to use that change in your installations, just make a new class that descends from TInstall, make the changes, and install your new component. You can now plop your component and create your installation. Your component will inherit all the powerful features of the TInstall and will add your own features. You can also do this piecemeal with any of the other components ([TInstallAlias](#), [TInstallFile](#), [TInstallFileGroup](#), [TInstallINIFile](#), [TProgramItem](#), [TInstallRSConnection](#), [TInstallFileGroupsDlg](#), [TInstallRSConnectionDlg](#), [TSelectWindowGroupDlg](#), and [TCopyFileDlg](#)).

# INIFileCount Property

## Applies To

TInstall

## Declaration

**property** INIFileCount: Integer;

## Description

Run-time only. This property contains the number of INI Files in the INIFiles property.

# ProgramItemCount Property

## Applies To

Install

## Declaration

**property** ProgramItemCount: Integer;

## Description

Run-time only. This property contains the number of Program Items in the ProgramItems property.

# RSConnectionCount Property

## Applies To

TInstall

## Declaration

**property** RSConnectionCount: Integer;

## Description

Run-time only. This property contains the number of ReportSmith Connections in the RSConnections property.

# Size Property

## Applies To

Install

## Declaration

**property** Size: Integer;

## Description

Run-time only. This property contains the size of the installation in bytes..



# SourceDir Property

## Applies To

Install

## Declaration

**property** SourceDir: String;

## Description

Run-time only. This property contains the number of the directory from which the user is running the install program.

# TotalBytesCopied Property

## Applies To

Install

## Declaration

**property** TotalBytesCopied: Integer;

## Description

Run-time only. This property contains the total number of bytes currently copied. This is useful for calculating percentages if you decide to make your own Copy File Dialog form.

# WindowGroupName Property

## Applies To

TInstall

## Declaration

**property** WindowGroupName: String;

## Description

This property contains the name of the window's group into which you want to install all of your program items. If you leave this blank, then the program items will be installed into the window groups dictated by the TProgramItem component's WindowGroup property.

# InsertFileGroup Method

## Applies To

TInstall

## Declaration

**procedure** InsertFileGroup(Value: TInstallFileGroup);

## Description

This method inserts a file group into the FileGroups property.

# InsertINIFile Method

## Applies To

TInstall

## Declaration

**procedure** InsertINIFile(Value: TInstallINIFile);

## Description

This method inserts an INI File into the INIFiles property.

# InsertAlias Method

## Applies To

TInstall

## Declaration

**procedure** InsertAlias(Value: TInstallAlias);

## Description

This method inserts an Alias into the Aliases property.

# InsertProgramItem Method

## Applies To

TInstall

## Declaration

**procedure** InsertProgramItem(Value: TProgramItem);

## Description

This method inserts a Program Item into the ProgramItems property.

# InsertRSConnection Method

## Applies To

TInstall

## Declaration

**procedure** InsertRSConnection(Value: TInstallRSConnection);

## Description

This method inserts a ReportSmith Connection into the RSConnections property.



# RemoveAlias Method

## Applies To

TInstall

## Declaration

**procedure** RemoveAlias(Value: TInstallAlias);

## Description

This method removes an alias from the Aliases property.

# RemoveFileGroup Method

## Applies To

TInstall

## Declaration

**procedure** RemoveFileGroup(Value: TInstallFileGroup);

## Description

This method removes a File Group from the FileGroups property.

# RemoveINIFile Method

## Applies To

TInstall

## Declaration

**procedure** RemoveINIFile(Value: TInstallINIFile);

## Description

This method removes an INI File from the INIFiles property.

# RemoveProgramItem Method

## Applies To

TInstall

## Declaration

**procedure** RemoveProgramItem(Value: TProgramItem);

## Description

This method removes a Program Item from the ProgramItems property.

# RemoveRSConnection Method

## Applies To

TInstall

## Declaration

**procedure** RemoveRSConnection(Value: TInstallRSConnection);

## Description

This method removes a ReportSmith Connection from the RSConnections property.

# Install Method

## Applies To

TInstall

## Declaration

**procedure** Install; virtual;

## Description

This method Installs all the Aliases, FileGroups, INI Files, Program Items, and ReportSmith connections associated with this TInstall component.

## **Building The Installation Files**

Double-click on the TInstall component. You should see another dialog box come up. Before you build the installation files, you should remember to save your project and compile it.

## Getting Started--BASIC

1. Open a new project.
2. Place the TInstall on your main form (You can actually put it on any form!).
3. Design your form anyway you want to; put a nice bitmap on the form, place a huge label on the form with the name of your software, whatever you want. Remember, your users will see this form when they crank your installation program up.
4. Think about your installation. You will probably have groups of files that you want to copy to the same directories; All the files that are going to a common directory on the user's computer should be part of a TInstallFileGroup component. So plop down some TInstallFileGroup components and set the properties appropriately (i.e. Remember to set a value for the InstallComponent property).
5. Double-click on the TInstallFileGroup components, one at a time. You should add the appropriate files to each file group in the dialog box.
6. You will probably want to use the TCopyFileDlg to show the user when the files are being copied. So plop down a TCopyFileDlg component onto the form. Next you will want to attach the following code to the OnCopyingFileEvent Event for the TInstall component:

```
CopyFileDlg1.Update;
```

7. To hide the dialog box, you will want to attach the following code to the OnDoneCopyingFiles Event for the TInstall Component.

```
CopyFileDlg1.Hide;
```

8. You probably want to have a button on your form that notifies when to start the install. So, wherever you want to start the installation (maybe in the OnClick event of the button), write the following code:

```
Install1.Install; { Unless of course you named the component something different. }
```

9. Save your project, compile it, and then double-click on the TInstall component. The rest is cake.

### SEE ALSO

Getting Started--Advanced



## Getting Started--Advanced

1. You will probably want the users to be able to include/exclude some of the file groups. I.e. this allows the user to install only pertinent portions of your software. To do this you will probably want to use the [TInstallFileGroupsDlg](#) component. So plop down a [TInstallFileGroupsDlg](#) onto your form (Did you remember to assign a value for the [InstallComponent](#) property?). To use this component, attach the following code to the [ChooseFileGroups](#) Event of the [TInstall](#) component:

```
InstallFileGroupsDlg1.Execute; { Unless of course the component is named something different. }
```

2. If you want to install any Program Items, you should plop some [TProgramItem](#) components down on your form (Of course, you should assign a value for the [InstallComponent](#) property).

3. If you want your users the ability to install the Program Items into a different Window's Group than what you had assigned at Design-Time, plop a [TSelectWindowGroupDlg](#) onto the form and assign a value for the [InstallComponent](#) property. To show this dialog to the users at the appropriate time, you should attach the following code to the [StartInstallingPMItems](#) event of the [TInstall](#) component:

```
SelectWindowsGroupDlg1.Execute; { Unless of course the component is named something different }
```

4. It is the same way if you want to install ReportSmith connections: Plop some [TInstallRSConnection](#) components onto your form. You will probably not want to show a dialog box to the user about installation ReportSmith connections, but if you do, plop down a [TInstallRSConnectionDlg](#) component onto your form and in the [StartInstallingRSConns](#) event of the [TInstall](#) component, attach the following code:

```
InstallRSConnectionDlg1.Execute;
```

SEE ALSO  
[GettingStarted--Basic](#)

# TCopyFileDlg Component



[Properties](#)



[Methods](#)

[Events](#)

## Unit

Install

## Description

You should only use this component in conjunction with a TInstall component. I.e. it is not really a stand-alone component. You connect this component to a TInstall component through the [InstallComponent](#) property. Call the [Update](#) method in the [OnCopyingFileEvent](#) event of the [TInstall](#) component and the [Hide](#) method in the [OnDoneCopyingFiles](#) event of the TInstall component to use this component.

You may change how the dialog looks at run-time by using the [DialogForm](#) property. This property references the form that will be seen by the user. You can reference the various components contained by their names, shown in the previous schematic. To change the text string "Extracting File", you would execute the following code at run-time:

```
DialogForm.IblCopyFileHeader.Caption := 'UnZipping File:';
```

You can use the [Title](#) property to change the caption of the dialog.



## Properties

DialogForm

InstallComponent

Title



## Methods

[Hide](#)

[Update](#)

## Events

# Hide Method

## Applies To

TInstall

## Declaration

**procedure** Hide: virtual;

## Description

This procedure hides the copy file dialog box. You should call this method in the OnDoneCopyingFiles event of the TInstall component.

# Update Method

## Applies To

TInstall

## Declaration

**procedure** Update; virtual;

## Description

This method updates the copy file dialog box (you MUST hook up a TInstall component to the InstallComponent property to use this). You should call this method in the OnCopyingFileEvent of the TInstall Component.

# TInstallFile Component



Properties



Methods

Events

## **Unit**

Install

## **Description**

This component is created for each file that you want in your installation. It stores information about the file itself and the things that you want done to the file when the user runs the installation.



# Size Property

## Applies To

InstallFile

## Declaration

`property` Size: Integer;

## Description

The size of the file in bytes.

# IncludeFromDir Property

## Applies To

TInstallFile

## Declaration

**property** IncludeFromDir: String;

## Description

Specifies where the installation builder will find your file to build the installation disk(s)/file.

# FileName Property

## Applies To

TInstallFile

## Declaration

**property** FileName: String;

## Description

Specifies the name of the file.

# Disk Property

## Applies To

TInstallFile

## Declaration

**property** FileName: String;

## Description

Run-Time Only. Specifies the disk on which the file is found.



## Methods

Create

Install

# Install Method

## Applies To

Install

## Declaration

**procedure** Install; virtual;

## Description

This method installs the file onto the user's computer. It will only be installed if it's corresponding file group's Include property is True.

# Create Method

## Applies To

TInstall

## Declaration

**constructor** Create(AOwner: TComponent; FName: String; FGroup: TInstallFileGroup); virtual;

## Description

This method creates the TInstallFile component. You must have a non-null FGroup parameter and a non-blank FName property. The FileName property will be assigned the value of FName and the FileGroup property will be assigned the value of FGroup..

# Events

AfterInstall

BeforeInstall

OnNeedChangeDisk



# OnNeedChangeDisk Event

## Applies To

TInstallFile

## Declaration

**property** OnNeedChangeDisk: TNotifyEvent;

## Description

This event is called when the file 'files' + DiskNumber + '.zip' cannot be found on the current disk. I.e. if the Disk property is 2, then this event will be triggered if 'files2.zip' cannot be found on the SourceDir path. If you do not attach any code, then a little message box will appear asking the user to enter the appropriate disk.

# OnNeedChangeDisk Event

## Applies To

TInstall

## Declaration

**property** OnNeedChangeDisk: TNeedChangeDiskEvent;

## Type

TNeedChangeDisk = procedure(Sender: TObject;NewDisk: Integer) of Object;

## Description

This event will be triggered when a new disk is needed. The NewDisk parameter is the number of the disk that needs to be inserted. This event is generated if the file 'files' + Disk Number + '.zip' cannot be found on the SourceDir path. I.e. if a file was supposed to be on disk #2, then if 'files2.zip' cannot be found on the SourceDir path, then this event will be generated.

## Installing this software into the Delphi VCL

You should get rid of all occurrences of TInstall on your computer, in your Delphi path and what not (if you have TInstall or an earlier copy of the Youseful components already installed on your computer):

1. Remove the TInstall component from your VCL library path (including the path to the TInstall component).
2. Delete all files associated with the TInstall. You will probably find the help files in the windows directory and the .kwf files in the \delphi\help directory.

Before you can use this software you must install the Youseful components into the Delphi palette and rebuild the component library. Unfortunately there is no programatic way to install components into your VCL library; you must do it manually. From within Delphi, select "Options|Install components" from the main menu, select "Add" from the available buttons at the right of the dialog, enter YOUSEFUL, and select the OK button. Before closing the "Install Components" dialog, append the path name where you installed the Youseful components to to the "Search path" edit control (if it isn't already there). Select OK to have Delphi compile and add the Youseful components to the component library.

If you receive an error that there has been a version mismatch, then you probably have the old TInstall component floating somewhere around your computer. The files inst\*.\* all belong to the TInstall component, so after you delete those (except for any Youseful files), try this procedure again.

## Merging the Youseful help into the Delphi Help System

From the windows Program Manager, run the Delphi Help File Installer program (HelpInst), located in your Delphi Program Group.

1. Click the Open an existing HDX file button or select the file and Open... menu options and open the file DELPHI.HDX in your \delphi\bin directory.
2. Click the Add a new keyword file button or select the Keywords and Add Keyword File... menu options and add the file YOUSEFUL.KWF to the list of keyword files being displayed.
3. Click the Compile and save the current HDX file button or select the File and Save menu options to save and compile the modified HDX file. Depending on your CPU, this process may take up to 30 seconds.
4. Close the Help File Installer window or select the File and Exit menu options to terminate the program.



## TModifiedReport Component



### Properties

**Unit**  
Rpt2

### **Declaration**

```
TModifiedReport = class(TReport);
```

### **Description**

The TModifiedReport allows you to "link" a reportsmith connection to an alias, a database, or a hard-coded path, thus allowing you to run reports on data files in different directories without having to create multiple reports. The TModifiedReport component also allows you to hook to a TReportManager component, which provides a level of abstraction for the location of your reports. E.g. normally you have to specify where your reports are in the ReportDir property, to be able to run a report. This isn't very flexible because there is never any guarantee that your users are going to have their system set up just like yours. Consider this example: you are designing a system for your users and your users have the option of running a report. So when you are designing the system, you set the ReportDir property of all your TReport components to 'c:\app\reports'. When you go to install your system, your users decide that they want your program on the D-drive. Now you have to go back to your office, change the ReportDir property for ALL of the TReport components, re-compile it, and then go back over to your users office. This is even worse if you are releasing software for general consumption. Very frustrating indeed! In essence, the TModifiedReport-TReportManager connection is just like the TTable-TDatabase connection; your TTable components do not have to know anything about where the physical data tables are, just as your TModifiedReport components do not have to know anything about where your reports are physically located.

As mentioned earlier, the TModified report component inherits all of the functionality of the TReport component. This means that you can specify a ReportDir and ReportName and the TModifiedReport component will behave just like the regular TReport component. However, you will probably want to use the full power of the component. To do this, you will want to specify to which ReportSmith connection you want to "link". You do this by changing the value of the ConnectionName property. After that, you will want to either specify a hard-coded path, database, or alias with which the ReportSmith connection will be "linked". To specify a hard-coded path, change the value of the NewPath property. To specify a database or alias change the value of the DatabaseName property. Finally, you can specify a ReportManager by setting the ReportManager property.

In essence, you can have BDE/IDAPI aliases in ReportSmith by creating ReportSmith connections and then using this component. At run-time, the ReportSmith connection's path will be replaced by the same as the BDE/IDAPI aliases path.

**Because the TModifiedReport component is a descendant of the TReport component, it inherits all the methods and properties of the TReport component. I.e. you would set all the other properties how you normally set them and then call the Run method.**

# ConnectionString Property

## Applies To

TModifiedReport

## Declaration

**property** ConnectionName: String;

## Description

The ConnectionName property allows you to specify the connection with which you want to "link". Specifying a value for the DatabaseName property along with this property, it is possible to use BDE/IDAPI aliases in ReportSmith. At run-time the ReportSmith connection's path will be replaced by the BDE/IDAPI (Databases's/aliases's) path.

# NewPath property

## Applies To

TModifiedReport

## Declaration

**property** NewPath: String;

## Description

The NewPath property allows you to specify a hard-coded path with which to connect the ReportSmith connection that you specified in the ConnectionName property. At run-time the ReportSmith connection's path will be replaced by the value of this property if it is non-blank path.

# DatabaseName property

## Applies To

TModifiedReport

## Declaration

**property** Database: TSymbolStr;

## Description

The Database property allows you to specify a database or alias with which to connect the ReportSmith connection that you specified in the ConnectionName property. Specifying a value for the ConnectionName property along with this property, it is possible to use BDE/IDAPI aliases in ReportSmith. At run-time the ReportSmith connection's path will be replaced by the BDE/IDAPI (Database's/aliases's) path.





## Properties

ConnectionName

DatabaseName

NewPath

ReportManager

# ReportManager Property

## Applies To

TModifiedReport

## Declaration

**property** ReportManager: String;

## Description

The ReportManager property provides a level of abstraction for the location of your reports. E.g. normally you would have to specify the ReportDir property to be able to run a report. This isn't very flexible because there is never any guarantee that your users are going to have their system setup just like yours. Consider this example: you are designing a system for your users and your users have the option of running a report. So when you are designing the system, you set the ReportDir property of all your TReport components to c:\app\reports. When you go to install your system, your users decide that they want your program on the D-drive. Now you have to go back to your work, change the ReportDir property for ALL of the TReport components, re-compile it, and then go back over to your users office. Very frustrating indeed! In essence, the TModifiedReport-TReportManager connection is just the TTable-TDatabase connection; just as your TTable components do not have to know anything about where the physical data tables are, your TModifiedReport components do not have to know anything about where your reports are physically located.



## Properties

Directory

FileGroup

Groups

IN!Name

ReportDirectory



# TReportManager Component



## Properties

### **Unit**

RptMan

### **Version Info**

This is version 1.0 of the TReportManager.

### **Description**

Use this component in conjunction with the [TModifiedReport](#) component to provide a level of abstraction for running reports, from Delphi, with ReportSmith. The TModifiedReport is analogous to the TDatabase component. While a single Database component provides an indirect reference to a path where data tables can be found, a single TReportManager component provides an indirect reference to a path where reports can be found. Currently, using TReport components, you have to specify the ReportDir property for each of your TReport components. This isn't very flexible because there is never any guarantee that your users are going to have their system setup just like yours. Consider this example: you are designing a system for your users and your users have the option of running a report. So when you are designing the system, you set the ReportDir property of all your TReport components to c:\app\reports. When you go to install your system, your users decide that they want your program on the D-drive. Now you have to go back to your office, change the ReportDir property for ALL of the TReport components, re-compile it, and then go back over to your users office. This is even worse if you are writing software for general consumption. Very frustrating indeed! In essence, the TModifiedReport-TReportManager connection is just the TTable-TDatabase connection; just as your TTable components do not have to know anything about where the physical data tables are, your TModifiedReport components do not have to know anything about where your reports are physically located.

If you want to hard-code a directory for TReportManager to use, you always have that option. To do this, simply specify a directory for the [Directory](#) property. However, I recommend that you go with the extra layer of abstraction by providing a file that the TReportManager will use to find the path. (Note: this is sort of like the BDE for reports). To do this, you must specify a filename in the [ININame](#) property. For more information on the format of this file see [INI File Format For TReportManager](#).

The [ReportDirectory](#) property will hold the directory that your reports are in, at run-time. You can obtain a list of the file groups, at run-time, that are in the file specified by the ININame property with the [Groups](#) property.

If you are using the [TInstall](#) component to install your programs, then if you want to use this component, you will want to have at least one [TInstallINIFile](#) component linked to the TInstall component that has the IncludeInstallInfo property set to true. This will automatically create the information for the TReportManager.

# IncludeInstallInfo

## Applies To

TInstallINIFile

## Declaration

**property** IncludeInstallInfo: Boolean;

## Description

You should set at least one TInstallINIFile component's IncludeInstallInfo property to true if you want to use the TReportManager component to point to a file group.

# Directory Property

## Applies To

TReportManager

## Declaration

**property** Directory: String;

## Description

The Directory property allows you to specified a hard-code directory in which the TReportManager component will find reports. I recommend using the ININame property instead.

# FileGroup Property

## Applies To

TReportManager

## Declaration

**property** FileGroup: String;

## Description

The FileGroup property specifies which file group, in the file specified by the ININame property, to use. E.g. each file group has an associated directory and the TReportManager component will return the filegroup's associated directory as the directory where reports will be found. For information on the format of this file see INI File Format For TReportManager.

# ININame Property

## Applies To

TReportManager

## Declaration

**property** ININame: String;

## Description

The ININame property tells the TReportManager component where to find a listing of the file groups and file group directories. For information on the format of this file see [INI File Format For TReportManager](#). If you specify a full path for ININame, then the TReportManager will look for the file in that path. If you do not specify a full path, but just a name, for ININame, then TReportManager will look in the current directory for the file. I recommend using the last method. Of course, if you do this, then you have to make sure that the file is in the same directory as your executable. **If you are using the [TInstall](#) component to install your programs and you specify a filename for the ININame property, then this file gets written automatically!**





# GoodInstallation Event

## Applies To

TInstall

## Declaration

**property** GoodInstallation: TNotifyEvent;

## Description

This event is called when there is no exception raised internally and the EAbortInstallation exception is not raised programatically. This is a good place to reassure the user that the software is ready to go.

## TInstallAliasNotifyEvent Type

### **Declaration**

TInstallAliasNotifyEvent = procedure(Sender: TObject; InstallAlias: TInstallAlias) of Object;

## TInstallININotifyEvent Type

### **Declaration**

TInstallININotifyEvent = procedure(Sender: TObject; InstallINIFile: TInstallINIFile) of Object;

# Groups Property

## Applies To

TReportManager

## Declaration

**property** Groups: TStrings;

## Description

Run-time only. The Groups property simply returns a list of all the group names in file specified by the ININame property.

# BadInstallation Event

## Applies To

TInstall

## Declaration

**property** BadInstallation: TNotifyEvent;

## Description

This event is called when there an exception is raised internally or the EAbortInstallation exception is raised programatically. This is a good place to tell the user what to do to rectify the situation.

# ReportDirectory Property

## Applies To

TReportManager

## Declaration

**property** ReportDirectory: String;

## Description

Run-time only. The ReportDirectory property is the directory in which your reports are found. If you have specified a value for the Directory property, then that directory will be found. If you have specified values for both the FileGroup property and the ININame property, then ReportDirectory will return the directory for the file group that is specified in the file (specified by the ININame property). For more information on the format of this file see INI File Format For TReportManager.

## INI File Format For TReportManager

The file you use must contain, but is not limited to (see sample #3), the following information; It can have anything other information that your program needs. **If you are using the TInstall component to install your programs and you specify a filename for the ININame property, then this file gets written automatically.** Note that sample #1 has one file group, specifically Reports1 and sample #2 has three file groups, specifically Finance, Commerce, and Industry. If you specify one of the file groups for the FileGroup property, then when the run method of any of your TModifiedReport components is called, its associated TReportManager component looks up the directory in this file and passes it to the TModifiedReport.

For example, Assume you have a file, named 'myapp.ini' that has the same format as in sample #1, in the same directory as your executable. If you specify 'myapp.ini' for the ININame property and Reports1 as the value for the FileGroup property, then the ReportDirectory property will be 'c:\myapp\reports1'. Also, if you have any TModifiedReport components hooked to this TReportManager, when their run method is called, the ReportDir property will be changed to 'c:\myapp\reports1' and ReportSmith will run the report, specified by ReportName, in this directory.

Sample formats:

{ The following is sample #1: the file does NOT contain this line }

```
[FileGroups]
GroupNames=Reports1
```

```
[Reports1]
Directory=c:\myapp\reports1
```

{ The following is sample #2: the file does NOT contain this line }

```
[FileGroups]
GroupNames=Finance,Commerce,Industry,
```

```
[Finance]
Directory=c:\myapp\finance
```

```
[Commerce]
Directory=c:\myapp\commerce
```

```
[Industry]
Directory=c:\myapp\industry
```

{ The following is sample #3: the file does NOT contain this line }

```
[MyProgramInfo]
Directory=c:\myapp
UserName=Bill
Version=1.0
Copyright=1995
```

```
[Miscellaneous]
Description=This is my application
```



[FileGroups]  
GroupNames=Reports1

[Reports1]  
Directory=c:\myapp\reports1

# TInstallRSConnectionNotifyEvent Type

## **Declaration**

TInstallRSConnectionNotifyEvent = procedure(Sender: TObject; InstallRSConnection: TInstallRSConneciton) of Object;

## TProgramItemNotifyEvent Type

### Declaration

TProgramItemNotifyEvent = procedure(Sender: TObject;ProgramItem: TProgramItem) of Object;

## TInstallFileNotifyEvent Type

### **Declaration**

TInstallFileNotifyEvent = procedure(Sender: TObject; InstallFile: TInstallFile) of Object;



## GoodInstallationDlg Component



Properties



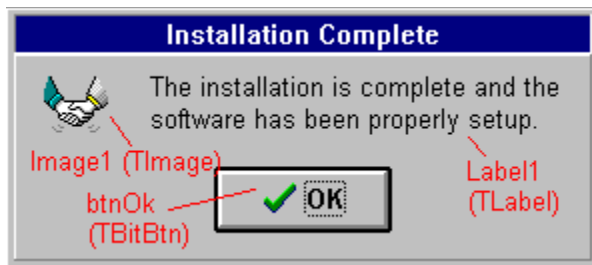
Methods

### Unit

IdGI

### Description

This is a stand-alone component, but not very useful outside of using it with the TInstall component. To use this component, call the Execute method in the GoodInstallation event of the TInstall component.



You may change how the dialog looks at run-time by using the DialogForm property. This property references the form that will be seen by the user. You can reference the various components contained by their names, shown in the previous schematic. To change the text string "The installation is complete and the software ...", you would execute the following code at run-time:

```
DialogForm.label1.Caption := 'Good going man, it's like all set up and everything.';
```

You can use the Title property to change the caption of the dialog.



## Properties

DialogForm

Title

# Execute Method

## Applies To

TGoodInstallationDlg

## Declaration

**procedure** Execute;

## Description

This method shows the dialog.



## Methods

Execute





## Registering this software

### Price Of Software

\$50.00 +  
\$5.00 Shipping/Handling Inside U.S.  
\$10.00 Shipping/Handling Outside U.S.

There are several ways for you to register this software:

Note: To receive your registered version faster, you can download a password protected ZIP file from either the World Wide Web site or the FTP site. Since I get notification of your registration on the day you register, you can send me e-mail telling me that you would like to receive the password for the software. As soon as I receive notification of your registration, I will e-mail you back and tell you the password. The name of the file on the FTP site will be ysfl + the version number + pw.zip. So for version 1.1, the file will be ysfl11pw.zip. For version 1.2, ysfl12pw.zip and so on and so forth.



### CompuServe

You may register this software via CompuServe. To do this, go to the SWREG (Software Registration) Forum. Select the Register Software option. The Id # for this software is 6010.

**Credit Card Orders:** Note: The author is not available at the following numbers.

You can order by credit card from PsL by calling 800-2424-PsL or 713-524-6394 from 7 a.m.-6 p.m. Mon-Thurs., 7 a.m.-Noon Friday or FAX 713-524-6398 or email to 71355.470@compuserve.com or mail credit card orders to PsL at P.O. Box 35705, Houston, TX 77235.

Ask for this software by it's number: 14160.

**THE ABOVE NUMBERS ARE FOR ORDERS ONLY. FOR INFORMATION OR SUPPORT, CONTACT BILL WHITE.**

Any questions about the status of the shipment of the order, refunds, registration options, product details, technical support, volume discounts, dealer pricing, site licenses, etc, must be directed to Bill White. To insure that you get the latest version, PsL will notify me the day of your order and I will ship the product directly to you.

## Author Information

Bill White  
3117 Raymond Drive  
Atlanta, GA 30340

CompuServe: 73612, 3477  
(770)457-5348

# CheckBoxes Property

## Applies To

TInstallFileGroupsDlg

## Declaration

**property** CheckBoxes[: Integer]: TCheckBox;

## Description

Use this property to gain access to the checkboxes at run-time in the InitializeDialog event.

# InitializeDialog Event

## Applies To

TInstallFileGroupsDlg, TComponentConflictDlg

## Declaration

**property** InitializeDialog: TNotifyEvent;

## Description

This event occurs just before the dialog is displayed. You may get access to the dialog through the DialogForm property.



## TUnInstall Component



[Properties](#)



[Methods](#)

[Events](#)

### Unit

UI

### Description

The TUnInstall component allows you to provide an un-installation program for your users. Just like the TInstall component, the TUnInstall component cannot work without a form. However, unlike the TInstall component, if you set the [RunAutomatic](#) property to True, the TUnInstall component will start doing its thing when the user runs the program and it will also close the application automatically. The most basic uninstallation program consists of four steps.

Step #1:

Drop a TUnInstall component onto your new project's main form.

Step #2:

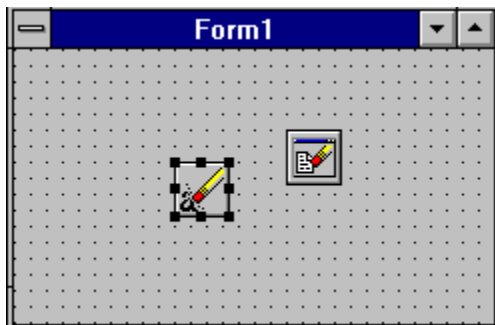
Drop a TUnInstallingFileDialog component onto your new project's form.

Step #3:

Set the [UnInstallingFileDialog](#) property of the TUnInstall component to point to the TUnInstallingFileDialog component that you added in step #2.

Step #4:

Set the INIFileName property to the name of an INI file that was set-up during the installation process, that includes the installation information. I.e. the [IncludeInstallInfo](#) property was set True. Therefore, the working directory for your UnInstall application must be the same directory in which this INI file is located.



| Object Inspector       |                     |
|------------------------|---------------------|
| Uninstall1: TUninstall |                     |
| INIFileName            | youseful.ini        |
| Name                   | Uninstall1          |
| RunAutomatic           | True                |
| Tag                    | 0                   |
| UninstallingFileDlg    | UninstallingFileDlg |
| Properties Events      |                     |

# INIFileName Property

## Applies To

TUnInstall

## Declaration

**property** INIFileName: String;

## Description

This property specifies the name of the INI File that holds information about the installation. I.e. an INI File that was created during the installation with a TInstallINIFile component whose IncludeInstallInfo property was set to True.



## Properties

INIFileName

RunAutomatic

UnInstallingFileDlg



# UnInstall Method

## Applies To

TUnInstall

## Declaration

**procedure** UnInstall; virtual;

## Description

This method uninstalls the files and program items created with the install program.

# RunAutomatic Property

## Applies To

TUnInstall

## Declaration

**property** RunAutomatic: Boolean;

## Description

If set to true, when your program is run, the UnInstall component will call the UnInstall method and when the uninstallation is finished, the TUnInstall component will terminate the application. If false, you must call the UnInstall method manually.

# UninstallingFileDlg Property

## Applies To

TUnInstall

## Declaration

**property** UninstallingFileDlg: TUninstallingFileDlg;

## Description

This property specifies the TUninstallFileDlg component that will be used in conjunction with this component. Set this property if you want show the users which files are being removed.



## Methods

[UnInstall](#)

class\_TUninstallingFileDlg

# OnAfterDeleteFile Event

## Applies To

TUnInstall

## Declaration

**property** OnAfterDeleteFile: TUnInstallFileEvent;

## Description

This event is called after a file has been removed.

## Events

OnAfterDeleteFile

OnBeforeDeleteFile

OnVerifyUnInstall

# OnBeforeDeleteFile Event

## Applies To

TUnInstall

## Declaration

**property** OnBeforeDeleteFile: TUnInstallFileEvent;

## Description

This event is called before a file has been removed. You can call the Abort procedure to abort the removal of the file.



# OnVerifyUnInstall Event

## Applies To

TUnInstall

## Declaration

**property** OnVerifyUnInstall: TNotifyEvent;

## Description

This event is called before any files or program items are removed. If you want to prompt the user about whether or not he/she wants to continue, this is the place. You can raise the EAbortUnInstall exception to abort the process.



## TColorFade Component

### **Unit**

YsfITool

### **Description**

This is a stand-alone component that lets you put a color fade background into your installations. You may set the direction of the fade with the Direction property. The color will fade from StartingColor to EndingColor. The ColorCount property specifies how many shades will appear between the StartingColor and EndingColor.

To use this with your installation, just plop this component onto your main form.



## Version Info

### Version 1.2

- Fixed the TInstallINIFile so that it does not overwrite the file that you specify; not it inserts the information into the file if it already exists (i.e. now it works like you think it would and should work).
- Added ParentFileGroup property to TInstallFileGroup.
- Added AllowChangeDir property to TInstallFileGroup.
- Added file group meta-variable capability to DestDir property for TInstallFileGroup.
- Fixed bug when groups were referenced with %f variable but the groups were not installed.
- Added CheckForVCLConflict property to TInstallFile.
- Enhanced the TInstallFileGroup component editor. You can now set the values for properties of multiple files at the same time.



• Checks for missing files before calling the Installation Builder--whereas before it just crashed.



• Installation Builder now allows you to partition your installation files onto the hard drive. I.e. you can build your multiple disk installations on the hard disk--whereas before you had to build to the floppy drive if you had multiple disk installations.



• Added CompressType to TInstall component. You can now use Microsoft's Compress utility, although you cannot use Installation Builder to build your installation disks if you use compress; you must build it manually..



• Added a new component TComponentConflictDlg. Allows developers to check the user's computer for possible conflicts with existing files in the user's Delphi VCL search path and files that will be installed.

### Version 1.1



• In version 1.0, when the Include property was set to False for a file group, it got installed anyway. This has been fixed in 1.1.



• Fixed the problem with Program Manager hiding the setup program at the end of the installation.



• Added a gradient fill component to the set of components.



• Added the TUnInstall and TUnInstallingFilesDlg component.

## Organization Of Setup Files

There are four parts to your installation. Note: it is only necessary to look at this if you have the CompressType property set to ctWinCompress or if you want to make the installation file(s)/disk(s) yourself.

### **setup.exe --**

A file named 'ysflpsu.exe' should have been copied to your Window's directory when you installed the Youseful Delphi Components. 'Setup.exe', on your installation, is this file. If you are manually building the installation, you must copy 'ysflpsu.exe' to your installation and I suggest that you rename it to 'setup.exe'. This is the file that the users will run to start your installation. This file actually copies your installation application executable to the user's Window's directory and then runs it.

### **files.lst --**

This file should be included with your installation and should have the following format:

```
[Begin Text File] { Do not include this line }
Name of your installation application executable
Disk #-Uncompressed Name Of File-Compressed Name Of File {ctWinCompress -- Optional;ctZIP -- N/A}
.
.
[EndTextFile] { Do not include this line }
```

You must list the files in the following order:

- All files for the first file group (In the order of first file first)
- All files for the second file group (In the order of first file first)
- .
- .
- All files for the last file group (In the order of first file first)

### **Unzipping DLL --**

A file name 'uzdll20.dll' should have been included in your Window's directory when you installed the Youseful Delphi Components. You MUST include this file in your installation.

### **Your Files --**

If you have the CompressType property set to ctZIP then you should ZIP the files you want on disk #1 to files1.zip, on disk #2 to files2.zip, on disk #3 to files3.zip, and so on and so forth. Important: You must zip your installation application executable file into files1.zip as 'XYZZY.EXE'. I.e. there should be a file in files1.zip called 'XYZZY.EXE', and it should be the compiled executable of your project with all the components.

If you have the CompressType property set to ctWinCompress then all you need do is to compress the files onto their respective disks with Microsoft's Compress utility.

# CompressType Property

## Applies To

TInstall

## Declaration

**property** CompressType: TCompressType;

## Description

The default value for this property is ctZIP. If you set it to ctZIP, then you can use the installation builder to build the installation disk(s)/file(s) for you. Even if you set this property to ctZIP, you can create the installation file(s)/disk(s) manually, see Organization Of Setup Files.

If you set this property to ctWinCompress, then YOU MUST build the installation file(s)/disk(s) manually--see Organization Of Setup Files. The files should be compressed onto your installation with Microsoft's Compress utility. Even though you must compress the files manually, the TInstall will uncompress them and copy them to their respective directories when the user runs your installation.



## TComponentConflictDlg Component



Properties



Methods

Events

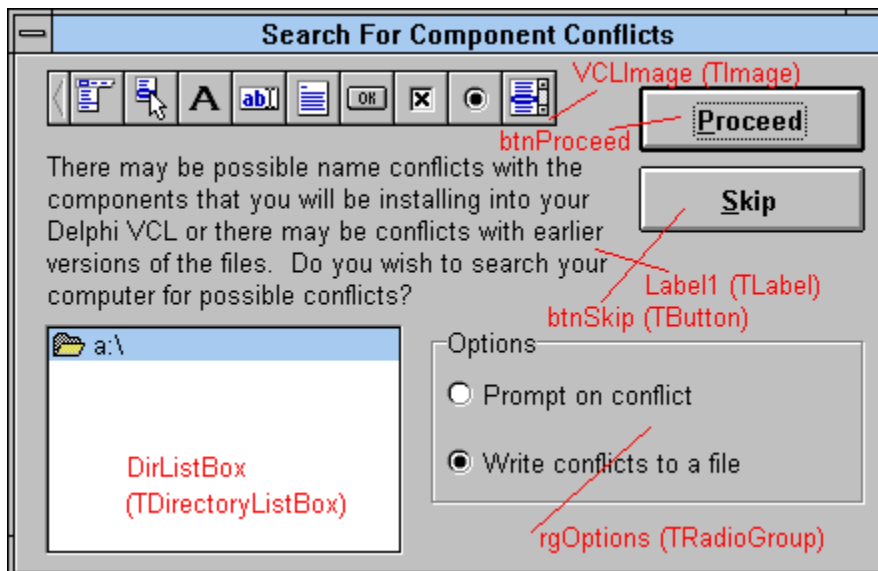
**Unit**

IdCS

### Description

You connect this component to a TInstall component through the InstallComponent property. Basically, this component allows **component writers** to build installations that will automatically search the user's computer for possible conflicts with existing components. A conflict occurs, in Delphi, when the user tries to install components or rebuild his/her library and there are two files with the same name in the library search path (for .PAS files, conflict will occur if there is another file with the same base name but with a .DCU extension; Similarly .PAS files will cause a conflict with .DCU files). For example, suppose the user already has a component name TWidget installed in his VCL. Your software is a Tfoobar component. The unit that TWidget is in is called 'cmpnt.pas'. Similarly, your Tfoobar component is contained in a unit called 'cmpnt.pas'. TWidget is located in the 'c:\widget' directory and the user installed your component into the 'c:\foobar' directory. When the user goes to install your component, Delphi will give an error message.

Hence, by using this component you can warn the user of possible conflicts that may exist with already installed components on his/her computer. If you want the user to see the default dialog, set the ShowDialog property to True. If you want the user to be prompted each time there is a conflict, set the ErrorReport property to erPrompt. If, however, you want the conflicts to be logged to a file, set the ErrorReport property to erLogToFile. Finally, you should call the Execute method AFTER the FILES in the installation have been COPIED--usually in the OnDoneCopyingFiles event of the TInstall component.



You may change how the dialog looks at run-time by using the DialogForm property. This property references the form that will be seen by the user. You can reference the various components contained by their names, shown in the previous schematic. To change the caption of the Proceed button, you would execute the following code at run-time:

```
DialogForm.btnProceed.Caption := 'Start';
```

You can use the Title property to change the caption of the dialog.



## Properties

DialogForm

ErrorReport

InstallComponent

ShowDialog

Title



# ErrorReport Property

## Applies To

TComponentConflictDlg

## Declaration

**property** ErrorReport: TErrorReport;

## Description

If set to erPrompt, then the user will be prompted whether or not to delete the file that caused the conflict.  
If set to erLogToFile then the user will be prompted to enter the name and path of a file in which to log all conflicts.

# ShowDialog Property

## Applies To

TComponentConflictDlg

## Declaration

**property** ShowDialog: Boolean;

## Description

If set to True then the user will see the default dialog. If set to false, then the user will NOT see the default dialog. This allows you to create your own custom dialogs if you like.



## Methods

Execute

# Execute Method

## Applies To

TComponentConflictDlg

## Declaration

**function** Execute: Boolean; virtual;

## Description

This method displays the dialog to the user, if the ShowDialog property is set to True. When the user hits the proceed button, the component then starts searching the user's computer for possible conflicts between units that are already on the user's computer and on the VCL path and units that have been installed. If the ShowDialog property is set to False, then the component starts the search without showing the dialog.

## Events

[OnInitializeDialog](#)

[OnSearchForConflictEvent](#)

[OnUnitConflict](#)

# OnSearchForConflict Event

## Applies To

TComponentConflictDlg

## Declaration

**property** OnSearchForConflict: TSearchForConflictEvent;

## Description

This event occurs for each file for which you have set the CheckForVCLConflict property to True. I.e. this event occurs when a file is being checked for possible conflicts on the user's computer. This event is provided so that you may present your own dialogs or interfaces to the user instead of using the default interface.

# TSearchForConflictEvent Type

## Unit

IdCs

## Applies To

TComponentConflictDlg

## Declaration

### type

```
TSearchForConflictEvent = procedure(Sender: TObject;  
    UnitName: String;  
    VCLPaths: TString;  
    var DefaultHandle: Boolean);
```

## Description

This event type returns the Sender, the unit that is being checked for conflicts, the paths on the user's VCL search path, and a variable that determines whether or not to perform the default handling. If you want to provide your own messages to the user, set the DefaultHandle to False.

# OnUnitConflict Event

## Applies To

TComponentConflictDlg

## Declaration

**property** OnUnitConflict: TUnitConflictEvent;

## Description

This event occurs whenever there is a conflict between the unit (as specified in the FileName parameter) and a file with the same name on the user's VCL search path.



# TUnitConflictEvent Type

## Unit

IdCs

## Applies To

TComponentConflictDlg

## Declaration

### type

```
TUnitConflictEvent = procedure(Sender: TObject;  
    FileName: String;  
    var DefaultHandle: Boolean);
```

## Description

This event type returns the Sender, the unit that is being checked for conflicts, and a variable that determines whether or not to perform the default handling. If you want to provide your own messages to the user, set the DefaultHandle to False.

# CheckForVCLConflict Property

## Applies To

TInstallFile

## Declaration

**property** CheckForVCLConflict: Boolean;

## Description

This property is used by the TComponentConflictDlg component. If set to True, then this file will be checked against the all files on the user's VCL search path for a possible conflict. A conflict occurs if another file has the same base name (i.e. w/o the file extension), and an extension of either .PAS or .DCU.

# AllowChangeDir Property

## Applies To

TInstallFileGroup

## Declaration

**property** AllowChangeDir: Boolean;

## Description

This property is mainly used by the TInstallFileGroupsDlg component. If set to true, then the user will not be allowed to change where the files will be copied. This should mostly be used if you have specified another TInstallFileGroup component in the ParentFileGroup property. By specifying a parent file group, you can put the %f meta-variable in the DestDir property to make sure that the files for this component always get copied to a subdirectory of the directory into which the parent file group gets copied. However, if you write your own interface, then you should mimic this behavior.

Example:

| <b>Property</b> | <b>Value</b>                          |
|-----------------|---------------------------------------|
| AllowChangeDir  | True                                  |
| DestDir         | %f\subdir1                            |
| ParentFileGroup | <another TInstallFileGroup component> |

If the user installs the parent file group to 'c:\install', then this file group will get copied to 'c:\install\subdir1'.

# ParentFileGroup Property

## Applies To

TInstallFileGroup

## Declaration

**property** ParentFileGroup: TComponent;

## Description

By specifying a parent file group, you can put the %f meta-variable in the DestDir property to make sure that the files for this component always get copied to a subdirectory of the directory into which the parent file group gets copied.

Example:

| <b>Property</b> | <b>Value</b>                          |
|-----------------|---------------------------------------|
| AllowChangeDir  | True                                  |
| DestDir         | %f\subdir1                            |
| ParentFileGroup | <another TInstallFileGroup component> |

If the user installs the parent file group to 'c:\install', then this file group will get copied to 'c:\install\subdir1'.

## It Just Will Not Work

Did you make sure that all the InstallComponent properties are set--especially for the file groups?

Did you make sure that you SAVED and COMPILED your project before building the installation.

Did you know that you must BUILD the installation before you can test it. You cannot test it under the Delphi IDE.

There is a bug in the software if you try to build the installation files in the same directory as your project. Try building it in another directory.

If you select a floppy drive as the destination for your installation files, then your installation will AUTOMATICALLY be split across disks if necessary.



