Properties Managed

This section shows all of the properties that **IniOut** is currently managing. In the first column, you will see a ✔ signifying that the entry is valid, or an
✘ indicating that the entry is not valid. New entries that are added will initially be marked as invalid, but once all the required fields have been defined, the entry will become valid. To the right of these symbols, you will see the component name and property of that component.
If the main **IniOut** dialog is enlarged, these columns will resize to show longer component or property names.

In the shareware version of **IniOut**, you cannot manage more than 20 properties.

Component

This drop-down list box allows you to select which component you want to save properties for. Any component or control that is sitting on the same form as the **IniOut** component will be present in this dialog.

Note that the form itself is also listed giving you access to its properties as well.

Property

This drop-down list box allows you to specify the particular property you wish to have managed. Any property that normally appears in the Object Inspector will be present here. If you are running the shareware version of **IniOut**, you will not see certain complex types (such as Fonts & class types), Method types (like OnClick, OnActivate), and Set types (a property that manages a set of True/False sub-properties).

Default

This drop-down list box allows you to specify what the default value will be for this property. This value is used if **IniOut** attempts to read from the INI file and the property is not defined there (or the INI file itself is missing). Note that the list will already have valid selections appropriate to the current property. For example, a Checkbox might want to save its Checked state and you might wish to have its default value be **True**. When the value is store in the INI file, it might look something like this:

```
[Checkbox States]
Checkbox1Checked=True
```

Default

This edit box allows you to specify what the default value will be for this property. This value is used if **IniOut** attempts to read from the INI file and the property is not defined there (or the INI file itself is missing).

In the case of certain complex types such as Fonts, there is actually more than one value that can appear here. Since Fonts actually have 6 sub-properties, you can specify theses 6 values all together separated by the vertical bar character ('|'). This character is typically found above the enter key on most keyboards. A typical font default might look something like this:

```
clWindowText|-11|Arial|fpDefault|8|[fsBold]
```

Note that there are 6 properties values here separated by vertical bars. These correspond to the 6 properties of a Font structure, namely **Color**, **Height**, **Name**, **Pitch**, **Size**, & **Style**.

Section

This drop-down list box allows you to specify the section that the property would be stored. For INI files, this is the section name that is surrounded by brackets. For Registry entries, this is the Subkey used to store the value. For example, if you entered **Checkbox States** here, the property would be stored under a section in the INI file that looks like this:

```
[Checkbox States]
Checkbox1Checked=False
```

If you wanted to save this same property in the Registry, the value for Section might be something like:

```
Software\ACME Games\Cosmo\CheckBox States
```

Note that this value is a Subkey under the RegistryRoot key defined with the Object Inspector. Please read the section on IniOut's Registry support in INIOUT.HLP.

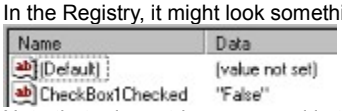
Note that prior entries are saved in this list to avoid the trouble of keying in the same section name over and over.

Identifier

This drop-down list box allows you to specify the identifier that will be used to specify this particular property. For INI files, this would be the identifier name on the left of the equal sign. For Registry entries, this is the value key used to identify this property. For example, if you entered **Checkbox1Checked** here, the INI file entry would look something like this:

```
[Checkbox States]
Checkbox1Checked=False
```

In the Registry, it might look something like this:



Note that prior entries are saved in this list to avoid the trouble of keying in the same identifier name over and over.

Add

Clicking this button will add a new property to the list of managed properties.

Delete

Clicking on this button will delete the currently selected property from the Properties Managed list.

IniOut© is a *Component Property Manager* for Delphi 2. With it you can easily provide INI file and Registry support to your Delphi application with virtually no code and a very small runtime footprint.

Currently, **IniOut** only supports Delphi 2.0 and 3.0. If you wish to be notified when a Delphi 1.0 version becomes available, or if you have general questions or comments about **IniOut**, please email me at **RobertV@compuserve.com**.

Note that context-sensitive help is available within the **IniOut** dialog by pressing F1 or by selecting the question mark icon at the top right corner and then clicking on an area of the form.

Sure... I've Seen This Before!

Yes, there are other products available that allow a component to save its value off to an INI file and restore the value later. Most of these perform this feat by taking a standard Delphi component and deriving a new component from it with the code necessary to link it to an INI file. The result? You now have a new component that is *INI-file aware*. ***But what are the disadvantages of such an approach?***

1. *Only the components provided are INI-File aware* - You may have 15 or 20 INI-aware components to use. But what if you want to save the value of a component that you don't have an INI-aware version of? Plus, this is 15 or 20 more components you need to manage on your Component Palette!
2. *Only one property will be saved by the component* - Typically, the author of the INI-aware components might well feel that there is really only one property of a component that needs to be written to and read from an INI file. For example, a TEdit would most likely want to write out its Text property. But what if you want to save the TEdit's location on the form, or its Enabled state?
3. *Won't work on any 3rd party components* - You won't be able to make any components INI File aware that you get from other companies. Plus, what if you made your own version of TEdit? Unless you want to rewrite your component to descend from a TIniEdit, then your own creation also won't be INI-file aware.

The **IniOut** Approach

The IniOut Approach

With **IniOut**, you don't need to deal with these issues anymore. Now there is a complete solution to making components INI-File aware.

1. *Works with any component* - Since **IniOut** is a manager of component properties and not just new INI-aware components derived from existing ones, it doesn't matter what component you want to make INI-File aware. **IniOut** will work with any component or control you might have. Even 3rd party controls or controls you have designed yourself!
2. *Works with virtually any property* - Want to save the color of that panel? The location of some TEdit fields? How about the Caption of a RadioButton or the Font of a Memo field? With **IniOut**, this isn't a problem, because you can read or write virtually *any property* of *any component* to an INI File.

How To Install IniOut

IniOut is installed into your component palette much like any other custom control. Simply copy all the included files into a single directory (or a directory where you store other custom controls) and then pick `Component|Install` from the Delphi IDE. Then click on Add and type in the name of the **IniOut** registration unit (INIREG.DCU). Click on OK and the component library will be rebuilt. **IniOut** is placed on the System tab of your component palette.

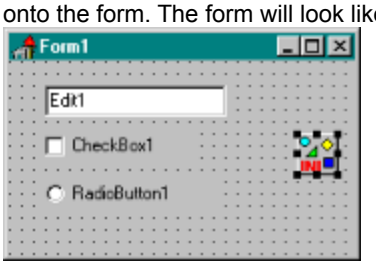
Make sure you install INIREG.DCU and **not** INIOUT.DCU. You will get an error when the component library is rebuilt if you do. The code is split into design-time and run-time features to minimize memory usage. The INIOUT.DCU file will be compiled in with your application for use at run-time, but the INIREG.DCU unit is only used at design time and includes the property and component editors for **IniOut**.

One further note… At times, IniOut will generate silent exceptions when trying to convert and assign property values. These are normal and are only seen when you are running your application through the Delphi IDE **and** "Break On Exceptions" has been turned on from the Tools|Options menu. These exceptions are all nested in Try..Except structures, but Delphi will still tell you about them if you have this option turned on.

How To Use **IniOut**

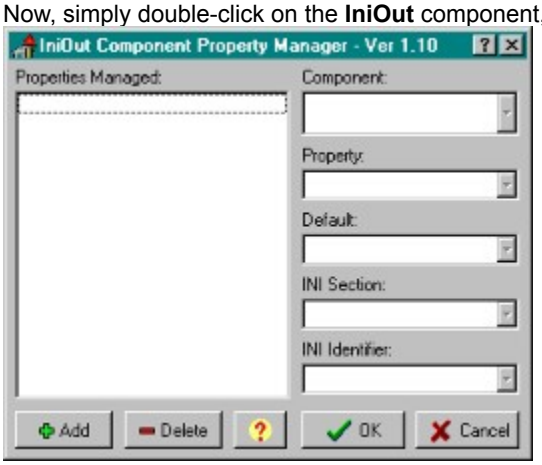
How To Use IniOut

To show you how easy it is to use **IniOut**, let's run through a quick demonstration. Start a new application and add a few controls to the form (how about an Edit field, a CheckBox, and a RadioButton). Lastly, drop an **IniOut** component onto the form. The form will look like this:

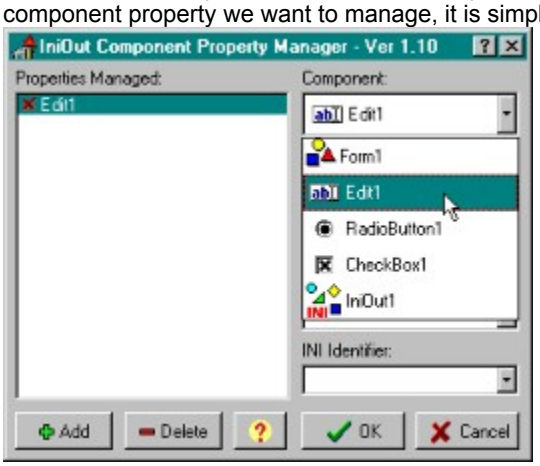


Next, set the **FileName** property in the Object Inspector to the name of the applications INI file. This will be the file used to store the values **IniOut** will be managing. If you do not specify a filename, you will get an warning message at runtime.

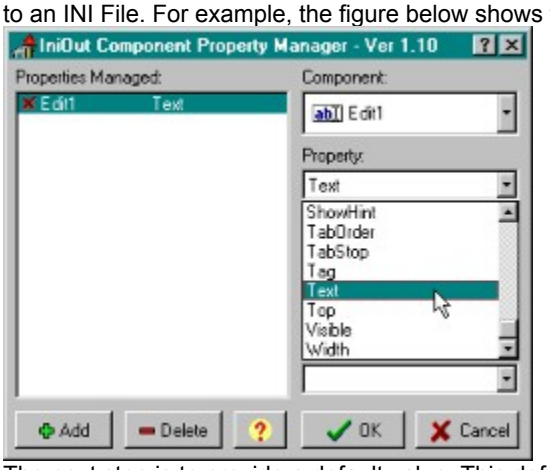
Now, simply double-click on the **IniOut** component, and you will see the following dialog box:



The list on the left shows you what properties are currently being managed. Click on the Add button to add a new component property. This inserts a new entry in the Properties Managed list box. Since we have not identified which component property we want to manage, it is simply listed as ***<Unknown>***. Now click on the Component combo box.

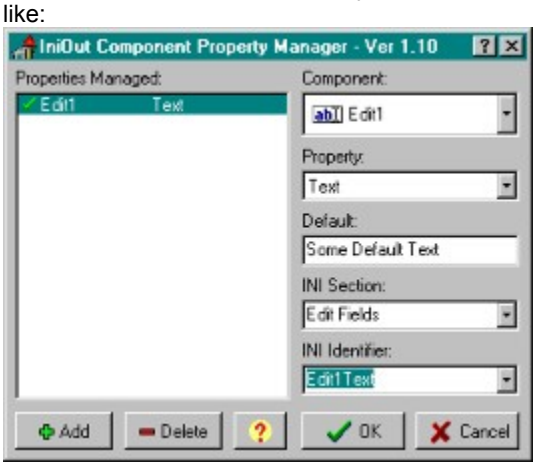


You will see a list of all the components currently on your form. After you chose one, you can access the other drop-down lists. The Properties combo box will show you all the properties of the component you chose that can be saved to an INI File. For example, the figure below shows the properties available for the TEdit component:



The next step is to provide a default value. This default is what **IniOut** will use in the event that the particular entry in the INI file is not present. For example, if we try to read Edit1's Text property from the INI file and either the INI file is not present, or the entry specific to Edit1 in the INI file is not specified, it will use this default when **IniOut**'s *LoadFromIni* method is called.

The last step is to provide a Section and Identifier where this property value will be stored in the INI File. For the *Section*, let's enter "Edit Fields" and for the *Identifier*, let's use "Edit1Text". After these last two items are defined, you will note that the red 'x' in the Properties Managed listbox has changed to a green check. This tells you that the entry is complete. If it is still an 'x' then you haven't provided all the information required. Here's what the dialog should look like:



We're almost there… Dismiss the **IniOut** dialog by clicking on **OK**. The next step is to tell **IniOut** when to load and save the component properties to and from the INI file. This is entirely up to your discretion. You might want to have a button or menu item on your application that saves defaults for example. In our example, we are going to make the system automated by having the INI file loaded in the FormCreate and saved in the FormDestroy. Go to the events tab for your form and double click on OnCreate. Enter the following code:

```
IniOut1.LoadFromIni;
```

then go to the OnDestroy, double-click and enter this code:

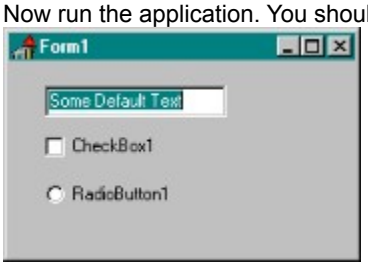
```
IniOut1.SaveToIni;
```

Your code should look like this:

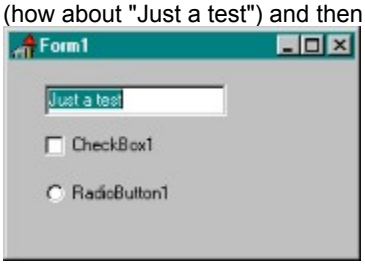
```
procedure TForm1.FormCreate(Sender: TObject);
begin
  IniOut1.LoadFromIni;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
```

```
  IniOut1.SaveToIni;
end;
```

Now run the application. You should see something like this:



Since the INI file did not exist, the default text we specified was provided in the Edit field. Change it to something else (how about "Just a test") and then quit the application. Then run it again. See what happened?

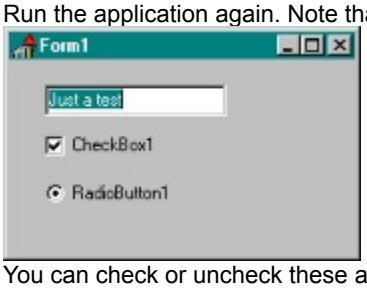


Since our OnDestroy event calls SaveToIni, it automatically saved the value of Edit1's text property. If you now look into the INI file you specified (probably created in your Windows directory), you will see something like this:

```
[Edit Fields]
Edit1Text=Just a test
```

With just 2 lines of code, we have created added a fairly sophisticated behavior to our simple Delphi application. To save the states of the CheckBox and Radio button, all you would need to do is add two more items to the **IniOut** manager like this:

Run the application again. Note that the CheckBox and RadioButton are now checked.



You can check or uncheck these as you wish and when you close the application their states are preserved in the INI file you specified.

```
[Edit Fields]
Edit1Text=Just a test
[Misc]
CheckBox1Checked=1
RadioButton1Checked=1
```

One further note… At times, IniOut will generate silent exceptions when trying to convert and assign property values.

These are normal and are only seen when you are running your application through the Delphi IDE **and** "Break On Exceptions" has been turned on from the Tools|Options menu. These exceptions are all nested in Try..Except structures, but Delphi will still tell you about them if you have this option turned on.

Where To Get It?

Shareware Notice

Registry Support

Where To Get It?

IniOut can be found in a number of locations including the Informant Communications Web site (http://www.informant.com/undu/iniout.zip) as well as at The Delphi Information Connection (http://www.delphi32.com/apps). On CompuServe, you can find **IniOut** in the Delphi 2 3rd Party Forum. The file name is INIOUT.ZIP. If you don't see it on these forums, you can contact me directly at **RobertV@compuserve.com** and I will email you a copy.

Shareware Notice

Shareware Notice

The downloadable version of **IniOut** is a slightly restricted demo version. For a mere US$20 you can get the full version from me by mail or email. The added features in the full version include:

1. Registry Support
2. Complex property types (Methods, Fonts, Sets, etc.)
3. No limit of properties managed
4. Free maintenance updates

If your version number at the top of the **IniOut** dialog ends with an 's', you are running the shareware version.

You can use CompuServe's SWREG system to register this component if you wish… the ID# for **IniOut** is **#14633**. Full source code is also available through me for US$99. Send your check or money order to:

Robert Vivrette
PO Box 1947
Swansboro, NC 28584-1947

Make checks payable to "Robert Vivrette". Be sure to include your email address (if you have one) so I can send you the appropriate version electronically. Sorry, but at this time I am unable to accept credit card orders. If you wish to contact me via email, I can be reached at **RobertV@compuserve.com**.

Where To Get It?

The Legal Stuff

The Legal Stuff

In A Nutshell: You can distribute the demo copy of IniOut as long as you don't change it and include all the documentation and files that came in the original package. You may not sell or share the full version (with or without source) to other people or I will send the police to your home or office. They won't be happy.

In case I said something in this paragraph above that conflicts with what is written blow, the legal notice below takes precedence.

So there.

License For Use

The use of this package indicates your understanding and acceptance of the following terms and conditions. This license shall supersede any verbal, or prior verbal or written, statement or agreement to the contrary. If you do not understand or accept these terms, or your local regulations prohibit "after sale" license agreements or limited disclaimers, you must cease and desist using this product immediately.

Copyright:

Liability disclaimer:

This product and/or license is provided as is, without any representation or warranty of any kind, either express or implied, including without limitation any representations or endorsements regarding the use of, the results of, or performance of the product, its appropriateness, accuracy, reliability, or correctness. The entire risk as to the use of this product is assumed by the user and/or licensee. **Robert Vivrette** does not assume liability for the use of this program beyond the original purchase price of the software. In no event will **Robert Vivrette** be liable for additional direct or indirect damages including any lost profits, lost savings, or other incidental or consequential damages arising from any defects, or the use or inability to use these programs, even if **Robert Vivrette** have been advised of the possibility of such damages.

Restrictions:

You may not use, copy, modify, translate, or transfer the programs, documentation, or any copy except as expressly defined in this agreement. You may not attempt to unlock or bypass any "copy-protection" or authentication algorithm utilized by the program. You may not remove or modify any copyright notice or the method by which it may be invoked.

Operating license:

You have the non-exclusive right to use any enclosed program only by a single person, on a single computer at a time. You may physically transfer the program from one computer to another, provided that the program is used only by a single person, on a single computer at a time. In group projects where multiple persons will use the program, you must purchase an individual license for each member of the group. Use over a "local area network" (within the same locale) is permitted provided that the program is used only by a single person, on a single computer at a time. Use over a "wide area network" (outside the same locale) is strictly prohibited under any and all circumstances.

Linking and distribution license:

In order to link and distribute compiled **IniOut** code in accordance to this license, you must be recognized by with **Robert Vivrette** as an authorized licensee. This occurs when you purchase the full product of **IniOut** (with or without source code). You may not reproduce or distribute copies of the full package, any of the source, interface, or code (.DCU) units, any of the documentation, nor may you supply any means by which your users could create, modify, or incorporate the **IniOut** component in their own products. Violations will be prosecuted to the maximum extent possible under the law.

Back-up and transfer:

You may make one copy of the program solely for "back-up" purposes, as prescribed by Canadian, United States, and international copyright laws. You must reproduce and include the copyright notice on the back-up copy. You may transfer the product to another party only if the other party agrees to the terms and conditions of this agreement, and completes and returns registration information (name, address, etc.) to **Robert Vivrette** within 30 days of the transfer. If you transfer the program you must at the same time transfer the documentation and back-up copy, or transfer the documentation and destroy the back-up copy. You may not retain any portion of the program, in any form, under any circumstance.

Terms:

This license is effective until terminated. You may terminate it by destroying the program, the documentation and copies thereof.

This license will also terminate if you fail to comply with any terms or conditions of this agreement. You agree upon such termination to destroy all copies of the program and of the documentation, or return them to **Robert Vivrette** for disposal.

Other rights and restrictions:

All other rights and restrictions not specifically granted in this license are reserved by **Robert Vivrette**.

Addendum for Demo Version:

IniOut has been distributed in a Demo form for the purpose of advertisement. This Demo copy may be freely distributed without prior consent from **Robert Vivrette** provided it remains intact and unchanged from its original state.

(New topic text goes here.)

IniOut Revision History

Version	Released	Comments
1.00	11/18/96	Initial Release
1.01	11/29/96	Fixed "Missing Resource" exception when component does not have DCR. Limited release.
1.02	01/07/97	Added access to Form properties. Added support for Class-type properties.
1.10	02/14/97	Significant code optimization. Added support for complex types, sets, floats, and methods. Integrated help system.
1.11	02/20/97	Finished registry support. Fixed error in reading method values. Updated help system to discuss Registry support.

Registry Support

The registered version of **IniOut** includes support for reading and writing values to the Windows Registry.

Before you read on, please understand one thing… It is important that you understand how the Registry works before you start using these features of IniOut. The Registry is a storehouse of all sorts of information essential to the normal day-to-day operations of Windows. As a result, if you inadvertently damage entries in the Registry, you could render Windows inoperative. IniOut's Registry support is reasonably safe and it is unlikely that you could do any lasting damage to the important areas of the Registry. However, please make sure you know what you are doing as I assume no responsibility for damaged you or IniOut may cause. If you are unsure, use the Registries export feature to save off its current state so you can recover it later if necessary.

So, with that out of the way, you need to complete the following steps to use the Registry in **IniOut**:

First, you need to identify what the **root key** will be. Normally, this is HKEY_CURRENT_USER and you select this value from the RegistryRoot property of **IniOut** in the Object Inspector. In order to provide an easy to use drop-down list, I have made the property hold values similar to the available root keys for the Registry. You can use any one you want, but make sure you know the impact of using something other than the default here.

Next, you need to identify what subkey will be used to store each property. In the **IniOut** dialog, the Section combo box is used to store this value.

If you wanted to save this same property in the Registry, the value for Section might be something like:

`Software\ACME Games\Cosmo\CheckBox States`

Note that this value is a Subkey under the RegistryRoot key defined with the Object Inspector (which is normally HKEY_CURRENT_USER).

Next, you specify the identifier used to store this particular property value in the Identifier combo box. For example, if you were storing the checked state of CheckBox1, you might want it's identifier to be CheckBox1Checked.

Finally, when you want to read values from the Registry, you call the LoadFromRegistry method of IniOut and when you want to save off the values to the Registry, you call the SaveToRegistry method (when working with INI files, you would use the LoadFromINI and SaveToINI methods). This typically should be done in the FormCreate and FormDestroy methods, but that isn't a requirement. The code to do this might look something like this:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  IniOut1.LoadFromRegistry;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  IniOut1.SaveToRegistry;
end;
```

Note that when you **save** to the Registry, subkeys that did not already exist will be created automatically. However, when **loading**, subkeys that do not exist will **not** be created.