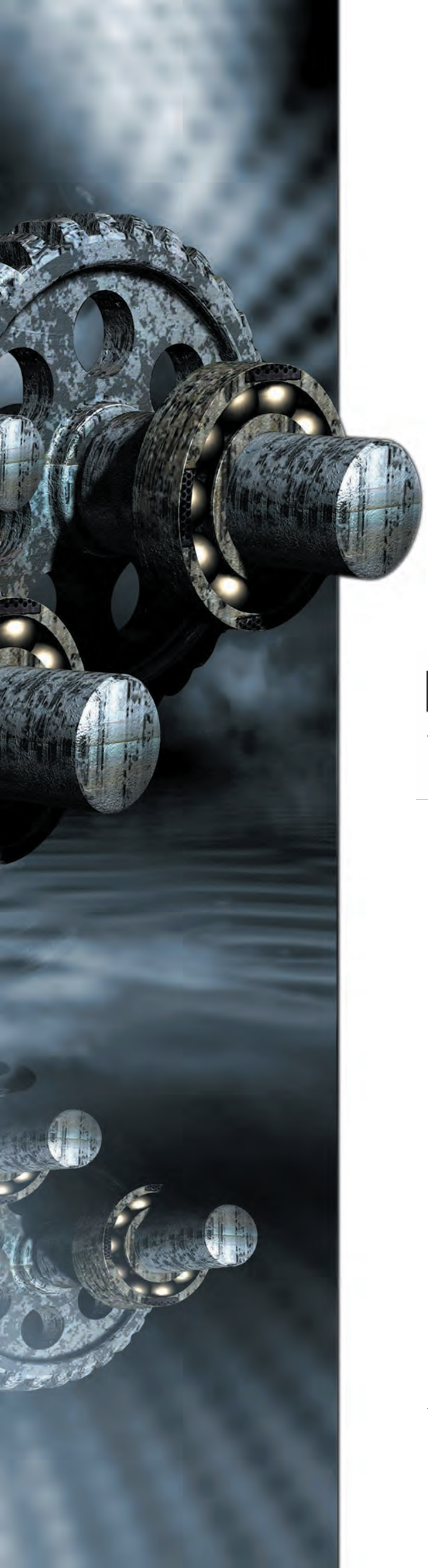


FOUNDATION 5

Developer Reference Manual

Fifth Edition

Foundation and the Foundation Developer Reference Manual
written by Walt Nelson with credit to Dave Batton

Software License & Limited Warranty

PLEASE READ THIS LICENSE CAREFULLY BEFORE USING THE SOFTWARE CONTAINED ON THE ACCOMPANYING DISKS. BY USING THE SOFTWARE, YOU AGREE TO BECOME BOUND BY THE TERMS OF THIS AGREEMENT, WHICH INCLUDES THE SOFTWARE LICENSE AND WARRANTY DISCLAIMER (collectively referred to herein as the "Agreement"). THIS AGREEMENT CONSTITUTES THE COMPLETE AGREEMENT BETWEEN YOU AND NELSON CONSULTING, INC. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, DO NOT USE THE SOFTWARE AND PROMPTLY RETURN THE PACKAGE FOR A FULL REFUND.

Nelson Consulting, Inc. ("Author") grants you a nontransferable, non-exclusive license to use this copy of the computer programs ("Software") and accompanying materials according to the following terms:

Ownership of Software

The enclosed manual and Software were developed and are copyrighted by the Author and are licensed, not sold, to you by the Author for use under the following terms, and the Author reserves any rights not expressly granted to you. You own the disks on which any software is recorded, but the Author retains ownership of all copies of the Software itself. Neither the manual nor the Software may be copied in whole or in part except as explicitly stated below.

License

You may:

i) make backup copies of the Software for your use provided all copies bear the Author's copyright notice;

ii) use this Software to create an unlimited number of custom or COMPILED commercial databases or applications created by the original licensee.

No additional product license or royalty is required except as noted below.

Databases created with the Software may be distributed only if they present the Software in a substantially modified form.

Restrictions

You may not:

i) distribute the UNCOMPILED, open source code of a custom database to more than one client, customer, developer, or other person or organization without written permission from the Author;

ii) distribute copies of the Software, even if substantially modified, to any client, customer, developer or other person or organization, for use as a development shell;

iii) remove any proprietary notices, labels or marks on the program and accompanying materials;

iv) remove or replace the 4D or the Author copyright notices displayed in the About Box of any program created with the Software; or

v) rent, transfer, or grant any rights in the Software to any person without the prior written consent of the Author.

Termination

Unauthorized copying of the Software or the accompanying materials, or failure to comply with the above restrictions, will result in automatic termination of this license and will make available other legal remedies to the Author. Upon termination you will destroy or return to the Author the program, accompanying materials, and any copies.

Limitation of Liability

In no event will The Author, ITS DEALERS, DISTRIBUTORS, AGENTS, OR EMPLOYEES be liable for any damages, including loss of data, lost profits, or other special, incidental, consequential or indirect damages arising from the use of OR INABILITY TO USE the SOFTWARE or accompanying materials, however caused and on any theory of liability. This limitation will apply even if The Author has been advised of the possibility of such damage. You acknowledge that the license fee reflects this allocation OF risk. Some jurisdictions do not allow limitation or exclusion of liability for incidental or consequential damages, so the above limitation may not apply to you.

Warranty Disclaimer

The Software and accompanying materials are provided "as is" without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose, except as noted below. The Author does not warrant that the functions contained in the program will meet your requirements or that the operation will be uninterrupted or error free. The entire risk as to the use, quality, and performance of the program is with you. Should the program prove defective, you, and not the Author, assume the entire cost of any necessary repair.

Limited Warranty

The Author warrants the diskettes on which the Software is furnished to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of delivery to you as evidenced by a copy of your receipt. The Author's entire liability and your exclusive remedy as to the diskettes (which is subject to you returning the diskettes to the Author with a copy of your receipt) will be the replacement of the diskettes. Some jurisdictions do not allow limitations on how long an implied warranty lasts so the above limitation may not apply to you. This warranty gives you specific legal rights. You may also have other rights which vary from jurisdiction to jurisdiction.

General

Use, duplication, or disclosure by the U.S. Government is subject to restrictions stated in paragraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at 252.227-7013.

This Agreement is the entire agreement between us and supersedes any other communications with respect to the program and accompanying materials.

If any provision of this Agreement is held to be unenforceable, the remainder of this Agreement shall continue in full force and effect.

Table of Contents

Introduction

About Foundation

Upgrading to Foundation 5

Starting Fresh with Foundation

Messing with Components

Art Component

Buttons Component

Data Component

Date Component

Dictionary Component

Dialogs Component

Find Component

General Component

Input/Output Component

Lists Component

Localization Component

Log Component

Menus Component

Message Component

Navigation Component

Preferences Component

Password Component

Records Component

Registration Component

Shell Component

Sort Component

Sequence Numbers Component

Text Component

Toolbar Component

Introduction

I would like to give a little background on Foundation for those of you just joining us.

History of Foundation

Foundation is a “shell” for 4th Dimension. It is a 4D database — just like the ones you can create — except that instead of specific tables and fields, it contains generic methods and forms that can be used as a starting point for your databases. You create custom databases with Foundation by adding tables and fields to a copy of the Foundation structure file. When you switch to 4D’s Custom Menus environment, Foundation will manage the menus, windows, and just about everything else.

With the release of Foundation 5, Foundation is now a single component that can be installed into any existing 4D database to provide additional functionality. Now you will be able to develop better databases in less time because Foundation is:

Professionally Designed and Coded

Foundation's original author, Dave Batton, has been working with 4th Dimension since its release in the U.S. in 1987. He is the author of the numerous 4D plugin packages, and many of the example databases available from the 4D web site. Dave is also a regular 4D Summit Conference presenter, co-creator of the 4DToday.com web site, and considered an expert in interface design within the 4D community. The new owner of Foundation is Walt Nelson from beautiful Seattle, WA. He has been working with 4D just as long as Dave has. He hopes to maintain the high quality of Foundation that Dave has created. Any problems with Foundation 5 are purely Walt's doing. Contact him at walt@foundationshell.com with your feedback.

Thoroughly Tested

Foundation has been extensively tested. It is getting a workout every day from thousands of end-users under widely varying conditions, providing much more testing than any single developer could possibly conduct.

Modular and Extensible

Foundation is designed to work for you. You should not have to change the way you work just because you are using a shell. Foundation's well-written, modular code is designed to be pulled apart and manipulated into whatever you need a shell to be. If you do not like one of Foundation's features, just take it out. Want to add a feature to your database? You can add new capabilities to your database just by installing a new component.

Multi-User Ready

All of Foundation's features are multi-user ready. Semaphores are used wherever necessary. If any feature that manipulates records encounters a locked record, the user is notified of the conflict.

Compiler Ready

Foundation is compatible with 4D's built-in compiler. All of Foundation's local variables have been declared at the top of the method in which they are used. Process and interprocess variables are typed in compiler methods. And 4D's **CLEAR VARIABLE** command is not used in any of Foundation's code, so your database will behave the same compiled as it does interpreted.

4D v11 SQL Required

Foundation 5 was created specifically to take advantage of the capabilities in 4th Dimension v11 SQL. It requires 4D v11, and we recommend you use 4D v11.4, at a minimum.

4D Version 2004 Users

If you are using a prior version of 4D, we still offer Foundation 4 which works with 4D 2003 and 4D 2004.

Human Interface Guidelines (HIG) Compliant

Foundation's interface is designed to give your database a professional look and feel. The form templates in the Foundation component will help you keep a consistent look as your database grows.

Acknowledgements

I (Walt Nelson) would like to thank you all for purchasing Foundation and supporting 4D third party products. Without your ongoing support, these products could not exist. Also, I would like to thank all the beta testers without whom this product would not have shipped. Guy Algot, Gary Boudreaux, Mike Erickson, Graham Langley, Justin Leavens, Mark Schaake, Wayne Stewart, and the members of the Seattle 4D SIG provided invaluable help and support in bringing Foundation 5 to market. And, of course, without Dave Batton there would be no Foundation Shell. Thanks to you all.

What Foundation Is Not

No shell can be all things to all people. Not everything you have ever wanted in a shell is going to be in Foundation, since your needs are different from those of other developers. We recommend that you try to make Foundation into the perfect shell for your needs. Add features where you need them. Pull out the stuff you never use. And if you want to, modify the behavior of Foundation's methods or the look of Foundation's forms.

Keep in mind that Foundation is not designed to teach you how to write 4D databases. However, you may be able to learn quite a bit from examining the code and reading the comments in Foundation's methods.

System Requirements

Computer

Foundation can be used on any computer that will support 4th Dimension v11.4 SQL.

Operating System

Foundation supports all system software that is compatible with 4th Dimension v11.4 SQL.

4th Dimension

To use Foundation 5, you will need either 4th Dimension v11.4 SQL, or 4D Server v11.4 SQL.

Compiling Your Database

Because of the complexity of the Foundation code, we highly recommend that you compile any databases created with it. Foundation is compiler ready, and includes complete compiler definition methods.

Developer Requirements

4th Dimension

If you have never completed a database that works in 4th Dimension's Custom Menus environment, you are probably not ready to tackle Foundation. You should feel comfortable with creating tables, fields, and forms. If you cannot do this, or find your way around in the Custom Menus environment, put Foundation aside for a while. Work through the 4th Dimension Tutorials before trying to use Foundation.

To work with Foundation you will need to be able to create form methods, assign menu bars to forms, and modify an existing project method.

Pointers

Once you are up and running with Foundation, you will want to make sure you have at least a basic understanding of pointers. Many of Foundation's generic methods rely on pointers. The more you know about pointers, the more value you will get out of this product and 4D in general.

Multiple Processes

Foundation is designed to use the multiple process capabilities of 4D. You do not need to know everything about multiple processes, but the more you know the easier it will be to take advantage of Foundation.

Database Design

One thing we would not even try to cover in this documentation is how to design a database. Read the documentation that comes with 4th Dimension, or get one of the many good books that are available. We also highly recommend joining a 4D users group in your area.

Foundation Support

Support is available from the Foundation web site <http://www.FoundationShell.com/support.php/>.

Technical Notes

On occasion we will post technical notes on our Web site. These answer commonly asked questions about Foundation, and describe components and example databases. We will occasionally publish technical notes from other Foundation users, too.

Components

We also produce additional components to add functionality to any database created with Foundation. Components may be provided for something as simple as a printout, or as complicated as an electronic mail system. We encourage you to tell us what kind of add-ons you would like us to provide. We will do our best to create components based on the most popular requests.

Shell Updates

Expect to see the Foundation shell updated on a regular basis. Updates may be needed to remain compatible with future versions of 4th Dimension, to fix bugs, or make improvements. You will be able to download minor Foundation updates at no cost for one year.

Along with any changes to Foundation, we will supply you with detailed information on what has changed and why. This will allow you to update your existing databases that are based on Foundation.

Bug Reports

‘Bug’ is not a dirty word with us. From time to time somebody will discover a problem in Foundation. When this happens, we will make a special effort to let you know about the problem and how to fix it.

User Submissions

We get lots of great ideas from other Foundation owners. So we will post these examples and techniques on our Web site. If you have got something cool to share, please send it to us! Do check out the Foundation Forum <http://www.FoundationShell.com/forum/>, where Foundation users congregate to share ideas, issues, and help each other out.

About Foundation

Although it is designed to be easy to use, Foundation is a complex product. It consists of components, plugins, and more. Before we show you how to use it, this chapter will help you understand how Foundation was designed, and how it is intended to be used.

Foundation Terminology

First, let us clear up some terminology.

Foundation

"Foundation" is the name of the entire product you are using now. There is no specific file or application or utility named Foundation. Foundation primarily consists of the Foundation Components, the Foundation Demo database, and the Foundation Extras plugin.

Foundation Shell

The Foundation Shell is a 4D structure file designed to be used as a starting point for creating new databases. It consists of about 2 menu bars, 4 tables, 7 forms, and 250 project methods. You can just add additional tables and fields to this database, create input and output forms, and then the database is ready to be used in 4D's Custom Menus environment.

The above numbers of forms and methods are slightly misleading. These are just the ones you can see. The Foundation Shell has been created using the Foundation Components. So in addition to the 20 or so public methods (methods you can modify) and approximately 230 protected methods (methods you can call, but cannot modify) in the shell, there are a couple hundred more private methods and more than a dozen private forms you cannot see.

Foundation Sub-Components

Foundation 5 now ships as a single component. Previously, Foundation was made up of several components. Those components will now be referred to as sub-components. These are self-contained groups of 4D methods, forms, menu bars, and pictures that can be copied into any 4D v11 database. Whenever an update to Foundation is released, the update can be easily installed into a Foundation Shell based database to fix bugs or add new features.

All of the Foundation sub-components require the Foundation General sub-component (Fnd_Gen). This component is the "engine" used by all of the other sub-components. It includes the routines necessary to bind the sub-components together in a project. The General sub-component also includes many generic routines, so these routines do not need to be duplicated inside of each individual sub-component.

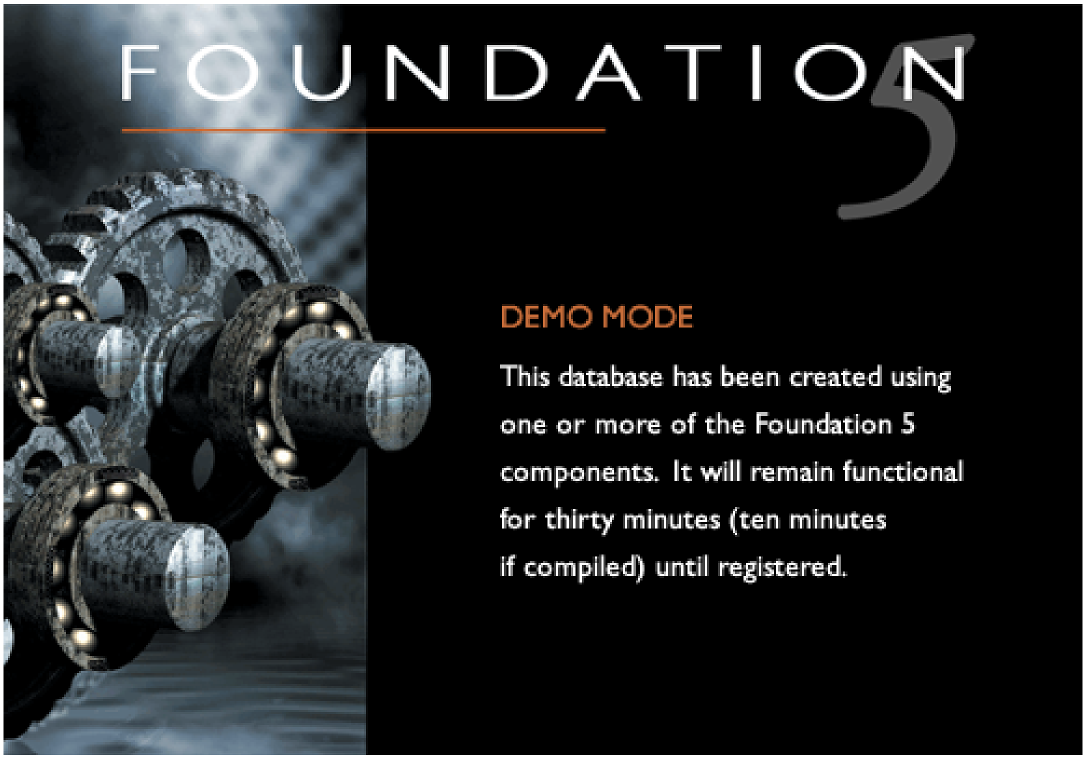
Foundation Extras Plugin

There are just a few things that cannot be done with 4D code that Foundation requires. So the Foundation Extras plugin contains these special routines. The plugin is available for both Macintosh and Windows, and the new bundle format introduced with 4D 2004.

The Foundation Extras plugin is required by the Foundation General component (Fnd_Gen), so it must be installed to use any of the Foundation components.

Demo Mode

If you are using the Foundation demo, you have a compiled version of the Foundation Shell component that is available as source code in the paid version of Foundation. You will receive an unlock code to turn off the annoying message you see when you first launch Foundation after purchase.


FOUNDATION 5

DEMO MODE

This database has been created using one or more of the Foundation 5 components. It will remain functional for thirty minutes (ten minutes if compiled) until registered.

This means that any database you create with the Foundation Shell demo can become fully functional when you purchase Foundation. Without the unlock code, any database created with the Foundation Shell or any database in which you install the Foundation Components will display a demo message when it is launched, and will time-out after 30 minutes (10 minutes, if compiled).

When you purchase Foundation, you will get an unlock code for the shell and the components.

Organization

It Is Just a Lot of 4D Code

We hate to admit it, but there is really nothing special about Foundation. It is simply a collection of 4D methods, forms, and pictures, just like you can create yourself with 4D. It is designed to provide you with the tools to help you deliver databases that run in 4D's Custom Menus environment, so you can have full control over the interface. It is designed to display tables and records, just like 4D's former User environment.

There are some significant differences between the Foundation interface and 4D's User environment. For example, Foundation displays each table in its own window. It displays each record in its own window. It also offers a simplified Find dialog, and platform specific interface elements.

The only thing that is missing is the tables and fields that the end user will work with. That is where you come in.

Component Architecture

Okay, so there is something about Foundation that is different from your structure files. We have created it by combining nearly 20 components. Of course, you could do this with any of your databases, but if your are like most 4D developers, you probably have not bothered to do this.

Components are bundles of 4D objects (methods, forms, style sheets, pictures, etc.) that are grouped together in one 4D structure file. You can install a component by copying it into the components folder next to your 4D structure file. The idea is that by installing a component you can easily add significant new functionality to your database. It is kind of like using a plugin, but it is written in 4D code rather than C or C++.

Hooks

The Foundation Shell includes more than a dozen public methods. You can, if you need to, modify the code in these methods to modify the shell's default behavior. These are referred to as "hooks," since they let you add code that "hooks into" the Foundation code.

If we make changes to a hook, we will document the change with the updated component so you can modify your hook directly in 4D.

No Data File Required

Unlike other shells, Foundation does not require a special data file. You, or your end users, can create a new data file, and it will work perfectly with Foundation.

Parlez-vous français?

Foundation 5 has been designed so that it can easily be localized for other languages. Although all of the menus, buttons and labels are displayed in English for the developer, this is just to make it easier to work with during development. At runtime, the text for all of these items are pulled from localization data stored as 4D lists in the structure file. Foundation can just as easily use French or German as it can English. And it is easy for you to add to this localization data to localize Foundation for any other language by modifying the components in Foundation.

Compiling

All Foundation methods are designed so they can be compiled with either the "All Variables are Typed" or "Type the variables" option selected in 4D's Compiler Preferences dialog.

We highly recommend compiling your Foundation Shell based databases before delivering them to the end user. Some Foundation routines behave differently depending on whether or not the database has been compiled. Basically, Foundation assumes that if the database is not compiled, it is being tested by the developer, rather than being used by an end user.

Using Foundation

Foundation Components

If you are starting a new project, you would not need to install the Foundation components first. However, you may want to install Foundation into your Components folder before you start to take advantage of the Foundation shell benefits.

The components can also be installed into any of your existing projects that were not developed using the Foundation Shell. You can also turn an existing project into a full Foundation Shell-style project by installing all of the required Foundation Components, and by making a few simple changes to your structure file. This process is described in the "Starting Fresh with Foundation" chapter.

Naming Conventions

All of the Foundation database object names are prefixed with “Fnd_”. This prevents naming conflicts with your code and with other non-Foundation components.

You should never create any database objects (forms, tables, methods, variables, etc.) in your database structure using a name that beings with “Fnd_” when working with the Foundation Shell or in a database that contains Foundation Components. This may cause runtime problems, and will prevent you from installing new and updated Foundation Components. You should prefix object names with “Fnd_” when modifying a Foundation component.

Foundation also uses a variable naming convention. You will never see any of Foundation’s variables in the Foundation Shell — these variable names are visible only when working with the component. All variables used in Foundation end with an underscore and a letter that indicates the variable type:

Variable	Type
VariableName_t	C_TEXT
VariableName_s	C_STRING (rarely used)
VariableName_i	C_LONGINT (C_INTEGER is not used)
VariableName_r	C_REAL
VariableName_b	C_BOOLEAN
VariableName_blob	C_BLOB
VariableName_ptr	C_POINTER
VariableName_pic	C_PICTURE
VariableName_at	ARRAY TEXT
VariableName_as	ARRAY STRING (rarely used)
VariableName_ai	ARRAY INTEGER or ARRAY LONGINT
VariableName_ar	ARRAY REAL
VariableName_ab	ARRAY BOOLEAN
VariableName_aptr	ARRAY POINTER
VariableName_apic	ARRAY PICTURE

Upgrading to Foundation 5

4D v11 SQL implements a dramatically different (and better) component system than previous releases of 4D. Rather than installing components with 4D Insider, component source code is now left in the source database, and this structure file, either interpreted or compiled, is dropped into a folder named "Components" at the same level as the host database's structure file.

Because of this significant change, 4D database structures with components installed directly into the structure file cannot be opened by 4D v11 SQL. The old components must first be removed, then replaced with updated components that are designed to work with the new component architecture.

The Foundation components have been upgraded to work with 4D v11 SQL, and we are calling them Foundation 5. Although they are designed to look and behave similar to the Foundation 4 components, internally many changes had to be made to work within the limitations imposed by the new component architecture.

This chapter describes the steps necessary to upgrade a Foundation 4-based structure file to work with 4D v11 SQL and Foundation 5. It does not cover creating new databases with Foundation 5.

About Foundation 5

Many things have changed with the new component architecture in 4D v11. For example, there is no longer such a thing as a public method. Also, components can no longer access the lists, forms, or menus of the host database.

Hook Methods

Foundation 5 still uses hooks just like Foundation 4 did, but with some important differences.

The names of the hook methods has changed. Wherever "_aa_" occurred in the name of a hook, it has been replaced with "_Hook_." So, for example, the *Fnd_aa_Shell_Find* hook is now named *Fnd_Hook_Shell_Find*.

This was done for two reasons. First, the usefulness of having the hooks sort to the top of the list of methods in the Explorer window is no longer important, since all of the shared component methods are now listed in another area of the list. Second, by changing the names of the hooks the upgrade process is simplified.

You will find 22 of these hook methods in the Foundation Example database called Product Sales. Copy these 22 hook methods to your Host database, if any of them are missing.

Host Methods

To work around some of these limitations, Foundation may ask you, depending on the components you are using, for permission to create some new shared methods in your structure file. The names of these methods will start with "Fnd_Host_."

These methods are created automatically by using the **AP Create method** command available in 4D Pack. 4D Pack may have been automatically added to your copy of 4D v11 during installation, so you may not need to worry about adding this plugin to your Plugins folder. However, you should check to make sure this plugin is availble to 4D before trying to work with the Foundation 5 components.

The necessary "Compiler_" methods are also automatically generated. If you delete or rename a host method, Foundation will ask again for permission to create the method. But you should always delete both the host method and its associated compiler method at the same time, since Foundation will attempt to create both of them.

Special Note regarding 4D Server:

These routines for creating hook and host methods will not set the **Shared by components and host database** attribute for methods, if you run them with 4D Server. Make sure you are running in Single User 4D to set the **Shared by component** attribute for these methods.

Project Forms

Unlike previous versions of 4D, 4D v11 SQL forms do not need to be associated with a table. These forms are referred to as project forms.

Components can no longer use table forms, so all Foundation forms are now project forms.

If you have created custom Startup Dialog or About Box forms, you will need to change these to project forms. This is easily done by dragging the form name in the Explorer window to the Project Forms label.

4D Version

This version of the Foundation components has been tested with 4D v11.4. You will probably have problems with earlier versions, and things might work even better with later versions.

Broken Parts

Unfortunately, not everything that was available in Foundation 4 is working in this release of Foundation 5. These features will return in future updates.

Localization

The Localization Editor is not yet functional. And no testing has been done to ensure that the Foundation components work properly when localized with non-English languages. Your localizations should work, but we are not making any promises.

Menus

4D v11 SQL Release 2 switches from Foundation's procedurally created menu bar back to Menu Bar #1 after the *On Startup* database method completes. So Foundation posts a keyboard event to trigger a menu item to switch back the correct menu at the end of the startup process. This works fairly well, but unfortunately you will still see the wrong menu bar if all windows are closed.

In 4D v11 SQL Release 2, the **Quit** and **Preferences** menu items are not functional if they are automatically moved to their proper positions on Mac OS X. So you will find the **Quit** menu item under the **File** menu and the **Preferences** menu item under the **Edit** menu on Mac OS X.

Missing

The Foundation Mail component has not yet been updated for 4D v11 SQL.

Intel Macs

Foundation Extras has been updated to a universal binary, so it will allow your database to run in native mode on Intel-based Macs.

Upgrading From a Version 5 Database to New Version 5 of Foundation

To upgrade from Foundation 5 to a newer version of Foundation 5, simply replace the Fnd_All.4DB component in your components folder next to your structure file. If you have modified any methods or layouts in the Fnd_All.4DB, you will have to make sure you marry your version with the new version. It is recommended that you not modify Fnd_All.4DB for this reason.

Upgrading Your Foundation Version 4 Structure

Backup the Database

Make a backup of your structure file and data file.

No, really. Right now. Before continuing.

Update Obsolete Method Calls

Search your structure for any calls to the obsolete Foundation Window calls that start with "Fnd_Wnd_Set." The word "Set" has been removed from these calls. For example, *Fnd_Wnd_SetTitle* is now *Fnd_Wnd_Title*. The older, obsolete method calls are not available in Foundation 5.

Make Copies of the Public Objects

In a moment, we will use 4D Insider to remove all of the Foundation 4 components from the structure file. But this will also remove any hooks you may have modified. It will also remove the preferences form. So first we need to create copies of these objects.

NOTE: It is possible to rename the existing hooks (although it should not be, since they belong to the component). Do not simply rename the hooks. They will still be deleted when the component is removed. You must create new methods and copy the contents of your hooks into the new methods.

Duplicate the Hooks

Launch the database with the older version of 4D. Take a look at each of the Foundation hook methods (they start with "Fnd_aa") and for any of them that you have modified, create a copy of it, replacing "_aa_" (or "_aaa_" in the case of the *Fnd_aaa_Shell_Setup* method) with "_Hook_." So, for example, the copy of the *Fnd_aaa_Shell_Setup* method will be named *Fnd_Hook_Shell_Setup*, and the copy of the *Fnd_aa_List_SetEditableLists* method will be named *Fnd_Hook_List_SetEditableLists*.

This technique causes two of the new method names to exceed 4D's 31 character limit so rename these files as shown here:

Fnd_aa_Shell_NavigationPalette becomes *Fnd_Hook_Shell_NavPalette*

Fnd_aa_Shell_InitializePlugins becomes *Fnd_Hook_Shell_InitPlugins*

Duplicate the Preferences Form

Create a new form named "Preferences." It does not matter which table you assign it to—we will change it to a project form (no related table) after upgrading to 4D v11 SQL. Copy all of the form objects from the `Fnd_Pref_Preferences` form to this new form. Also copy the form method to the new form. Set the form size to match the size of the `Fnd_Pref_Preferences` form. Select the **Fixed Width** checkbox in the Property List, and make sure the necessary form events are selected.

Duplicate the Registration Forms

If you are using the Foundation Registration component, and you have modified either the `Reg_DemoDialog` or `Reg_RegisterDialog` forms, create a new form and copy the contents to the new forms. As with the Preferences form, it does not matter which table you assign it to. Do not forget to copy the form methods, also.

Remove the Components

Quit 4D and open the structure with 4D Insider. Remove all of the components. Even non-Foundation components must be removed. 4D v11 SQL will not convert a database that contains old-style components. Quit 4D Insider.

Upgrade Foundation Extras

If your database has a Mac4DX or Win4DX folder, rename this folder "Plugins." Remove Foundation Extras from this folder and replace it with the Foundation Extras 5.0 plugin. An alias/shortcut to the plugin will also work.

Install the Components

Copy the Components folder to the folder that contains your database structure. Aliases/shortcuts to the components will also work.

Upgrade to 4D v11 SQL

Open the database with 4D v11 SQL. Click through the conversion dialog. An error will occur during startup when one of the missing Foundation components is called. Click **Abort**.

Unicode

Open the 4D Preferences window. Select the **Application/Compatibility** area. Select the **Unicode mode** checkbox. Click **OK** to save the change.

Share the Hook Methods

Go to the design environment and, for each of the Hook methods, select the **Shared by components and host database** checkbox in the Method Properties window. The easiest way to do this is to select the **Batch Setting of Attributes** command from the gear menu in the Explorer window. Enter "Fnd_Hook@" in the field at the top, then select **Shared by components and host database** from the menu. Click **True**, then click the **Apply** button. Make sure you are running in Single User 4D to set the **Shared by component** attribute for these methods.

Convert Table Forms to Project Forms

In the Explorer window, go to the Forms list, then find the Preferences form you created earlier and drag it up to the **Project Forms** label. This will move the form.

If you are using the Registration component, drag the Reg_DemoDialog and Reg_RegistrationDialog forms to the Project Forms label.

If you created custom Startup or About Box forms, find them and drag them to the **Project Forms** list too.

If you have a [Fnd_Forms] table, and it no longer contains any forms, you can delete this table.

Fnd_Dlg_Request

Fnd_Dlg_Request no longer returns the value entered by the user. You must now call *Fnd_Dlg_GetRequest* to get this:

```
Fnd_Dlg_Request ("Enter password:")
If (OK=1)
  $result_t:=Fnd_Dlg_GetRequest
End if
```

Fnd_Gen_Get4DString

Fnd_Gen_Get4DString has been removed. If you used this method, replace these with calls to *Fnd_Gen_GetString* and add "Fnd_Gen" as the first parameter. For example:

```
Fnd_Gen_Get4DString("Stop")
```

becomes:

```
Fnd_Gen_GetString("Fnd_Gen";"Stop")
```

If you were using a lookup code of "Don't Save" you will need to change it to "DontSave."

Fnd_Art

Any calls to *Fnd_Art_SetAboutForm* and *Fnd_Art_SetStartupDialogForm* must be modified to pass just the form name. The form must now be a project form, not a table form.

Style Sheets

If you assigned any of Foundation's style sheets to your form objects, those objects no longer have style sheets. You will need to recreate the style sheets and reassign them to the form objects.

Fnd_Objects

From the included Fnd_Objects database, drag the Fnd_IO_InputForm and Fnd_Tlbr_Toolbar forms to your project (you will need two copies of 4D v11 SQL running to do this). This will also move some pictures and style sheets to your project.

From the same database, drag the *Compiler_Fnd_IO* and *Compiler_Fnd_Tlbr* project methods to your database.

Open each of your input forms and set it to inherit the Fnd_IO_InputForm project form.

Open each of your output forms and set it to inherit the Fnd_Tlbr_Toolbar form.

Menu Bar

Select **Menus** in the Toolbox. You should still have a Fnd_Shell menu. In this menu, make sure there is an item (it is probably labeled "Start Foundation") that calls the *Fnd_Shell_OnStartup* method, and has Command-S (or Control-S on Windows) assigned as the shortcut. This is a temporary work-around required because of a bug in 4D v11 SQL. This has been reported to 4D and, hopefully, this step will not be necessary with the 11.4 release.

Compile

Try compiling. You may get errors for the variables on the Preferences form. If so, add these compiler directives to your project in a "Compiler_" method.

If the compiler finds any methods that start with "Fnd_Wnd_Set," update these commands by removing the word "Set" from the method name. These commands were made obsolete about a year ago, and are no longer supported. For example, *Fnd_Wnd_SetPosition* has been replaced by *Fnd_Wnd_Position*.

Hopefully, now your database will compile. Do not try to use it yet, just make sure it compiles, and fix any remaining compiler errors.

Delete the Resource File

Quit 4D. Locate the .rsr file. This is no longer needed and can be deleted.

Launch

Launch the database again. This time, Foundation will then ask you for permission to create methods named *Fnd_Host_ExecuteFormula* and *Fnd_Host_GetFormProperties*. Click the **Add Method** button in each of these dialogs. Associated compiler methods will be created, too.

Your database should now startup normally. If menu bar 2 does not appear (this might happen if you do not display the navigation palette at startup), select **Start Foundation** from the **File** menu.

Starting Fresh with Foundation

No matter how much experience you have with 4th Dimension, you will want to read this chapter to learn how to create a database with Foundation. Then you can decide whether you want to keep going on your own, or finish reading this manual.

Setup

Foundation 5 consists of a single structure file. It is named Fnd_All.4DB. The Fnd_All.4DB is the 4D Component database that you will place inside the Components folder next to your 4D structure file. To create a new database from scratch, start by launching 4D and create a new database. Quit 4D, and just copy Fnd_All.4DB to your Components folder next to your brand new structure file. To add Foundation to an existing structure, do the same thing, copy Fnd_All.4DB to your Components folder. See the Chapter called Messing with Components for more information on dealing with 4D Components.

In addition, you will need to copy the Foundation Extras plugin named Extras.bundle to your Plugins folder next to your 4D structure.

After the startup message has closed, you may select Quit (Macintosh) or Exit (Windows) from the File menu. Because Foundation defaults to development mode (not to be confused with 4D's Design Environment), it will not really quit 4D. Instead the menu bar will change and you may select Return to Design mode from the Mode menu.

Create the Tables, Fields and Forms

To create a database with Foundation, start off just like you would to create a database normally in 4D. Then you just need to make a few minor changes. Begin with a simple structure so that you can quickly get the hang of Foundation. Create a few tables with a few fields each. Create an input and output form for each table that your users will see in the Custom Environment. Any tables or fields marked as invisible will not be displayed in the Custom Menus environment by Foundation. Any searchable/sortable fields will automatically be displayed in Foundation's Find Dialog and Sort Dialog.

Create the Forms

First, copy the Fnd_IO_InputForm and Fnd_Tlbr_Toolbar forms from the Product Sales.4DB. You will probably want to copy these to your Project Forms area, rather than to a specific table.

Foundation requires an input and output form for each visible table. You should name the input forms "Input" and the output forms should be named "Output."

The Product Sales.4DB example database contains form templates to help you create input and output forms. Be sure to set the size using 4D's Form Properties dialog, so Foundation can figure out what window sizes to use.

Display 4D's Property List, and locate the "Inherited Form Name" setting. Select "Fnd_IO_InputForm" as the Inherited Form Name for your input forms, and select "Fnd_Tlbr_Toolbar" as the Inherited Form Name for your output forms.

Set Up the Forms

To get your forms working properly in the Custom Menus environment, you will need to do a few more things:

1. Create a form method for each input and output form (just type Command-option-K (Macintosh) or Ctrl-option-K (Windows). The input form method should call Foundation's *Fnd_IO_InputFormMethod* and the output form should call Foundation's *Fnd_IO_OutputFormMethod* method.

2. For output forms, set the Header, Detail, Break 0, and Footer lines using the guides provided on the form template. The Foundation Toolbar extends down to 76 pixels, so your objects should be below 77 pixels. Header line should be at 93 pixels, Detail, Break and Footer lines should be at 111 pixels, if you are using our templates.

3. Using the Property List palette, associate the form with the "Fnd_Shell" menu bar, and select the Active Menu Bar check box. Then select "None" from the Associate Menu Bar.

4. Also in the Property List palette, set the form's events to handle the events listed on each template.

Input Form Events checked:

On Load
On Validate
On Activate
On Close Box
On Outside Call
On Clicked
On Data Change
On Resize

Output Form Events checked:

On Load
On Unload
On Activate
On Deactivate
On Close Box
On Outside Call
On Clicked
On Double Clicked
On Resize
On Header
On Display Detail

5. Set the window size as desired. Foundation will use the form size as the default size for the window. Foundation is designed to work properly with resizable input and output windows.

You can also set the minimum window width and height as desired.

Try compiling. You may get errors for the variables on the Preferences form. If so, add these compiler directives to your project in a "Compiler_" method. If methods from Components do not tokenize or Syntax check says the methods do not exist, force retokenizing with Command-Shift-Enter.

Launch the database again. This time, Foundation will ask you for permission to create methods named *Fnd_Host_ExecuteFormula* and *Fnd_Host_GetFormProperties*. Click the Add Method button in each of these dialogs. Associated compiler methods will also be created. Your database should now startup normally. If menu bar 2 does not appear (this might happen if you do not display the navigation palette at startup), select Start Foundation from the File menu. Be sure you have a menu item added to your Menu Bar No 1, that contains a Start Foundation item that calls *Fnd_Shell_OnStartup* and has the shortcut of Command-S enabled.

(This is a BIG ONE) - If Methods inside Execute Method are not recognized, do the following. Make sure any Fnd_Host_* methods have the "Shared by components and host database" option set, in the Method properties. From Execute Method command reference: If you call this command from a component and pass a method name belonging to the host database in methodName (or vice versa), the method must have been shared ("Shared by components and host database" option, in the Method properties).

That is all you need to do to begin working in the Custom Menus environment. However, you will probably want to go into the *Fnd_Hook_Shell_Setup* method to customize your database. Your database should work properly now. You can switch to the Custom Menus environment by selecting Test Application from the Run Menu, and select Start Foundation from the File menu.

Customize It

Now that you have all of the basics working, you will want to go into the *Fnd_Hook_Shell_Setup* method to customize your database. This public method contains four calls to the *Fnd_GenSetDatabaseInfo* command.

Modify these four calls with your own custom information:

```
Fnd_Gen_SetDatabaseInfo ("DatabaseName";"A Great Database Built with Foundation")
Fnd_Gen_SetDatabaseInfo ("DatabaseVersion";"1.0 beta 1")
Fnd_Gen_SetDatabaseInfo ("DatabaseCopyright";"Copyright ©2009 Your Company Name")
Fnd_Gen_SetDatabaseInfo ("DatabaseURL";"http://www.YourCompanyURL.com/")
```

DatabaseName

Pass the name of your database to this method. The name you enter here will be shown in the Startup Message when the database is launched and will also be displayed in the Custom About Box. The About menu item will display this name.

This name will also be used when creating the local preferences file. Each database you create should be given a unique name so that it does not overwrite the preference file of another database.

DatabaseVersion

This variable is used to display your database's version number to the end user in the Startup Message and the Custom About Box . It can be in any format you want, since it is simply for display in the Startup Message and Custom About Box dialogs.

DatabaseCopyright

Place a copyright statement here. This information will be displayed in the Custom About Box and in the Startup Message . This string can contain up to two lines separated by a carriage return, although only the first will be displayed in the Startup Message.

DatabaseURL

Place a full Web page URL (including the "http://" part) here. This will be displayed in the Custom About Box, and users can click on it to get to display the page in their browser.

Try It

Once you have the Fnd_Hook_Shell_Setup method customized, you are ready to try it out. Select Test Application from the Run menu, and select Start Foundation from the File menu. Foundation will run through the startup process again, this time displaying the name of your database in the Startup Message.

Here are just a few of the many things that Foundation will set up for you, based on the tables and fields you have created, and the changes you make to the Fnd_Hook_Shell_Setup method:

- **Navigation Palette**

The first thing you will probably notice is that the Navigation Palette displays each of the visible tables you have created. Clicking on a table name in this palette will open a window to display the records in that table using the input and output forms you created. You can modify this palette using the Fnd_Hook_Shell_NavigationPalette hook.

• Custom About Box

The About menu item under the Apple () menu (Macintosh) or under the Help menu (Windows) displays the name of your database. Notice that the window includes your URL, and clicking on it will launch your browser and open that web page.

• Find Dialog

After you have opened a table window (by clicking a button on the Navigation Palette or selecting Open from the File menu) select Find from the Select menu. All of the visible, searchable fields from the current table are listed. If you would like to offer the end user a different selection of fields to query, modify the Fnd_Hook_Shell_Find method.

• Sort Dialog

Just like Find Dialog , the Sort Dialog (also available from the Select menu) will display all of the visible, sortable fields from the current table. If you would like a different selection of fields presented to the end user, modify the Fnd_Hook_Shell_Sort method.

When you are ready to move back to the Design environment, select Quit or Exit from the File menu, then select Return to Design mode from the Mode menu. Or you can just click on one of the Design environment windows to bring it to the front.

Still More Control

Take a look at the methods in Foundation that have names beginning with “Fnd_Hook_.” You can customize these methods to alter default sorting, selection filtering, and much more. Read each method’s comments for guidance. If you need more information, each of these methods is documented in the Component chapters of this manual that follow.

Messing with Components

No matter how much experience you have with 4th Dimension, you will want to read this chapter to learn how to deal with 4D Components.

Foundation 5 has been completely revised to comply with 4D v11 SQL's new component architecture. In previous versions of 4D, components were created and installed using a separate program called 4D Insider. Starting with v11, components are now just a special kind of 4D structure file (cannot contain tables) and installed by simply copying them to the Components folder next to your 4D structure file.

What this means for Foundation users is a great simplification in installation, use and upgrading. If you never modify any part of Foundation (Fnd_All.DB4), an upgrade will mean a simple replacement of a single file (Fnd_All.DB4) in your Components folder.

Now that 4D Insider is no longer with us, all of its issues are gone as well. Many of us struggled with 4D Insider and the many quirks it had in upgrading components. I know my 4D life will be much simpler with 4D v11 and components. Thanks, 4D!

What 4D Documentation Should I Read?

All of it? Someday, you should, but to use Foundation, you just need to know how to create tables, fields, forms and a few methods. You do not need to know anything about 4D components, but if you want to learn about them, Foundation is a good starting point. Please read the 4D v11 Upgrade and v11 Design Reference manuals for complete details on using Components in 4D.

Foundation is now shipping as a single component, but it has been written in a modular fashion so that you can take certain parts of Foundation and use them independently of the full Foundation Shell. There are some dependencies between the Foundation sub-components (for lack of a better term). For example, Fnd_Gen is used by just about all the other Foundation sub-components and, therefore, is required.

Sub-component Dependencies

Gary Boudreaux <garybx@gmail.com> has kindly laid out a complete list of the dependencies:

——— Fnd_Art ———

Fnd_Gen
Fnd_Wnd

——— Fnd_Bttn ———

Fnd_Gen

——— Fnd_Cmpt ———

Fnd_Dlg
Fnd_Gen
Fnd_List
Fnd_Pref
Fnd_Wnd

——— Fnd_Data ———

Fnd_Gen
Fnd_Text

——— Fnd_Date ———

Fnd_Gen

——— Fnd_Dict ———

Fnd_Gen

——— Fnd_Dlg ———

Fnd_Gen

Fnd_Wnd

——— Fnd_Find ———

Fnd_Gen
Fnd_Wnd

——— Fnd_Gen ———

None

——— Fnd_IO ———

Fnd_Dlg
Fnd_Gen
Fnd_Loc
Fnd_Menu
Fnd_Pref

Fnd_Rec
Fnd_Tlbr
Fnd_VS
Fnd_Wnd

———— Fnd_List ————

Fnd_Dlg
Fnd_Gen
Fnd_Wnd

———— Fnd_Loc ————

Fnd_Gen

———— Fnd_Log ————

None

———— Fnd_Menu ————

Fnd_Gen

Fnd_Loc

———— Fnd_Msg ————

Fnd_Gen

———— Fnd_Nav ————

Fnd_Gen

———— Fnd_Pref ————

Fnd_Gen

———— Fnd_Prnt ————

Fnd_Gen

———— Fnd_Pswd ————

Fnd_Gen

———— Fnd_Rec ————

Fnd_Dlg
Fnd_Gen
Fnd_Wnd

———— Fnd_Reg ————

Fnd_Gen
Fnd_Text
Fnd_Wnd

———— Fnd_RegG ————

Fnd_Gen
Fnd_Text

———— Fnd_Shell ————

Fnd_Art
Fnd_Dlg
Fnd_Gen
Fnd_IO
Fnd_List
Fnd_Loc
Fnd_Menu
Fnd_Pref
Fnd_Rec
Fnd_VS
Fnd_Wnd

———— Fnd_Sort ————

Fnd_Gen
Fnd_Wnd

———— Fnd_SqNo ————

Fnd_Dlg
Fnd_Gen
Fnd_Wnd

———— Fnd_Text ————

Fnd_Gen

———— Fnd_Tlbr ————

Fnd_Bttn
Fnd_Gen

———— Fnd_VS ————

Fnd_Gen

———— Fnd_Wnd ————

Fnd_Gen

The following is a list of sub-components in proper order to make the compiler happy (there are groupings of them that the order can be mixed, but this particular order should work):

Fnd_Gen
Fnd_Log
Fnd_Wnd
Fnd_Text
Fnd_Art
Fnd_Bttn
Fnd_Data
Fnd_Date
Fnd_Dict
Fnd_Dlg
Fnd_Find
Fnd_List
Fnd_Loc
Fnd_Menu
Fnd_Msg
Fnd_Nav
Fnd_Pref
Fnd_Prnt
Fnd_Pswd
Fnd_Rec
Fnd_Reg
Fnd_RegG
Fnd_Sort
Fnd_SqNo
Fnd_Tlbr
Fnd_VS
Fnd_Cmpt
Fnd_IO
Fnd_Shell

Art Component

(Fnd_Art)

The Foundation Art component is responsible for displaying the startup dialog and About box in the Foundation Shell. It is called the Art component because it does not really serve any functional purpose other than to make your application look good.

The Art component includes default forms for the startup dialog. You can change the look of the startup dialog by calling *Fnd_Art_SetStartupDialogForm* to tell the component to use your own form rather than the built-in form.

If you pass an empty form name to the *Fnd_Art_SetStartupDialogForm* method, the startup dialog will not be displayed when the Shell calls *Fnd_Art_StartupDialog*. You must still pass a pointer to the *Fnd_Art_SetStartupDialogForm* method, but it will be ignored.

The Art component also includes a default form for the About box. You can change the look of the About box by calling *Fnd_Art_SetAboutForm* to specify your own form.

Language Reference

Here are the routines in Foundation's Art component:

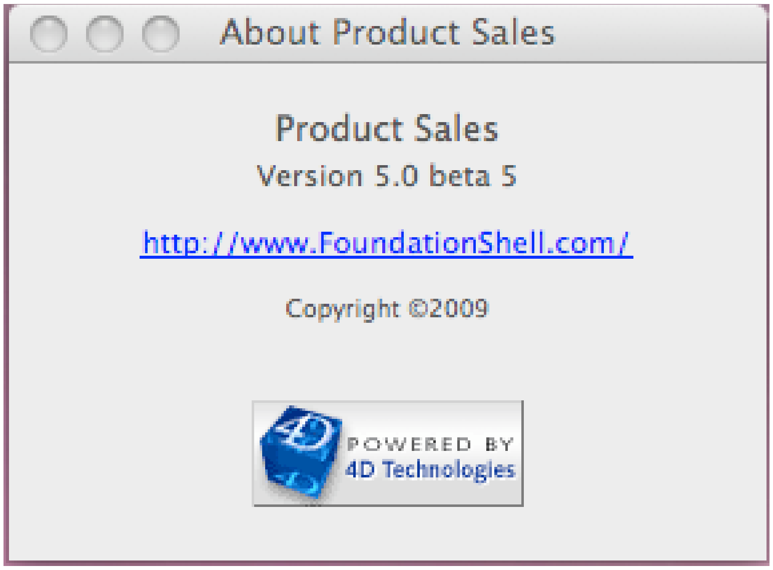
- Fnd_Art_About
- Fnd_Art_Info
- Fnd_Art_SetAboutForm
- Fnd_Art_SetStartupDialogForm
- Fnd_Art_StartupDialog
- Fnd_Art_StartupDialogClose
- Fnd_Art_StartupDialogFormMethod

Fnd_Art_About

Fnd_Art_About

Parameter	Type	Description
No parameters required.		

The *Fnd_Art_About* routine displays the About Box dialog in a new process.



If you are using the Foundation Shell, this routine is automatically installed as the About Box routine. The information displayed in this window is obtained from the values set up in the *Fnd_Hook_Shell_Setup* hook.

If you are not using the Foundation Shell, use *Fnd_Gen_SetDatabaseInfo* method to configure the information that will be displayed in this dialog. Then use 4D's **SET ABOUT** command to install this method as your application's About Box handler:

```
Fnd_Gen_SetDatabaseInfo ("NAME";"Super Accounting Program")
Fnd_Gen_SetDatabaseInfo ("VERSION";"5.0.7 beta 2")
Fnd_Gen_SetDatabaseInfo ("COPYRIGHT";"Copyright ©2006 Amazing Company, Inc.")
Fnd_Gen_SetDatabaseInfo ("URL";"http://www.AmazingCompany.com/")
SET ABOUT("About "+Fnd_Gen_GetDatabaseInfo ("NAME")+"...";"Fnd_Art_About")
```

Use *Fnd_Art_SetAboutForm* to use a custom form, rather than Foundation's default About form, to display when this method is called.

Fnd_Art_Info

Fnd_Art_Info (info requested) ➔ Text

Parameter	Type	Description
info requested	Text	Info desired
Function Result	Text	Response

This function returns the requested information about the Art component.

```
$version_t:=Fnd_Art_Info ("version")
```

The *Fnd_Art_Info* method will respond to these requests:

Request	Response	Example
name	The component's full name	Foundation Art
version	The component's version number	4.0.5 beta 4

This routine can also be called using the *Fnd_Gen_ComponentInfo* method without first testing to see if the component is installed:

```
$version_t:=Fnd_Gen_ComponentInfo ("Fnd_Art";"version")
```

See the *Fnd_Gen_ComponentInfo* method for more information.

Fnd_Art_SetAboutForm

Fnd_Art_SetAboutForm (->table; form name)

Parameter	Type	Description
table	Pointer	Pointer to the form's table
form name	Text	Name of the form

The *Fnd_Art_SetAboutForm* routine installs a form to use instead of the default About form. The window size of the About dialog is determined from the form size. The form should call the *Fnd_Gen_FormMethod* method, and these events must be enabled:

On Activate
On Close Box
On Outside Call

Call this method from the On Startup database method just before the call to *Fnd_Shell_OnStartup*. You can use the *Fnd_Gen_GetDatabaseInfo* method to get information to display in the form, such as the database name, version number, copyright information, etc.

Fnd_Art_SetStartupDialogForm

Fnd_Art_SetStartupDialogForm (->table; form name)

Parameter	Type	Description
table	Pointer	Pointer to the form's table
form name	Text	Name of the form

Fnd_Art_SetStartupDialogForm allows the developer to specify the form to display during startup.

This routine installs a form to use instead of Foundation's default startup form. The window size of the startup dialog is determined from the form size. The form should call the *Fnd_Art_StartupDialogFormMethod* method, and these events must be enabled:

On Load
On Timer
On Outside Call

You will need to call this method from the *On Startup* database method. Because this method does not run when you quit and restart Foundation without actually quitting 4D, you will only see this custom form the first time you open your database. However, your end users will always see your custom startup form.

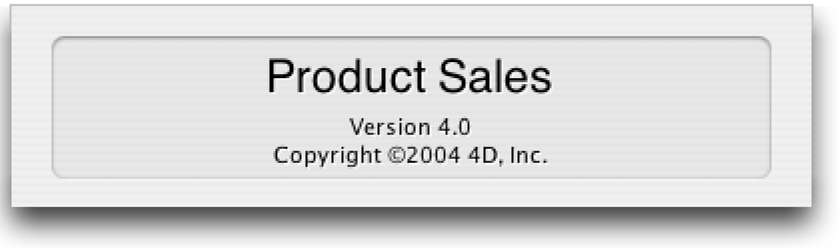
You can use the *Fnd_Gen_GetDatabaseInfo* method to get information to display in the form, such as the database name, version number, copyright information, etc.

Fnd_Art_StartupDialog

Fnd_Art_StartupDialog

Parameter	Type	Description
No parameters required.		

Fnd_Art_StartupDialog displays the startup window in a new process.



Product Sales

Version 4.0

Copyright ©2004 4D, Inc.

Close the dialog by calling *Fnd_Art_StartupDialogClose* method.

```
Fnd_Art_StartupDialog
  ` Run your database initialization code here.
Fnd_Art_StartupDialogClose
```

If *Fnd_Art_StartupDialogClose* does not get called (due to debugging or a code error), the startup dialog will close automatically after 30 seconds. You can also close the standard dialog by clicking the upper left corner of the dialog. There is a 10 pixel by 10 pixel invisible Cancel button hidden here.

Fnd_Art_StartupDialogClose

Fnd_Art_StartupDialogClose

Parameter	Type	Description
No parameters required.		

Fnd_Art_StartupDialogClose closes the startup message. This routine is safe to call even if no startup message was displayed.

The startup dialog is designed to stay open for at least three seconds once it is displayed, so this command may take a few seconds to execute as it waits for this three second delay. If the startup dialog has been displayed for at least three seconds before calling *Fnd_Art_StartupDialogClose,* the window will close immediately.

See the *Fnd_Art_StartupDialog* method for an example.

Fnd_Art_StartupDialogFormMethod

Fnd_Art_StartupDialogFormMethod

Parameter	Type	Description
No parameters required.		

The *Fnd_Art_StartupDialogFormMethod* routine must be called from the startup dialog's form method, even if a custom form is used.

Call this method from your custom startup dialog form (installed by calling *Fnd_Art_SetStartupDialogForm*). To work properly, the form must have these events enabled:

On Load
On Outside Call
On Timer

Buttons Component

(Fnd_Bttn)

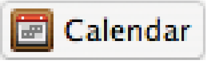
The Foundation Buttons component creates the rollover buttons used by the Foundation Toolbar component. These routines let you define an image and a text label, and then creates a multi-part image that can be used to display a rollover button using 4D's picture button form object.

The Foundation Buttons component is based on Mark Mitchenall's free Rollover Component <http://www.Mitchenall.com/> and is used with permission.

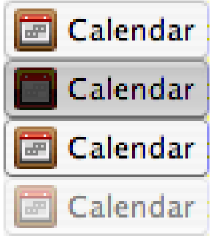
This component currently exists primarily to support the Toolbar component. When drawing toolbars, in most cases it will be easier to use the Toolbar component directly.

4D Picture Buttons

4th Dimension offers Picture Button form objects. You can assign an image to this button type, and 4D will use parts of the image to draw the button in various states. For example, to create a button that looks like this on a form...



...you could create an image like this:



The image above includes each state of the button. The first is the "active" or "default" state, where the button is enabled. The second is the "clicked" state, which is drawn when the user clicks on the button. The third column is the "roll over" state, which is displayed when the cursor is over the button, and the last image is the "disabled" state, which is displayed when the button is disabled.

Although you can design an image like this for each icon button in your project, this presents two problems. First, if you need to change the background behind the icon and label, you must modify every image in the database. Secondly, you cannot dynamically label buttons at runtime. This becomes especially problematic when localizing an interface for multiple languages.

Both of these problems are solved by using this component. The Foundation Buttons component dynamically creates images at runtime by combining an icon, dynamic text, and three parts of a background image (left, middle, and right) to create images that can be used with 4D Picture Button form objects. A small selection of background images are already built into the component, as is a collection of generic icons. So to create the image above, simply call the *Fnd_Bttn_GetPicture* routine:

```
Fnd_Bttn_GetPicture ("Fnd_Bttn_Calendar";"Calendar")
```

This picture could then be used by a 4D Picture Button. Of course, you would need to be careful to properly set the button's attributes in the Property List. And you would need to procedurally set the button's size at runtime, since the size of the image returned depends on the current button style settings, the text passed, and the button label length. To help you do all of this, you can instead just call the *Fnd_Bttn_CreateButton* routine:

```
Fnd_Bttn_CreateButton (->myButton;->myPicture;"Fnd_Bttn_Calendar";"Calendar")
```

However, this component will also let you use custom background and icon images for even greater control.

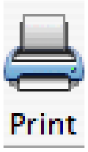
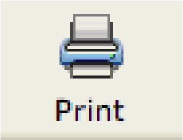
See the 4D Design Reference for complete information about using Picture Button form objects.

Built-In Button Styles

The Foundation Buttons component includes a few built-in background images. You can use these images by setting a button style for the current process before creating the buttons. The button style is set using the *Fnd_Bttn_Style* method.

Currently three button styles are available: Large, Small1, and Small2. Each of these button styles is available in a Windows and a Macintosh look.

The Large style is designed to draw platform specific toolbar buttons. Here's an example of a button created with the Large style:

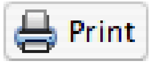
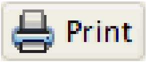
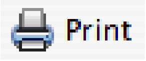
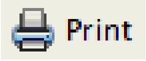
Macintosh Platform	Windows Platform
	

Note that the platform is determined by the *Fnd_Bttn_Platform* command, not by the actual platform in use. So either platform style can be displayed on either platform. By default, the platform is set to "Auto," so it will detect and use the actual platform.

The Large button style is designed to be displayed on a matching toolbar background image. These background images (one for Macintosh and one for Windows), along with matching dividers, are included with the Foundation Toolbar component.

The Large style for Windows requires approximately a 24x24 pixel icon. The Macintosh style requires a 32x32 icon. If a built-in icon is used, the size is selected for you automatically.

The Small1 and Small2 styles are designed to be displayed on the default Windows or Macintosh background. Both button styles are 24 pixels tall, but vary in width depending on the contents displayed.

	Macintosh	Windows
Small1		
Small2		

These button styles are designed for a 16x16 icon. However, the icon image does not need to be exactly 16x16.

Built-in Icons

Built into this component are over 50 high-quality icons that you can use when creating buttons. Each icon is included in three sizes: 16, 24, and 32 pixels.

Fnd_Bttn_Address
Fnd_Bttn_Alarm
Fnd_Bttn_Arrows
Fnd_Bttn_Backup
Fnd_Bttn_Browser
Fnd_Bttn_Calculator
Fnd_Bttn_Calendar
Fnd_Bttn_CardBlocks
Fnd_Bttn_CardMagnifier
Fnd_Bttn_CardMinus
Fnd_Bttn_CardPlus
Fnd_Bttn_CardPrinter
Fnd_Bttn_Cards
Fnd_Bttn_Card
Fnd_Bttn_Cart
Fnd_Bttn_Chart
Fnd_Bttn_Checklist
Fnd_Bttn_Connect
Fnd_Bttn_Disconnect
Fnd_Bttn_Diskette
Fnd_Bttn_DocMag
Fnd_Bttn_DocPlus
Fnd_Bttn_Document
Fnd_Bttn_DownArrow
Fnd_Bttn_Duplicate
Fnd_Bttn_Earth
Fnd_Bttn_Envelope
Fnd_Bttn_Export
Fnd_Bttn_Fax
Fnd_Bttn_Forms
Fnd_Bttn_Filter
Fnd_Bttn_Gear
Fnd_Bttn_GreenPlus
Fnd_Bttn_Import
Fnd_Bttn_IndexCards
Fnd_Bttn_Info
Fnd_Bttn_Letters
Fnd_Bttn_Lock
Fnd_Bttn_MacDelete
Fnd_Bttn_MacFind
Fnd_Bttn_MacNew
Fnd_Bttn_MacPrint
Fnd_Bttn_MacShowAll
Fnd_Bttn_MacSort
Fnd_Bttn_Magnifier

Fnd_Bttn_Mail
Fnd_Bttn_Notepad
Fnd_Bttn_Options
Fnd_Bttn_People
Fnd_Bttn_Person
Fnd_Bttn_Phone
Fnd_Bttn_Printer
Fnd_Bttn_Question
Fnd_Bttn_RedX
Fnd_Bttn_Script
Fnd_Bttn_Star
Fnd_Bttn_Stop
Fnd_Bttn_Toolbox
Fnd_Bttn_Unlock
Fnd_Bttn_UpArrow
Fnd_Bttn_Wand
Fnd_Bttn_WinDelete
Fnd_Bttn_WinFind
Fnd_Bttn_WinNew
Fnd_Bttn_WinPrint
Fnd_Bttn_WinShowAll
Fnd_Bttn_WinSort
Fnd_Bttn_Wrench

To use one of these icons when creating a button, just pass the button name displayed above to one of the routines in this component. The Foundation Buttons component will select the appropriate size of the icon based on the button style.

Creating Custom Icon Images

In addition to using the icons included with the Foundation Buttons component, you can also create buttons that use your own custom artwork. The Foundation Buttons component routines can accept one of the icon names listed above, or the name of a series of pictures contained in the 4D Picture Library. When using a custom icon name, the Buttons component will look for an image with the specified icon name plus "_a" to use when drawing the button in its "active" state. The icon name plus "_b" is used to draw the button in the clicked state. And the icon name plus "_c" is used to draw the button in the disabled state.

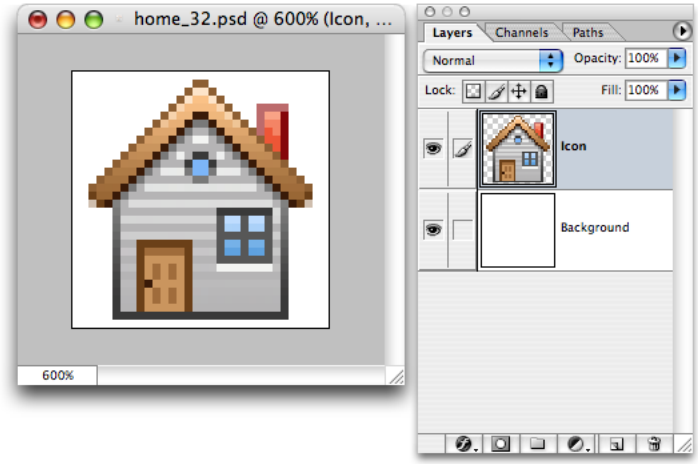
For example, to create a custom button icon named "Button _Home" we would actually need to create three images in the 4D Picture library named "Button_Home_a," "Button_Home_b," and "Button_Home_c":

Button_Home_a
Button_Home_b
Button_Home_c

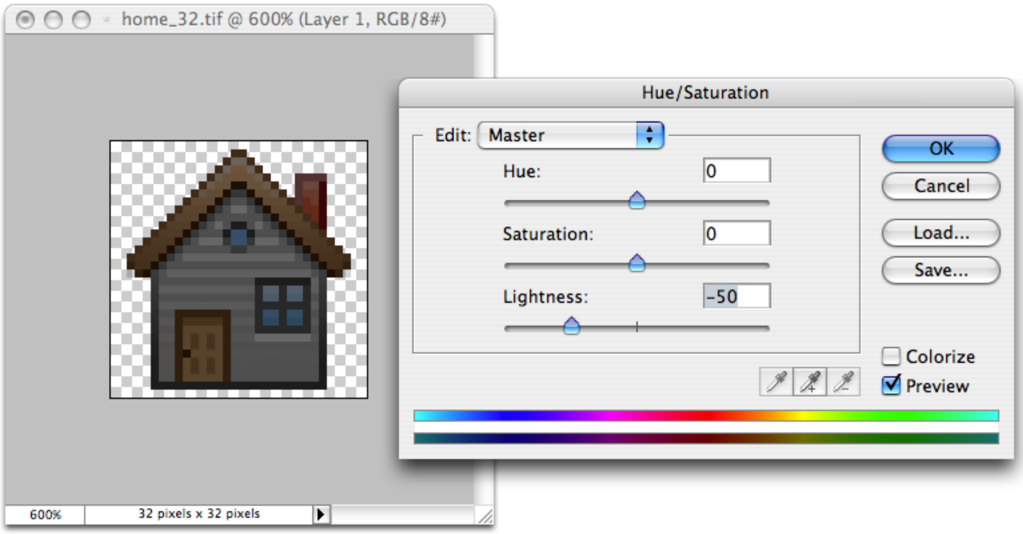
All of the icons should include a white background.

The "clicked" and "disabled" icons can easily be created from the original icon artwork using a good image editing application. The instructions below will describe one way the image could be created using Adobe Photoshop.

Start with an icon image with no background. Create a white background in a separate layer.



Create two more copies of the icon layer to use as the "clicked" and "disabled" versions. Select one of these and then select the **Image->Adjustments->Hue/Saturate** menu item. The Hue/Saturation dialog will appear. Enter -50 into the Lightness field. This will darken the image. Click **OK** and name this layer as the "clicked" layer.



Edit: Master

Hue: 0

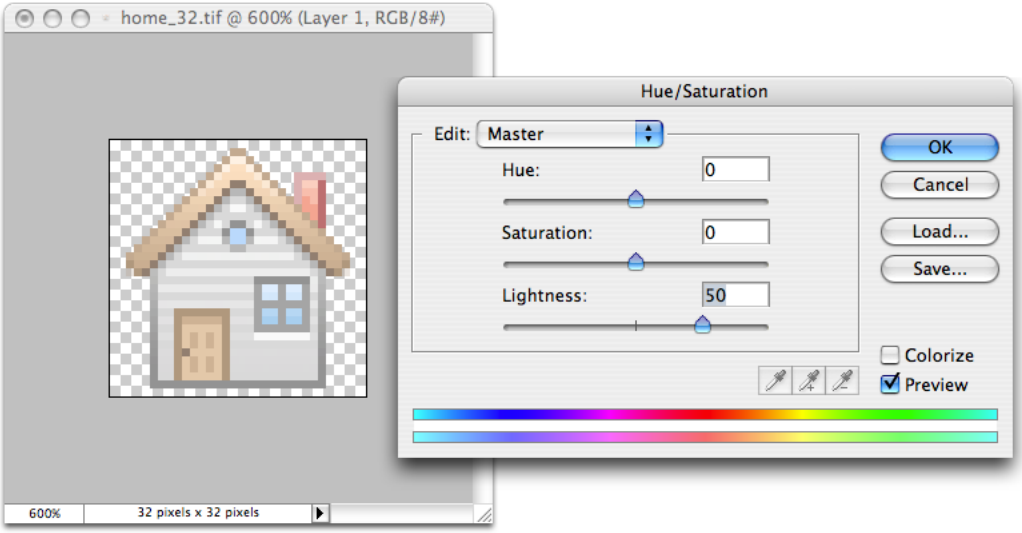
Saturation: 0

Lightness: -50

☐ Colorize

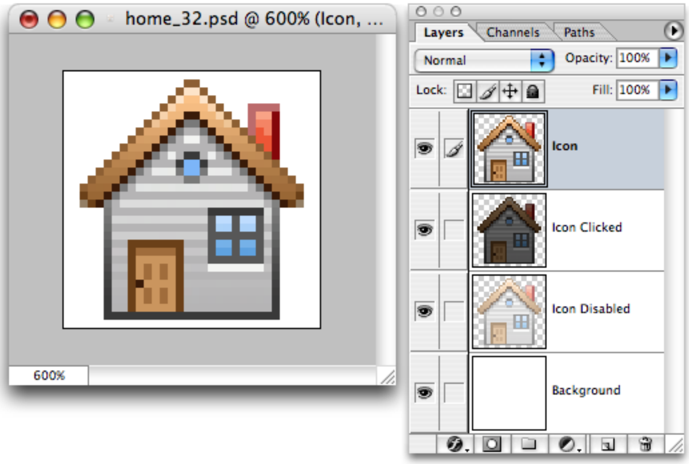
☑ Preview

Select another icon layer and again select the **Image->Adjustments->Hue/Saturate** menu item. This time in the Hue/Saturation dialog, enter 50 into the Lightness field and click **OK**. This will lighten the icon. Name this layer as the "disabled" icon.



Edit: Master
Hue: 0
Saturation: 0
Lightness: 50
☐ Colorize
☑ Preview

You should now have three icons and a background image.



Normal
Opacity: 100%
Fill: 100%

Hide the "clicked" and "disabled" buttons, select the first icon, then select the **Edit->Copy Merged** menu command. This will copy both the icon and the background to the clipboard. Switch to 4D and open the Picture Library. Paste the image into the Picture Library and give it a name ending with "_a."

Switch back to Photoshop and use the same technique to copy the clicked and disabled icon images (including the background) into the 4D Picture Library. The "clicked" image should have the same name as the first image, except that it should end with "_b." The "disabled" image should end with "_c." All three images should have exactly the same height and width.

Do not use the "Fnd_Bttn" prefix when creating custom button icons. This may cause unexpected results.

Language Reference

Here are the routines in Foundation’s Buttons component:

- Fnd_Bttn_CreateButton
- Fnd_Bttn_GetPicture
- Fnd_Bttn_GetPictureName
- Fnd_Bttn_GetPictureWidth
- Fnd_Bttn_Info
- Fnd_Bttn_Platform
- Fnd_Bttn_SetBackgroundPict
- Fnd_Bttn_SetTextProperties
- Fnd_Bttn_Style

Fnd_Bttn_CreateButton

Fnd_Bttn_CreateButton (->button object; ->picture; icon name; label)

Parameter	Type	Description
button object	Pointer	A picture button form object
picture	Pointer	A picture variable
icon name	Text	Name of the icon to use from the 4D picture library
label	Text	Label for the button

Use this routine in the On Load phase to quickly set up a picture button on a 4D form. Pass this routine a pointer to a picture button form object, a picture variable to hold the button image, the icon name to use and a label to use.

```
Case of
  : (Form event=On Load )
    C_PICTURE(myPicture)
    Fnd_Bttn_Style ("Small2")
    Fnd_Bttn_CreateButton (->myButton;->myPicture;"Button_Home";"Home")
End case
```

The style of the button created will depend on the current button settings set by the *Fnd_Bttn_Platform* and *Fnd_Bttn_Style* commands.

The first parameter is a pointer to a Picture Button form object. The size of this form object is unimportant, as it will be resized by this call. The object’s attributes do not need to be specified — this routine will set them to properly display a rollover picture button.

The picture variable is used as storage space for the picture button’s image. It must be declared as a picture variable before calling this method. You will not need to access this variable to use the button, however, you may clear the contents of this variable when the button is no longer needed.

After this call, the picture button will be resized to properly fit the button image. This varies depending on the current button style, the size of the icon, and the width of the button label. If necessary, you can procedurally move the button using 4D’s **MOVE OBJECT** command.

You can control the button's attributes using 4D's **ENABLE BUTTON** and **DISABLE BUTTON** commands:

```
DISABLE BUTTON(myButton)
```

Use the form object's method to detect a click on the button:

```
Case of
  : (Form event=On Clicked )
     ` Put your code here to handle the click.
End case
```

Fnd_Bttn_GetPicture

Fnd_Bttn_GetPicture (icon name; button label) ➔ Picture

Parameter	Type	Description
icon name	Text	Name of the icon to use from the 4D picture library
button label	Text	Label for the button
Function result	Picture	Picture to use for a 4D picture rollover button

This routine returns a picture of a rollover based on the parameters you pass for the icon, text, size and type. The Buttons component caches the images it creates, so if the picture already exists in the cache, it is retrieved from there instead of being generated again.

```
Fnd_Bttn_GetPicture ("Fnd_Bttn_Calendar";"Events")
```

The icon name can be one of the icons pre-installed in the Button component (e.g. "Fnd_Bttn_Alarm"), or it can be the name of a series of pictures in the 4D Picture Library. This routine will look for an image with the specified icon name plus "_a" to use when drawing the button in its "active" state. The icon name plus "_b" is used to draw the button in the clicked state. And the icon name plus "_c" is used to draw the button in the disabled state. See the details at the beginning of this chapter.

Fnd_Bttn_GetPictureName

Fnd_Bttn_GetPictureName (icon name; button label) ➔ Text

Parameter	Type	Description
icon name	Text	Name of the icon to use from the 4D picture library
button label	Text	Label for the button
Function result	Text	The cache picture name

This routine returns the name that is used to store the specified picture button image in the cache. It has been exposed specifically for use only by the Toolbar component. This routine will be removed in a future update.

Fnd_Bttn_GetPictureWidth

Fnd_Bttn_GetPictureWidth (icon; text) ➔ Number

Parameter	Type	Description
icon	Text	Name of the icon
text	Text	Button label
Function result	Number	Width of the picture

Fnd_Bttn_GetPictureWidth returns the width of the rollover picture that uses the specified icon and text.

```
$buttonWidth:=Fnd_Bttn_GetPictureWidth ("Fnd_Bttn_Wrench";"Special Functions")
```

The size returned will depend on the button platform and style currently set for the current process.

Although this routine does not return the image, the composite button image is generated and stored in the cache.

Fnd_Bttn_Info

Fnd_Bttn_Info (info requested) ➔ Text

Parameter	Type	Description
info requested	Text	Info desired
Function Result	Text	Response

This routine returns the requested information about the component.

```
$version_t:=Fnd_Bttn_Info ("version")
```

The *Fnd_Bttn_Info* method will respond to these requests:

Request	Response	Example
name	The component's full name	Foundation Buttons
version	The component's version number	4.1

This component can also be called using the *Fnd_Gen_ComponentInfo* method without first testing to see if the component is installed:

```
$version_t:=Fnd_Gen_ComponentInfo ("Fnd_Bttn";"version")
```

See the *Fnd_Gen_ComponentInfo* method for more information.

Fnd_Bttn_Platform

Fnd_Bttn_Platform (platform) ➔ Text

Parameter	Type	Description
platform	Text	"Auto" or "Mac" or "Win"
Function result	Text	Current platform setting

Call this routine to specify the platform style to use when creating the next button image. The platform value must be either “Auto,” “Mac,” or “Win.”

```
Fnd_Bttn_Platform ("Mac")  ` Draw Macintosh style buttons regardless of the platform.
```

The “Auto” setting lets the component select the appropriate platform setting for the current platform.

This routine can also be called as a function to determine the current platform setting:

```
$bttnPlatform:=Fnd_Bttn_Platform
```

Only “Mac” or “Win” will be returned. “Auto” will not be returned.

This command affects only the current process.

Fnd_Bttn_SetBackgroundPict

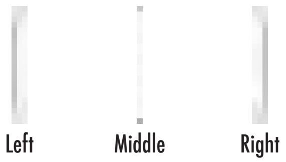
Fnd_Bttn_SetBackgroundPict (left; middle; right)

Parameter	Type	Description
left	Picture	Left edge of the button
middle	Picture	Middle of the button (generally 1 pixel wide)
right	Picture	Right edge of the button

If you would like to design your own artwork to be used as a button background, pass images to these button parts using this method.

```
GET PICTURE FROM LIBRARY("MyButtonBackgroundLeft";$leftImage)
 GET PICTURE FROM LIBRARY("MyButtonBackgroundMiddle";$middleImage)
GET PICTURE FROM LIBRARY("MyButtonBackgroundRight";$rightImage)
 Fnd_Bttn_SetBackgroundPict ($leftImage;$middleImage;$rightImage)
```

For example these images are used to create the Small1 button for Macintosh (enlarged):



All three images must be exactly the same height. The middle image must be exactly 1 pixel wide. It will be replicated as needed to provide a final button background image wide enough for the specified icon and text.

The middle image is required, but the outer images (left and right) are optional. For example, the Macintosh style Large button uses only a middle image to create the button background. No left or right ends to the button are used.

Fnd_Bttn_SetTextProperties

Fnd_Bttn_SetTextProperties (font name; font size; font style)

Parameter	Type	Description
font name	Text	Font name
font size	Longint	Font size
font style	Longint	Font style (use 4D constants)

Fnd_Bttn_SetTextProperties sets the text properties for the button labels. Call this routine before calling the *Fnd_Bttn_CreateButton*, *Fnd_Bttn_GetPicture*, or *Fnd_Bttn_GetPictureWidth* methods.

```
Fnd_Bttn_SetTextProperties ("Avant Garde";15;Plain )
```

4D’s font style constants should be used for the font style parameter.

Fnd_Bttn_Style

Fnd_Bttn_Style ({desired style name}) ➔ Text

Parameter	Type	Description
desired style name	Text	Desired button style name (optional)
Function result	Text	Actual button style name

Pass a style name to set the button style to use when drawing the next button image. See the beginning of this chapter for examples of each style.

```
Fnd_Bttn_Style ("Small2")
```

The style name must be either “Large,” “Small1,” or “Small2.” Call this routine before calling the *Fnd_Bttn_CreateButton*, *Fnd_Bttn_GetPicture*, or *Fnd_Bttn_GetPictureWidth* methods. This command affects only the current process.

This method can also be called as a function to determine the current style setting:

```
$style:=Fnd_Bttn_Style
```

Data Component

Fnd_Data

The Data component provides a series of utility methods for dealing with common data formats. Phone numbers, postal codes, web URLs, e-mail addresses, and personal names are handled by these routines. You can call these commands anywhere in your database.

Language Reference

Here are the routines in Foundation's Data component:

Fnd_Data_EmailAddressError
Fnd_Data_FormatBypassKey
Fnd_Data_FormatError
Fnd_Data_FormatPhone
Fnd_Data_FormatPostalCode
Fnd_Data_FormatText
Fnd_Data_FormatWebURL
Fnd_Data_Info
Fnd_Data_ParseName

Fnd_Data_EmailAddressError

Fnd_Data_EmailAddressError (email address) ➜ Longint

Parameter	Type	Description
email address	Text	e-mail address to verify
funtion result	Longint	error number

The *Fnd_Data_EmailAddressError* function checks the validity of an e-mail address. Function returns 0 if e-mail is valid and 1 if invalid.

```
If (Fnd_Data_EmailAddressError (Self->)>0)
` Email address is invalid
End if
```

Fnd_Data_FormatBypassKey

Fnd_Data_FormatBypassKey ({bypass key}) ➜ Longint

Parameter	Type	Description
bypass key	Longint	key mask used to bypass data formatting (optional)
current bypass key	Longint	current bypass key (default is Option key mask)

The *Fnd_Data_FormatBypassKey* routine allows the user to bypass data entry formatting. The bypass key defaults to Option key mask while other acceptable values are Command key mask, Control key mask, Shift key mask, or Caps lock key mask.

```
$bypass_key_i:= Fnd_Data_FormatBypassKey (Shift key mask)
```

Fnd_Data_FormatError

Fnd_Data_FormatError ➜ Longint

Parameter	Type	Description
Function result	Longint	last formatting error number

The *Fnd_Data_FormatError* function returns the last formatting error.

This function allows the developer to detect if any of the data formatting routines had a problem trying to format the data.

```
$error_i:= Fnd_Data_FormatError
```

Fnd_Data_FormatPhone

Fnd_Data_FormatPhone (phone number) ➔ Text

Parameter	Type	Description
phone number	Text	phone number to format
Function result	Text	formatted phone number

The *Fnd_Data_FormatPhone* function formats text of a phone number. Function returns the formatted phone number.

```
Self->:=Fnd_Data_FormatPhone (Self->)
```

Fnd_Data_FormatPostalCode

Fnd_Data_FormatPostalCode (postal code) ➔ Text

Parameter	Type	Description
postal code	Text	postal code to format
Function result	Text	formatted postal code

The *Fnd_Data_FormatPostalCode* function formats text of a postal code. Function returns the formatted postal code.

```
Self->:=Fnd_Data_FormatPostalCode (Self->)
```

Fnd_Data_FormatText

Fnd_Data_FormatText (text) ➔ Text

Parameter	Type	Description
text	Text	text to format
Function result	Text	formatted text

The *Fnd_Data_FormatText* function formats text by stripping any leading or trailing spaces, removes any double spaces from within the text and capitalizes the first letter of each word, unless the user holds down the bypass key (default Option/Alt). Function returns the formatted text.

```
Self->:=Fnd_Data_FormatText (Self->)
```

Fnd_Data_FormatWebURL

Fnd_Data_FormatWebURL (url) ➔ Text

Parameter	Type	Description
url	Text	url to format
Function result	Text	formatted url

The *Fnd_Data_FormatWebURL* function formats text of a web URL. Function returns the formatted web URL.

```
Self->:=Fnd_Data_FormatWebURL (Self->)
```

Fnd_Data_Info

Fnd_Data_Info (info requested) ➔ Text

Parameter	Type	Description
info requested	Text	Info desired
Function Result	Text	Response

This function returns the requested information about the Data component.

```
$version_t:=Fnd_Data_Info ("version")
```

The *Fnd_Data_Info* method will respond to these requests:

Request	Response	Example
name	The component's full name	Foundation Data
version	The component's version number	4.0.5 beta 4

This routine can also be called using the *Fnd_Gen_ComponentInfo* method without first testing to see if the component is installed:

```
$version_t:=Fnd_Gen_ComponentInfo ("Fnd_Data";"version")
```

See the *Fnd_Gen_ComponentInfo* method for more information.

Fnd_Data_ParseName

Fnd_Data_ParseName (fullname; ->first name{; ->last name{; ->middle initial}})

Parameter	Type	Description
fullname	Text	fullname to parse
first name	Pointer	pointer to person's first name
last name	Pointer	pointer to person's last name (optional)
middle initial	Pointer	pointer to person's middle initial (optional)

The *Fnd_Data_ParseName* command parse text of a person's name. The command attempts to split the person's full name into separate parts while removing titles such as Mr. Mrs. Jr. Sr. and M.D.

```
Fnd_Data_ParseName (Self->;->[Contact]FirstName;->[Contact]LastName)
```

Date Component

Fnd_Date

The Date component provides utility routines for formatting and filtering dates. You can call these commands from anywhere in your database.

Language Reference

Here are the routines in Foundation's Date component:

- Fnd_Date_Calendar
- Fnd_Date_DateAndTimeToISO
- Fnd_Date_DateToString
- Fnd_Date_EndOfMonth
- Fnd_Date_EntryFilter
- Fnd_Date_Info
- Fnd_Date_ISOtoDate
- Fnd_Date_ISOtoTime
- Fnd_Date_MonthName
- Fnd_Date_RoundTime
- Fnd_Date_StringToDate
- Fnd_Date_SystemDateFormat
- Fnd_Date_YearMonthDayToDate

Fnd_Date_Calendar

Fnd_Date_Calendar (->date field or variable)

Parameter	Type	Description
field or variable	Pointer	Pointer to the field or variable

The Fnd_Date_Calendar routine displays a small calendar window and allows the user to select a date which is stored in the field or variable that the parameter points to. The OK variable is set to 1 if a date is selected.

```
$Ok_b:= Fnd_Date_Calendar(->[Contact]Birthdate)
```

Fnd_Date_DateAndTimeToISO

Fnd_Date_DateAndTimeToISO (date{; time}) ➜ Text

Parameter	Type	Description
date	Date	A date value to convert
time	Time	A time value to convert (optional)
Function result	Text	ISO date formatted as text

This function returns an ISO 8601 formatted date or date-time value. If only a date is supplied, a calendar date is returned in the format YYYY-MM-DD. If both a date and time are supplied, a date-time value is returned in the format YYYY-MM-DDThh:mm:ss. If both are passed but the date is !00/00/00! then only the time is returned in the format hh:mm:ss.

For more information, see:

<http://www.iso.org/iso/en/prods-services/popstds/datesandtime.html>

```
$datetimestamp_t:= Fnd_Date_DateAndTimeToISO(today_d;time_h)
$invoicedate_t:=Fnd_Date_DateToString ([Invoice]InvoiceDate)
```

Fnd_Date_DateToString

Fnd_Date_DateToString (date{; relative to date{; format}}) ➜ Text

Parameter	Type	Description
date	Date	Date to convert
relative to date	Date	Relative date (optional)
format	Text	Date format (optional)
Function result	Text	Date converted to text

This function returns "Today", "Tomorrow", or "Yesterday", if appropriate, and is localized if the Localization component is available, otherwise returns a short date string by default. If relative to date is included, "Today", "Tomorrow", or "Yesterday" will be relative to that date. Date will be formatted as a short date string unless the date format is passed.

```
$invoicedate_t:=Fnd_Date_DateToString ([Invoice]InvoiceDate)
```

Fnd_Date_EndOfMonth

Fnd_Date_EndOfMonth (date) ➔ Date

Parameter	Type	Description
date	Date	Date to find last day of the month
Function result	Date	Last day of the month of date parameter

This function returns the last day of the month passed.

```
$endofthemonth_d:=Fnd_Date_EndOfMonth ([Invoice]InvoiceDate)
```

Fnd_Date_EntryFilter

Fnd_Date_EntryFilter (->date field or variable)

Parameter	Type	Description
date	Pointer	Pointer to date field or variable
Function result	Date	Last day of the month of date parameter

This method interprets what has been entered as a date. If the function key is pressed, the following actions take place.

Function	Action
+ or =	Add a day
-	Subtract a day
T	Today
M	First of the Month
N	Middle of the moNth
H	End of the montH
Y	First of the Year
R	End of the yeaR
C	Display the calendar

Place in the object method of the date field or variable. Be sure to enable the object's On Before Keystroke and On Data Change events.

```
Fnd_Date_EntryFilter(Self)
```

Fnd_Date_Info

Fnd_Date_Info (info requested) ➔ Text

Parameter	Type	Description
info requested	Text	Info desired
Function Result	Text	Response

This function returns the requested information about the Date component.

```
$version_t:=Fnd_Date_Info ("version")
```

The *Fnd_Date_Info* method will respond to these requests:

Request	Response	Example
name	The component's full name	Foundation Date
version	The component's version number	4.0.5 beta 4

This routine can also be called using the *Fnd_Gen_ComponentInfo* method without first testing to see if the component is installed:

```
$version_t:=Fnd_Gen_ComponentInfo ("Fnd_Date";"version")
```

See the *Fnd_Gen_ComponentInfo* method for more information.

Fnd_Date_ISOtoDate

Fnd_Date_ISOtoDate (date string) ➔ Date

Parameter	Type	Description
date string	Text	ISO 8601 formatted date
Function result	Date	Converted date

This function returns the date value from an ISO 8601 formatted date. The input can be in either of these formats: YYYY-MM-DD or YYYY-MM-DDThh:mm:ss

```
$ISOdate_d:=Fnd_Date_ISOtoDate ("1997-01-12")
```

Fnd_Date_ISOtoTime

Fnd_Date_ISOtoTime (date-time string) ➔ Time

Parameter	Type	Description
date-time string	Text	ISO 8601 formatted date-time string

Function result	Time	Converted time

This function returns the time value from an ISO 8601 formatted date-time string. The input can be in either of these formats: YYYY-MM-DDThh:mm:ss or hh:mm:ss

```
$ISOdatetime_time:=Fnd_Date_ISOtoTime ("10:23:45")
```

Fnd_Date_MonthName

Fnd_Date_MonthName (month number) ➔ Text

Parameter	Type	Description
month number	Longint	Month number 1 through 12
Function result	Text	Converted time

This function returns name of the month with the specified number. This routine will use Foundation's Localization routine, if it is available, to return the localized month name. If the Fnd_Loc component is not available, the name will be localized using 4D's language.

```
$monthname_t:=Fnd_Date_MonthName (10)
```

Fnd_Date_RoundTime

Fnd_Date_RoundTime (time {; option}) ➔ Time

Parameter	Type	Description
time	Time	Time to round
option	Longint	Rounding style
Function result	Time	Rounded time

This function rounds the time based on the rounding option, default is to round to the nearest quarter hour. If option is 1, round up or down to the nearest quarter hour. If option is 2, round down to the nearest quarter hour.

```
$time_time:=Fnd_Date_RoundTime (current time)
```

Fnd_Date_StringToDate

Fnd_Date_StringToDate (date as text{; date format}) ➔ Date

Parameter	Type	Description
date as text	Text	Text of date to be converted to a date
date format	Text	Date format (optional)
Function result	Date	Converted date

This function returns the date equivalent of the string passed. The input can be in any of these formats: M/D/Y, D/M/Y, Y/M/D and Y/D/M. Two digit years are resolved as per the **SET DEFAULT CENTURY** command. Dates can be entered with any character as a delimiter. If no year is entered, the current year is assumed. If only a one or two digit number is entered, the current month is assumed. If no delimiter is used, the following formats are supported (M/D/Y format):

070476 Returns a date of 07/04/76
07041976 Returns a date of 07/04/1976
0704 Returns 07/04/00 (If this is in fact the year 2000)
04 or 4 Returns the 4th day of the current month
74, 704 and 074 are not supported
"Today" returns the $today_date
"Yesterday" returns the $today_date -1
"Tomorrow" returns the $today_date +1

```
$date_d:=Fnd_Date_StringToDate ("07/04/76")
```

Fnd_Date_SystemDateFormat

Fnd_Date_SystemDateFormat ➜ Text

Parameter	Type	Description
Function result	Text	Current system's date format

This function returns a string that can be used in If and Case statements to test the current system's date format. It will return Y, M, and D based in the following format:

"M/D/Y" = Month/Day/Year
"D/M/Y" = Day/Month/Year

```
$systemdateformat_t:=Fnd_Date_SystemDateFormat
```

Fnd_Date_YearMonthDayToDate

Fnd_Date_YearMonthDayToDate (year; month; day) ➜ Date

Parameter	Type	Description
year (2 or 4 digits)	Longint	Year to be converted to a date
month	Longint	Month to be converted to a date
day	Longint	Day to be converted to a date
Function result	Date	Converted date

This function returns the date equivalent of the year, month and day passed.

```
$date_d:=Fnd_Date_YearMonthDayToDate (2001;12;3)
```

Dictionary Component

(Fnd_Dict)

The optional Foundation Dictionary component is basically a lookup table feature. It allows you to create collections of data (values), with each item tagged with a unique name (keys). The values can be added to and retrieved from the collection in any order. Each collection, or dictionary, is referenced by a unique reference number. Dictionaries are an ideal way to manage multiple instances of preferences, data structures, and parameters. Dictionaries can also be saved to and retrieved from records and disk files. You can all do this without polluting 4D's process variable namespace.

This component works similarly to Aparajita Fishman's ObjectTools plugin. You create a dictionary, get a reference to it, and start adding key/value pairs. The component uses reference counting to help manage memory, and uses XML for storage. The code is simple and lightweight.

For example, imagine you have a method to which you need to pass more than 50 parameters. Or you want to pass a variable number of parameters of differing types, depending on the value of the first parameter. You cannot do these things with native 4D, but you can using the Dictionary component. Just create and pass a dictionary with any combination of data.

This component was written by Rob Leveaux of Pluggers Software: <http://www.pluggers.nl/>

Creating Dictionaries

You create a new dictionary by calling the *Fnd_Dict_New* function. This function will return a reference number that you'll use for all subsequent access to the dictionary and its contents.

```
$dict_i:=Fnd_Dict_New
```

You can optionally pass a name for the dictionary.

```
$dict_i:=Fnd_Dict_New ("NewContact")
```

By naming a dictionary, you can later get the dictionary's reference number from other parts of your code.

```
$dict_i:=Fnd_Dict_ID ("NewContact")
```

Dictionaries are shared by all processes. So a dictionary created in one process can be accessed by any other process.

Storing and Retrieving Values

Once you have created a dictionary, you will store values in the dictionary. A dictionary can store any of these value types:

Text or String
Integer or Long Integer
Real
Boolean
Date
Time
Pointer
Text or String Array
Integer or Long Integer Array
Real Array
Boolean Array
Date Array
Pointer Array

You will assocate a name to each value you want to store. These names are referred to as the "key" to the value. The key can be up to 32,000 characters.

For example, to store the number 300.56, we might select a key name of "Item Cost." We can then store this value in the dictionary using the *Fnd_Dict_SetReal* command:

```
Fnd_Dict_SetReal ($dict_i;"Item Cost";300.56)
```

Later, we can retrieve this value using the *Fnd_Dict_GetReal* function:

```
$value_r:= Fnd_Dict_GetReal ($dict_i;"Item Cost")
```

Reference Counting

To help you manage the memory required by a dictionary, the Dictionary module uses a common technique called “reference counting.” Each dictionary contains a counter that keeps track of how many methods or processes are using it. When a dictionary is created (or loaded) its reference count is set to 1. Each call you make to the *Fnd_Dict_Retain* method increments this counter. And each call made to *Fnd_Dict_Release* decrements this counter. Once the counter reaches zero, its contents are deleted to free the memory it was using.

So for each dictionary that you create or retain, you should be sure to release it at some point.

For example, here is a routine that creates a new dictionary, stores it in a BLOB, then releases it:

```
$dict_i:=Fnd_Dict_New
Fnd_Dict_SetBoolean ($dict_i;"BoolValue";True)
Fnd_Dict_SetText ($dict_i;"TextValue";"Example")
Fnd_Dict_SaveToBlob ($dict_i;->MyDict_blob)
Fnd_Dict_Release ($dict_i)
```

Without the call to *Fnd_Dict_Release*, the dictionary and the data that it stores will remain in memory until you quit 4D.

More Information

See this Wikipedia article for more information about software dictionaries:
<http://en.wikipedia.org/wiki/Associative_array>

See this Wikipedia article for more information about reference counting:
<http://en.wikipedia.org/wiki/Reference_counting>

Language Reference

Here are the routines in Foundation's Dictionary component:

Fnd_Dict_DataType
Fnd_Dict_GetArray
Fnd_Dict_GetBoolean
Fnd_Dict_GetDate
Fnd_Dict_GetLongint
Fnd_Dict_GetPointer
Fnd_Dict_GetReal
Fnd_Dict_GetText
Fnd_Dict_GetTime
Fnd_Dict_HasKey
Fnd_Dict_ID
Fnd_Dict_Info
Fnd_Dict_IsValid
Fnd_Dict_ItemCount
Fnd_Dict_Keys
Fnd_Dict_LoadFromBlob
Fnd_Dict_LoadFromFile
Fnd_Dict_Name
Fnd_Dict_New
Fnd_Dict_Release
Fnd_Dict_Remove
Fnd_Dict_Retain
Fnd_Dict_RetainCount
Fnd_Dict_SaveToBlob
Fnd_Dict_SaveToFile
Fnd_Dict_SetArray
Fnd_Dict_SetBoolean
Fnd_Dict_SetDate
Fnd_Dict_SetLongint
Fnd_Dict_SetPointer
Fnd_Dict_SetReal
Fnd_Dict_SetText
Fnd_Dict_SetTime
Fnd_Dict_Values

Fnd_Dict_DataType

Fnd_Dict_DataType (dict ID; key name) ➔ Number

Parameter	Type	Description
dict ID	Number	A dictionary ID
key name	Text	A key name
Function Result	Number	The data type

This routine returns the data type of a key. The data type is a number which corresponds to 4D's standard data type constants:

Is Text
Is Longint
Is Real
Is Boolean
Is Date
Is Time
Is Pointer
Text array
Longint Array
Real Array
Boolean array
Date array
Pointer array

Fnd_Dict_GetArray

Fnd_Dict_GetArray (dict ID; key name; ->values)

Parameter	Type	Description
dict ID	Number	A dictionary ID
key name	Text	A key name
values	Pointer	Text array to receive values

This method returns an array of values from a dictionary given a key.

Fnd_Dict_GetBoolean

Fnd_Dict_GetBoolean (dict ID; key name) ➔ Boolean

Parameter	Type	Description
dict ID	Number	A dictionary ID
key name	Text	A key name
Function Result	Boolean	The value

This function returns a Boolean value with the specified key from a dictionary.

Fnd_Dict_GetDate

Fnd_Dict_GetDate (dict ID; key name) ➔ Date

Parameter	Type	Description
dict ID	Number	A dictionary ID
key name	Text	A key name
Function Result	Date	The value

This function returns a 4D Date value with the specified key from a dictionary.

Fnd_Dict_GetLongint

Fnd_Dict_GetLongint (dict ID; key name) ➔ Longint

Parameter	Type	Description
dict ID	Number	A dictionary ID
key name	Text	A key name
Function Result	Longint	The value

This function returns a long integer value with the specified key from a dictionary.

Fnd_Dict_GetPointer

Fnd_Dict_GetPointer (dict ID; key name) ➔ Pointer

Parameter	Type	Description
dict ID	Number	A dictionary ID
key name	Text	A key name
Function Result	Pointer	The value

This function returns the pointer with the specified key from a dictionary.

Fnd_Dict_GetReal

Fnd_Dict_GetReal (dict ID; key name) ➔ Real

Parameter	Type	Description
dict ID	Number	A dictionary ID
key name	Text	A key name
Function Result	Real	The value

This function returns a floating point numeric value with the specified key from a dictionary.

Fnd_Dict_GetText

Fnd_Dict_GetText (dict ID; key name) ➔ Text

Parameter	Type	Description
dict ID	Number	A dictionary ID
key name	Text	A key name
Function Result	Text	The value

This function returns a text value with the specified key from a dictionary.

Fnd_Dict_GetTime

Fnd_Dict_GetTime (dict ID; key name) ➔ Time

Parameter	Type	Description
dict ID	Number	A dictionary ID
key name	Text	A key name
Function Result	Time	The value

This function returns a 4D time value with the specified key from a dictionary.

Fnd_Dict_HasKey

Fnd_Dict_HasKey (dict ID; key name) ➔ Boolean

Parameter	Type	Description
dict ID	Number	A dictionary ID
key name	Text	A key name
Function Result	Boolean	True if the key exists

This function tests whether or not a specific key exists in a dictionary. It will return **True** if the key exists.

Fnd_Dict_ID

Fnd_Dict_ID (dict name) ➔ Number

Parameter	Type	Description
dict name	Text	A dictionary name
Function Result	Number	The dictionary's reference number

This function returns the reference number with the first dictionary found with the specified name.

Keep in mind that it is possible to create multiple dictionaries with the same name. If more than one dictionary exists with the specified name, the first match will be returned.

Fnd_Dict_Info

Fnd_Dict_Info (info requested) ➔ Text

Parameter	Type	Description
info requested	Text	Info desired
Function Result	Text	Response

This function returns the requested information about the Dict component.

```
$version_t:=Fnd_Dict_Info ("version")
```

The *Fnd_Date_Info* method will respond to these requests:

Request	Response	Example
name	The component's full name	Foundation Dict
version	The component's version number	4.0.5 beta 4

This routine can also be called using the *Fnd_Gen_ComponentInfo* method without first testing to see if the component is installed:

```
$version_t:=Fnd_Gen_ComponentInfo ("Fnd_Dict";"version")
```

See the *Fnd_Gen_ComponentInfo* method for more information.

Fnd_Dict_IsValid

Fnd_Dict_IsValid (dict ID) ➔ Boolean

Parameter	Type	Description
dict ID	Number	A dictionary ID
Function Result	Boolean	True if the dictionary is valid

This function tests whether or not the specified dictionary reference is valid. It will return **True** if the reference is valid and can be used.

Fnd_Dict_ItemCount

Fnd_Dict_ItemCount (dict ID) ➔ Number

Parameter	Type	Description
dict ID	Number	A dictionary ID
Function Result	Number	The number of items

Returns the number of items (or keys) in a dictionary.

Fnd_Dict_Keys

Fnd_Dict_Keys (dict ID; ->keys)

Parameter	Type	Description
dict ID	Number	A dictionary ID
keys	Pointer	Pointer to a text array

This command fills an array with the keys in the specified dictionary.

Fnd_Dict_LoadFromBlob

Fnd_Dict_LoadFromBlob (->dict BLOB) ➔ Longint

Parameter	Type	Description
dict BLOB	Pointer	A dictionary in BLOB form
Function Result	Number	The new dictionary's reference number

This function loads a dictionary from a 4D BLOB created by a previous call to *Fnd_Dict_SaveToBlob*. It returns the reference number to the new dictionary.

Fnd_Dict_LoadFromFile

Fnd_Dict_LoadFromFile (path) ➔ Longint

Parameter	Type	Description
path	Text	A path to a dictionary file
Function Result	Number	The new dictionary's reference number

This function loads a dictionary from the file specified by the path parameter. It returns the reference number to the new dictionary.

Fnd_Dict_Name

Fnd_Dict_Name (dict ID) ➔ Text

Parameter	Type	Description
dict ID	Number	A dictionary ID
Function Result	Text	The dictionary's name

This function returns the name of the dictionary. Each dictionary has a name, whether assigned when calling the Fnd_Dict_New method, or if automatically assigned by the component.

Fnd_Dict_New

Fnd_Dict_New ({name}) ➔ Longint

Parameter	Type	Description
name	Text	A name (optional)
Function Result	Number	The new dictionary's reference number

This funciton creates a new dictionary and returns its reference number.

If no name is passed, the dictionary will be named "Dictionary *n*" where *n* is the dictionary’s reference number. Note that this number may not reflect the dictionary’s actual reference number if the dictionary is saved and later reloaded.

Fnd_Dict_Release

Fnd_Dict_Release (dict ID)

Parameter	Type	Description
dict ID	Number	A dictionary ID

This command decrements the retain count of a dictionary. Once the retain count reaches 0, the dictionary is automatically cleared from memory.

Fnd_Dict_Remove

Fnd_Dict_Remove (dict ID; key name)

Parameter	Type	Description
dict ID	Number	A dictionary ID
key name	Text	A key name

Use this command to remove a key, and its associated data, from a dictionary.

Fnd_Dict_Retain

Fnd_Dict_Retain (dict ID)

Parameter	Type	Description
dict ID	Number	A dictionary ID

This command increments the retain count for a dictionary

Fnd_Dict_RetainCount

Fnd_Dict_RetainCount (dict ID) ➔ Longint

Parameter	Type	Description
dict ID	Number	A dictionary ID
Function Result	Number	The retain count

This function returns the retain count for the specified dictionary.

If the dictionary reference is not valid, 0 is returned.

Fnd_Dict_SaveToBlob

Fnd_Dict_SaveToBlob (dict ID; ->BLOB)

Parameter	Type	Description
dict ID	Number	A dictionary ID
BLOB	BLOB	A BLOB variable or field

Use this command to save a dictionary as a 4D BLOB.

Because 4D's XML creation routines were introduced in 4D 2004, this command will work only with 4D 2004 or later.

Fnd_Dict_SaveToFile

Fnd_Dict_SaveToFile (dict ID; path)

Parameter	Type	Description
dict ID	Number	A dictionary ID
path	Text	A file path

Use this command to save a dictionary as a file.

If a file already exists at the path's location, it will be replaced with the updated dictionary file.

Because 4D's XML creation routines were introduced in 4D 2004, this command will work only with 4D 2004 or later.

Fnd_Dict_SetArray

Fnd_Dict_SetArray (dict ID; key name; ->values)

Parameter	Type	Description
dict ID	Number	A dictionary ID
key name	Text	A key name
values	Array	An array of values

Use this command to store an array of values in a dictionary.

The array can be an array any of these types:

Text or String
Longint or Integer
Real
Boolean
Date
Pointer

Fnd_Dict_SetBoolean

Fnd_Dict_SetBoolean (dict ID; key name; value)

Parameter	Type	Description
dict ID	Number	A dictionary ID
key name	Text	A key name
value	Boolean	The value to store

This command adds the specified key and Boolean value to the dictionary.

If the key already exists, the old value is replaced with this new value.

Fnd_Dict_SetDate

Fnd_Dict_SetDate (dict ID; key name; value)

Parameter	Type	Description
dict ID	Number	A dictionary ID
key name	Text	A key name
value	Date	The value to store

This command adds the specified key and 4D date value to the dictionary.

If the key already exists, the old value is replaced with this new value.

Fnd_Dict_SetLongint

Fnd_Dict_SetLongint (dict ID; key name; value)

Parameter	Type	Description
dict ID	Number	A dictionary ID
key name	Text	A key name
value	Boolean	The value to store

This command adds the specified key and long integer (or integer) value to the dictionary.

If the key already exists, the old value is replaced with this new value.

Fnd_Dict_SetPointer

Fnd_Dict_SetPointer (dict ID; key name; value)

Parameter	Type	Description
dict ID	Number	A dictionary ID
key name	Text	A key name
value	Pointer	The pointer to store

This command adds the specified key and pointer to the dictionary.

If the key already exists, the old value is replaced with this new value.

Fnd_Dict_SetReal

Fnd_Dict_SetReal (dict ID; key name; value)

Parameter	Type	Description
dict ID	Number	A dictionary ID
key name	Text	A key name
value	Real	The data to store

This command adds the specified key and floating numeric value to the dictionary.

If the key already exists, the old value is replaced with this new value.

Fnd_Dict_SetText

Fnd_Dict_SetText (dict ID; key name; value)

Parameter	Type	Description
dict ID	Number	A dictionary ID
key name	Text	A key name
value	Text	The data to store

This command adds the specified key and text (or string) value to the dictionary.

If the key already exists, the old value is replaced with this new value.

Fnd_Dict_SetTime

Fnd_Dict_SetTime (dict ID; key name; value)

Parameter	Type	Description
dict ID	Number	A dictionary ID
key name	Text	A key name
value	Time	The data to store

This command adds the specified key and 4D time value to the dictionary.

If the key already exists, the old value is replaced with this new value.

Fnd_Dict_Values

Fnd_Dict_Values (dict ID; ->values)

Parameter	Type	Description
dict ID	Number	A dictionary ID
values	Pointer	Pointer to a text array

This function fills the text array with the values stored in the dictionary.

All of the values will be returned as text, regardless of their actual data type.

Dialogs Component

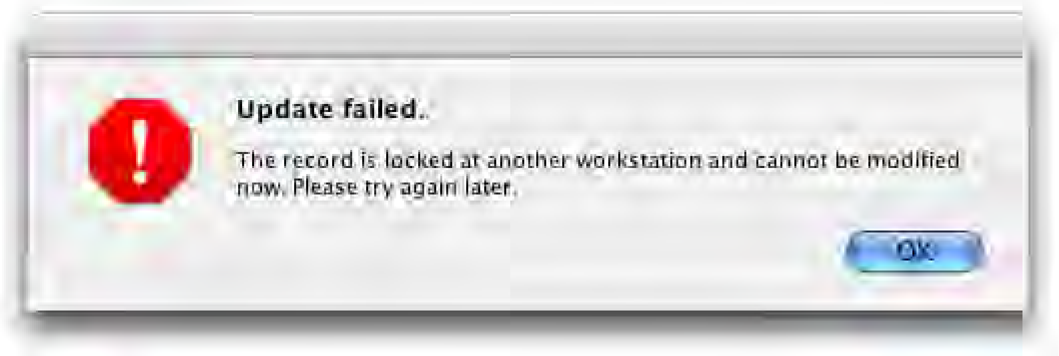
(Fnd_Dlg)

The Dialogs component enables you to display a variety of dialogs and messages in order to retrieve data from the user, to ask the user a question, or to notify the user about something.

Dialogs

The routines in this component can be used to display alert, confirm, and request dialog boxes.

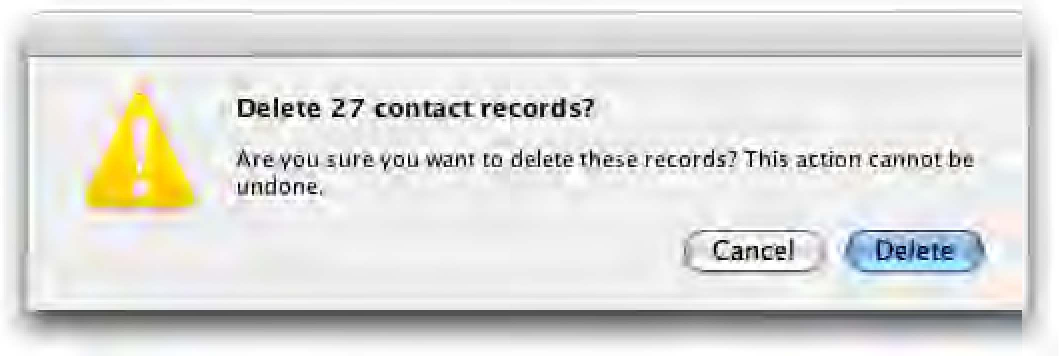
Foundation's Alert dialog box:



Update failed.

The record is locked at another workstation and cannot be modified now. Please try again later.

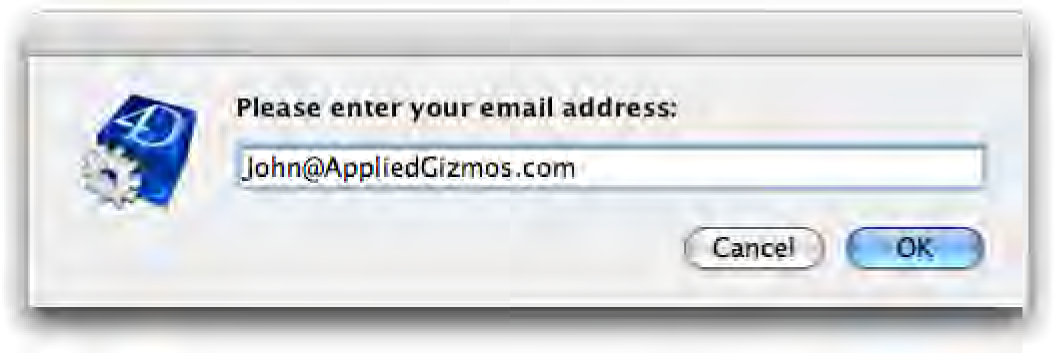
Foundation's Confirm dialog box:



Delete 27 contact records?

Are you sure you want to delete these records? This action cannot be undone.

Foundation's Request dialog box:



Please enter your email address:

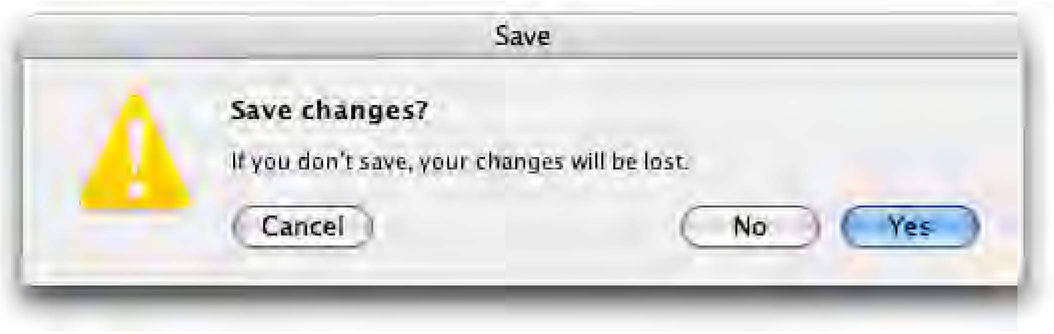
John@AppliedGizmos.com

The easiest way to use this component is simply to replace 4D's **ALERT**, **CONFIRM**, and **Request** commands with this component's equivalents: *Fnd_Dlg_Alert*; *Fnd_Dlg_Confirm*; and *Fnd_Dlg_Request*.

If you want additional control you can use the component's other routines to configure the dialogs. You can define the text for the buttons (*Fnd_Dlg_SetButtons*), whether or not the upcoming dialog can be canceled (*Fnd_Dlg_SetCancelable*), which icon to display (*Fnd_Dlg_SetIcon*), the position of the dialog (*Fnd_Wnd_Position*), the default value for a request dialog (*Fnd_Dlg_SetRequest*), the message text (*Fnd_Dlg_SetText*), the window title (*Fnd_Wnd_Title*) and position (*Fnd_Wnd_Position*). After setting up the dialog, you call the *Fnd_Dlg_Display* routine to display it to the user.

For example:

```
Fnd_Wnd_Title ("Save")
Fnd_Dlg_SetText ("Save changes?";"If you don’t save, your changes will be lost.")
Fnd_Dlg_SetButtons ("Yes";"No";"Cancel")
Fnd_Dlg_SetIcon (Fnd_Dlg_WarnIcon )
Fnd_Wnd_Position (Fnd_Wnd_CenterOnWindow )
Fnd_Dlg_Display
```



Save

Save changes?

If you don't save, your changes will be lost.

If you have requested information to be entered by the user using the *Fnd_Dlg_SetRequest* routine, call the *Fnd_Dlg_GetRequest* routine to find out what the user has entered.

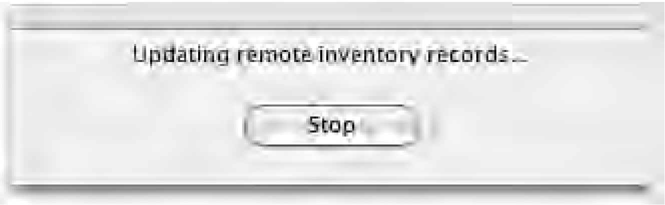
Messages

The routines in this component allow you to open a message window (*Fnd_Dlg_MessageOpen*), update it (*Fnd_Dlg_MessageUpdate*), check if it has been canceled by the user (*Fnd_Dlg_MessageCanceled*), define the parameters for the progress indicator (*Fnd_Dlg_SetProgress*), and close it (*Fnd_Dlg_MessageClose*).

Alternatively, you can use the *Fnd_Dlg_Message* routine which replaces 4D's **MESSAGE** command.

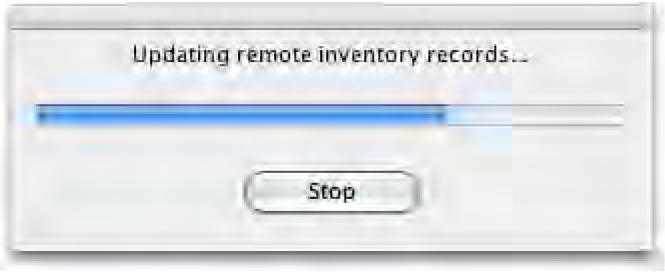
Much like the dialog routines, you can set the following aspects of the message window: the text for the buttons (*Fnd_Dlg_SetButtons*), whether or not the upcoming dialog can be canceled (*Fnd_Dlg_SetCancelable*), the message text (*Fnd_Dlg_SetText*), and the window title (*Fnd_Wnd_Title*).

Foundation's standard Message window:



Updating remote inventory records...

Foundation's standard Message window with progress indicator:



Updating remote inventory records...

Language Reference

Here is the list of routines in Foundation's Dialog component:

Fnd_Dlg_Alert
Fnd_Dlg_Confirm
Fnd_Dlg_CustomIcon
Fnd_Dlg_Display
Fnd_Dlg_GetRequest
Fnd_Dlg_Info
Fnd_Dlg_Message
Fnd_Dlg_MessageCanceled
Fnd_Dlg_MessageClose
Fnd_Dlg_MessageOpen
Fnd_Dlg_MessageUpdate
Fnd_Dlg_Request
Fnd_Dlg_SetButtons
Fnd_Dlg_SetCancelable
Fnd_Dlg_SetIcon
Fnd_Dlg_SetProgress
Fnd_Dlg_SetRequest
Fnd_Dlg_SetText

Fnd_Dlg_Alert

Fnd_Dlg_Alert (message{; OK button})

Parameter	Type	Description
message	Text	Message to display
OK button	Text	OK button text or "*" for localized "OK" (optional)

The *Fnd_Dlg_Alert* routine replaces 4D's **ALERT** command. You can either pass a value to the OK button parameter, which will be used in place of the **OK** button. Otherwise, "OK" will be used or the localized version of "OK" will be used.

```
Fnd_Dlg_Alert ("The selected record cannot be deleted.")
```

Fnd_Dlg_Confirm

Fnd_Dlg_Confirm (message{; OK button{; Cancel button}})

Parameter	Type	Description
message	Text	Message to display
OK button	Text	OK button text "or "*" for localized "OK" (optional)
Cancel button	Text	Cancel button text or "*" for localized "Cancel" (optional)

The *Fnd_Dlg_Confirm* routine replaces 4D's **CONFIRM** command. You can pass titles for the **OK** and **Cancel** buttons. If you leave the OK button parameter empty or pass "*", "OK" (or its localized equivalent) will be used. If you specify "*" for the Cancel button parameter, the localized version of "Cancel" will be used.

```
Fnd_Dlg_Confirm ("Are you sure you want to delete the record?";"Delete")
```

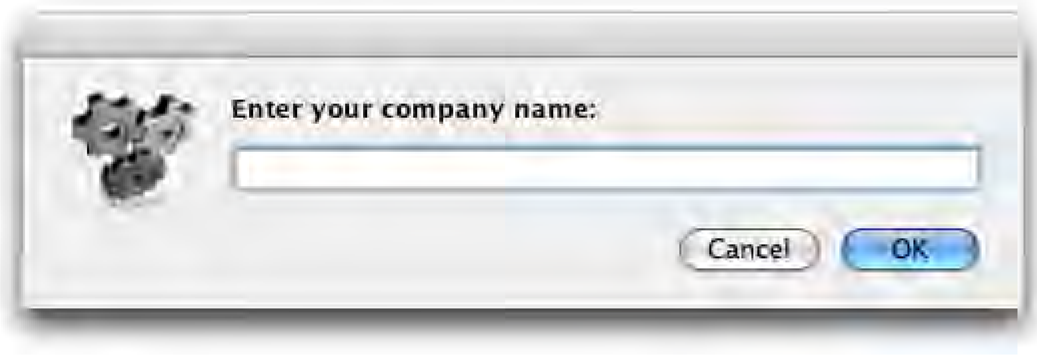
Fnd_Dlg_CustomIcon

Fnd_Dlg_CustomIcon (picture library name) ➔ Text

Parameter	Type	Description
message	Text	Message to display
OK button	Text	OK button text "or "*" for localized "OK" (optional)
Cancel button	Text	Cancel button text or "*" for localized "Cancel" (optional)

Use this routine to set and get the name of an image from the 4D Picture library to display as the icon when displaying an alert, confirm, or request dialog.

```
Fnd_Dlg_CustomIcon ("ApplicationIcon")
Fnd_Dlg_Request ("Enter your company name:")
```



Enter your company name:

An image with the same name must be available in the 4D Picture Library. It should be exactly 64 pixels wide and 64 pixels high.

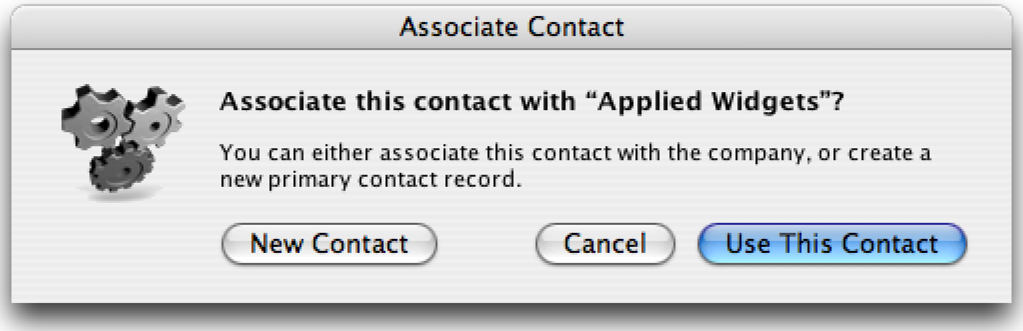
Fnd_Dlg_Display

Fnd_Dlg_Display

Parameter	Type	Description
No parameters required.		

The *Fnd_Dlg_Display* routine displays the dialog that you have defined by using the other routines in this component and in the Windows component (*Fnd_Dlg_SetButtons*, *Fnd_Dlg_SetCancelable*, *Fnd_Dlg_SetIcon*, *Fnd_Dlg_SetProgress*, *Fnd_Dlg_SetRequest*, *Fnd_Dlg_SetText*, *Fnd_Wnd_Position*, and *Fnd_Wnd_Title*).

```
Fnd_Wnd_Title ("Associate Contact")
Fnd_Dlg_SetText ("Associate this contact with “"+[Company]Name+"”?";"You can either
associate this contact with the company, or create a new primary contact record.")
Fnd_Dlg_SetButtons ("Use This Contact";"Cancel";"New Contact")
Fnd_Dlg_CustomIcon ("ApplicationIcon")
Fnd_Wnd_Position (Fnd_Wnd_CenterOnWindow )
Fnd_Dlg_Display
```



Associate Contact

Associate this contact with “Applied Widgets”?

You can either associate this contact with the company, or create a new primary contact record.

Fnd_Dlg_GetRequest

Fnd_Dlg_GetRequest ➔ Text

Parameter	Type	Description
No parameters required.		
Function result	Text	Value entered by the user

The *Fnd_Dlg_GetRequest* routine returns the value entered by the user after displaying a Request dialog with the *Fnd_Dlg_Display* routine in conjunction with the *Fnd_Dlg_SetRequest* routine.

See the *Fnd_Dlg_SetRequest* routine for an example. This routine is unnecessary if you call the *Fnd_Dlg_Request* function.

Fnd_Dlg_Info

Fnd_Dlg_Info (info requested) ➔ Text

Parameter	Type	Description
info requested	Text	Info desired
Function Result	Text	Response

Returns the requested information about the component.

```
$version_t:=Fnd_Dlg_Info ("version")
```

The *Fnd_Dlg_Info* method will respond to these requests:

Request	Response	Example
name	The component's full name	Foundation Dialogs
version	The component's version number	4.0.1

This routine can also be called using the *Fnd_Gen_ComponentInfo* method without first testing to see if the component is installed:

```
$version_t:=Fnd_Gen_ComponentInfo ("Fnd_Dlg";"version")
```

See the *Fnd_Gen_ComponentInfo* method for more information.

Fnd_Dlg_Message

Fnd_Dlg_Message (message text)

Parameter	Type	Description
message text	Text	Message text to display

The *Fnd_Dlg_Message* routine displays a message window. This routine replaces 4D's **MESSAGE** command.

```
Fnd_Dlg_Message ("Connecting to the data server...")
ODBC_Connect
Fnd_Dlg_Message ("Downloading the structure...")
ODBC_GetMetaData
ODBC_CloseConnection
Fnd_Dlg_MessageClose
```

You can update the message while it is displayed by calling this method again.

To close the message window that appears after calling this routine, call this routine again and pass an empty string ("") as the message text parameter, or call the *Fnd_Dlg_MessageClose* command.

For more control over the message dialog, use the *Fnd_Dlg_MessageOpen* routine instead.

Fnd_Dlg_MessageCanceled

Fnd_Dlg_MessageCanceled ➔ Boolean

Parameter	Type	Description
No parameters required.		
Function result	Boolean	Returns True if the user clicked Cancel

The *Fnd_Dlg_MessageCanceled* routine returns **True** if the user has clicked the **Cancel** button in the message dialog that is currently being displayed.

Fnd_Dlg_MessageClose

Fnd_Dlg_MessageClose

Parameter	Type	Description
No parameters required.		

The *Fnd_Dlg_MessageClose* routine closes the message window that was previously opened with *Fnd_Dlg_Message* or *Fnd_Dlg_MessageOpen*.

See the *Fnd_Dlg_Message* routine for an example.

Fnd_Dlg_MessageOpen

Fnd_Dlg_MessageOpen

Parameter	Type	Description
No parameters required.		

The *Fnd_Dlg_MessageOpen* routine opens a message window. You can also specify the button text, the text to display, the window title, and whether the user can cancel the message window by calling the following routines: *Fnd_Dlg_SetButtons*, *Fnd_Dlg_SetText*, *Fnd_Wnd_Title*, and *Fnd_Dlg_SetCancelable*.

Fnd_Dlg_MessageUpdate

Fnd_Dlg_MessageUpdate (number)

Parameter	Type	Description
number	Longint	Next number in the sequence

The *Fnd_Dlg_MessageUpdate* routine updates the progress indicator if used by calling the *Fnd_Dlg_SetProgress* routine.

The number parameter updates the progress indicator.

Fnd_Dlg_Request

Fnd_Dlg_Request (message{; default{; OK button{; Cancel button}}}) ➔ Text

Parameter	Type	Description
message	Text	Message to display
default	Text	Default response text (optional)
OK button	Text	OK button text "or "*" for localized "OK" (optional)
Cancel button	Text	Cancel button text or "*" for localized "Cancel" (optional)
Function result	Text	User's response

The *Fnd_Dlg_Request* routine is designed to replace 4D's **Request** function. It displays a **Request** dialog that can be modified by setting the default response text, the text of the **OK** button, and the text of the **Cancel** button. This routine returns the user's response and sets the OK variable.

```
$reply_t:=Fnd_Dlg_Request ("Enter the customer's last name:")
```

If you pass "*" to the OK button or Cancel button parameter, the localized text for "OK" or "Cancel" will be used.

If you have used the *Fnd_Dlg_Request* routine, you do not need to call *Fnd_Dlg_GetRequest* because this routine returns the data entered by the user.

Fnd_Dlg_SetButtons

Fnd_Dlg_SetButtons ({button 1{; button 2{; button 3}}})

Parameter	Type	Description
button 1	Text	OK button text "or "*" for localized "OK" (optional)
button 2	Text	Cancel button text or "*" for localized "Cancel" (optional)
button 3	Text	Third button text (optional)

The *Fnd_Dlg_SetButtons* routine sets the button texts for the upcoming alerts that will be opened by using the routines in this component. The third button is the "Don't Save" button.

Pass values for the **OK** and **Cancel** buttons in the button 1 and button 2 parameters. If you pass "*", the localized version for these buttons will be used. The button 3 parameter sets the text of the third button.

If you do not pass a value to button 2 or button 3, these buttons will not be displayed. So, you can create a dialog with one, two, or three buttons.

See the *Fnd_Dlg_Display* routine for a code example.

Fnd_Dlg_SetCancelable

Fnd_Dlg_SetCancelable (cancelable)

Parameter	Type	Description
cancelable	Boolean	True if the dialog can be canceled

With the *Fnd_Dlg_SetCancelable* routine, you can specify whether or not the upcoming dialog can be canceled. This routine can be used in conjunction with the *Fnd_Dlg_MessageOpen* routine.

Fnd_Dlg_SetIcon

Fnd_Dlg_SetIcon ({icon number})

Parameter	Type	Description
icon number	Longint	Icon number (optional)

Use the *Fnd_Dlg_SetIcon* routine to set the icon for an upcoming alert displayed with the *Fnd_Dlg_Display* routine. Pass one of the following values:

Value	4D Constant	Description	Icon
0		Default icon (Note)	
1	Fnd_Dlg_NoteIcon	Note Icon	
2	Fnd_Dlg_WarnIcon	Warn icon	
3	Fnd_Dlg_StopIcon	Stop Icon	

See the *Fnd_Dlg_Display* routine for a code example.

Fnd_Dlg_SetPosition

This command is obsolete and has been removed. Use the *Fnd_Wnd_Position* command instead.

Fnd_Dlg_SetProgress

Fnd_Dlg_SetProgress (to or from{; to})

Parameter	Type	Description
to or from	Longint	Starting number or ending number
to	Longint	Ending number (optional)

Call *Fnd_Dlg_SetProgress* routine to set the from and to numbers in a message window. If only one number is passed to the to or from parameter, the from is assumed to be 1. If to is greater than from, the progress indicator will go backwards (from right to left).

This routine can be used in conjunction with the *Fnd_Dlg_MessageOpen* routine.

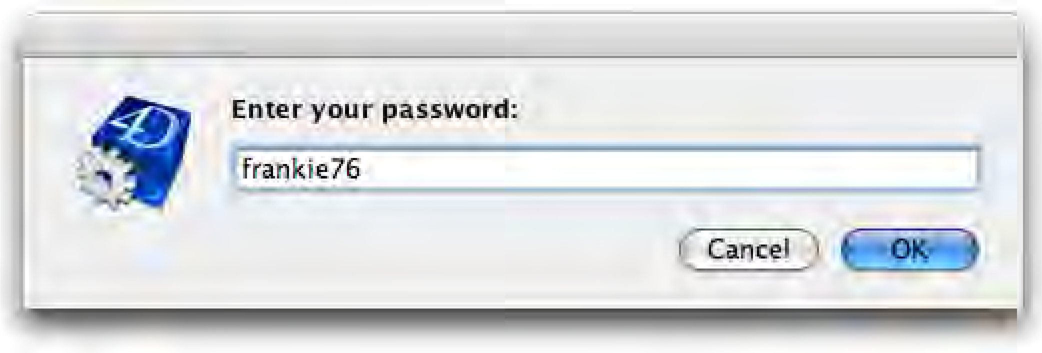
Fnd_Dlg_SetRequest

Fnd_Dlg_SetRequest ({default})

Parameter	Type	Description
default	Text	Default value (optional)

The *Fnd_Dlg_SetRequest* routine sets up the dialog to allow the user to enter a value when calling the *Fnd_Dlg_Display* routine. Call this method with no parameters to cause the request area to be displayed in the dialog:

```
Fnd_Dlg_SetText ("Enter your password:")
Fnd_Dlg_SetRequest
Fnd_Dlg_Display
If(OK=1)
  $password_t:=Fnd_Dlg_GetRequest
End if
```



Enter your password:

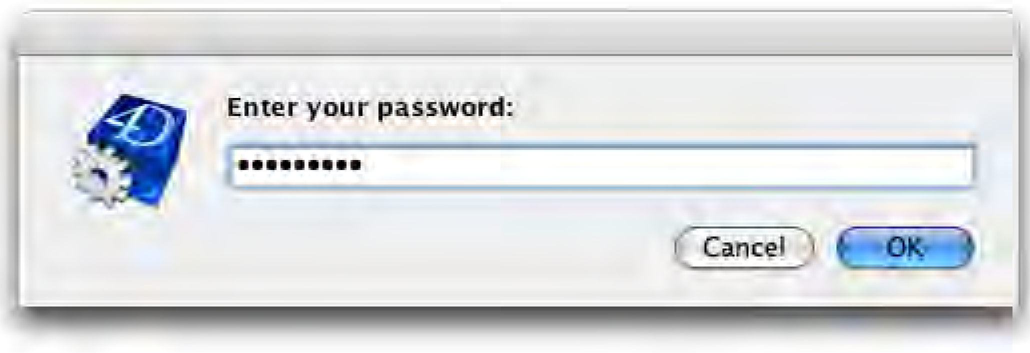
frankie76

Or pass a text value to display in the request field:

```
Fnd_Dlg_SetText ("Enter the preferred date:")
Fnd_Dlg_SetRequest (String(Current date))
Fnd_Dlg_Display
If(OK=1)
  $date_d:=Fnd_Dlg_GetRequest
End if
```

If you pass “*” to the default parameter, the entry area displays bullets rather than text. This feature is useful for entering passwords.

```
Fnd_Dlg_SetText ("Enter your password:")
Fnd_Dlg_SetRequest ("*")
Fnd_Dlg_Display
If(OK=1)
  $password_t:=Fnd_Dlg_GetRequest
End if
```



Enter your password:

••••••••••

Fnd_Dlg_SetText

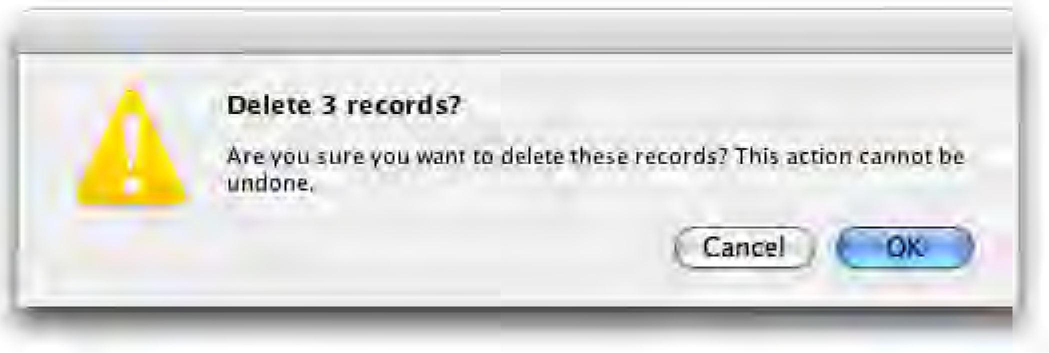
Fnd_Dlg_SetText (text 1{; text 2})

Parameter	Type	Description
text 1	Text	Primary alert text
text 2	Text	Smaller alert text (optional)

The *Fnd_Dlg_SetText* routine sets the message texts for the upcoming dialogs opened with the *Fnd_Dlg_Display* and *Fnd_Dlg_MessageOpen* routines.

See the *Fnd_Dlg_Display* routine for a code example.

You can define a smaller alert text as on Macintosh OS X in the text 2 parameter as shown in the dialog below:



Delete 3 records?

Are you sure you want to delete these records? This action cannot be undone.

Fnd_Dlg_SetWindowTitle

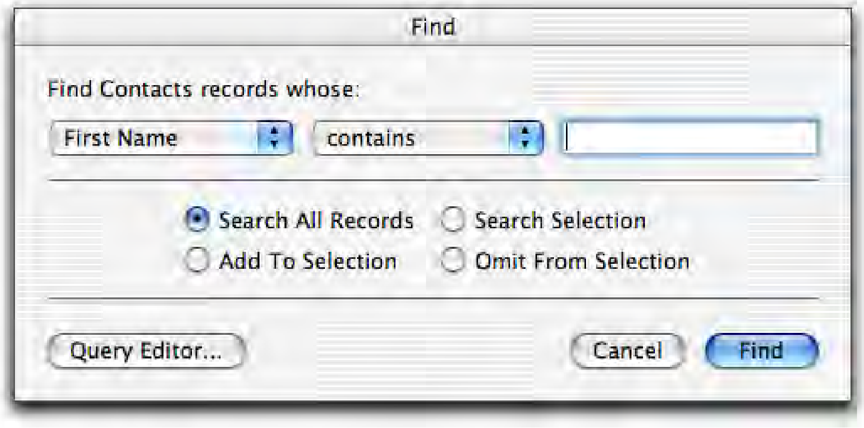
This command is obsolete and has been removed. Use the *Fnd_Wnd_Title* command instead.

Find Component

Fnd_Find

Foundation's Find component enables you to execute queries on the data in your 4D application.

Foundation's Find dialog box:



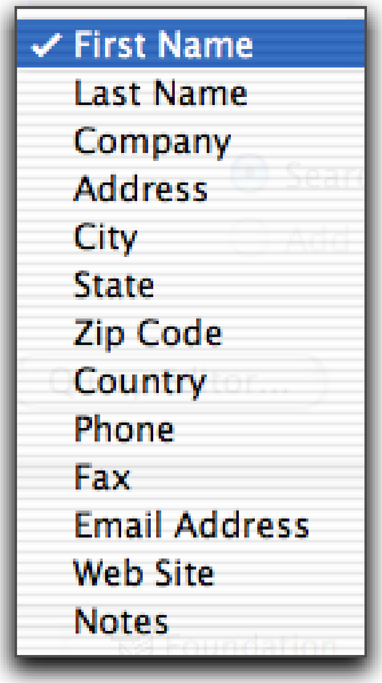
Find

Find Contacts records whose:

First Name | contains |

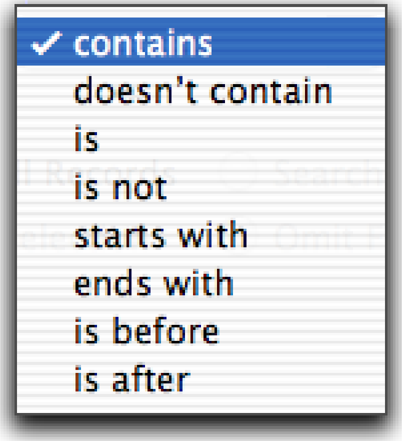
● Search All Records ○ Search Selection
○ Add To Selection ○ Omit From Selection

The visible fields in your database are displayed in this pop-up menu:



- ✓ First Name
- Last Name
- Company
- Address
- City
- State
- Zip Code
- Country
- Phone
- Fax
- Email Address
- Web Site
- Notes

The list of operators changes depending on the type of the field selected:



- ✓ contains
- doesn't contain
- is
- is not
- starts with
- ends with
- is before
- is after

The Search All Records, Add To Selection, Search Selection, and Omit From Selection radio buttons allow you to define the query even more. By selecting Query Editor..., the standard 4th Dimension Query Editor is displayed.

The Find dialog can be displayed simply by calling the *Fnd_Find_Display* method. Foundation will automatically determine the table to use (by calling the *Fnd_Gen_CurrentTable* method) and display all of the visible, searchable fields in the pop-up menu.

You can have more control over the dialog by calling the other component methods before calling Fnd_Find_Display. For example, the following code will display all of the visible fields from the current table, then add a separator line and a field from a related table to the pop-up menu:

```
Fnd_Find_AddTable (Fnd_Gen_CurrentTable)
Fnd_Find_AddSeparator
Fnd_Find_AddField (->[Companies]Company Name)
```

Here is the list of routines in Foundation's Find component:

- Fnd_Find_AddCustom
- Fnd_Find_AddField
- Fnd_Find_AddMultiField
- Fnd_Find_AddSeparator
- Fnd_Find_AddSubfield
- Fnd_Find_AddTable
- Fnd_Find_Display
- Fnd_Fnd_Info
- Fnd_Find_SetEditorButton

Language Reference

Here are the routines in Foundation's Find component:

Fnd_Find_AddCustom

Fnd_Find_AddCustom (label; method name{; position})

Parameter	Type	Description
label	Text	Label to use
method name	Text	Method to call
position	Longint	Position in which to add the item (optional)

The *Fnd_Find_AddCustom* routine lets the developer add a custom field to the search dialog.

If the user selects the custom field label from the pop-up menu, Foundation will not attempt to search the database. Instead, the specified method will be called and will be passed these parameters:

Parameter	Type	Description
label	Text	The label passed to the Fnd_Find_AddCustom method
operator	Text	The operator code selected in the second pop-up menu
search	Text	The text entered into the Find dialog
button	Number	The selected radio button

Search Operator Options:

SW = Starts With
EW = Ends With
C = Contains
DC = Doesn't Contain
0

> = Greater Than
< = Less Than
# = Doesn't Equal
N> = Number greater than
N< = Number less than
N>= = Number is Greater or equal to
N<= = Number is less than or equal to
N= = Number equals
N# = Number is not equal to
T = True
F = False

Radio button options:

1 = Search All
2 = Search Selection
3 = Add to Selection
4 = Omit from Selection

The method specified in this call must contain the following compiler declarations:

```
C_TEXT($1;$2;$3)
C_LONGINT($4)
```

Fnd_Find_AddField

Fnd_Find_AddField (->field{; position})

Parameter	Type	Description
field	Pointer	Pointer to the field to add
position	Longint	Position in which to add the item (optional)

The *Fnd_Find_AddField* routine lets the developer add a field to the search dialog. If position is not specified, the field is added to the end of the list.

The field can be from the current table or a related table.

Fnd_Find_AddMultiField

Fnd_Find_AddMultiField (->array of field ptrs; label{; position})

Parameter	Type	Description
array of field ptrs	Pointer	Pointer to an array of pointers
label	Text	Label for the popup item
position	Longint	Position in which to add the item (optional)

The *Fnd_Find_AddMultiField* routine lets the developer add one item for searching multiple fields to the search dialog. If position is not specified, this item is added to the end of the list.

The array of field ptrs parameter must be an array of pointers, each of which points to a text or alpha field.

After calling *Fnd_Find_Display*, you should reset the array to zero elements to release the memory used by the array.

This routine is ideal for allowing users to search all of the names fields without having to specify a specific field each time. For example:

```
ARRAY POINTER (aFields;3)
aFields{1}:=->[Employees]First Name
aFields{2}:=->[Employees]Middle Name
aFields{3}:=->[Employees]Last Name
Fnd_Find_AddMultiField (->aFields;"Name")
Fnd_Find_Display
ARRAY POINTER (aFields;0)
```

Fnd_Find_AddSeparator

Fnd_Find_AddSeparator ({position})

Parameter	Type	Description
position	Longint	Position of the separator (optional)

The *Fnd_Find_AddSeparator* routine adds a separator line at position to the list of searchable fields to the Find dialog.

Fnd_Find_AddSubfield

Fnd_Find_AddSubfield (->subfield; field type{; position})

Parameter	Type	Description
subfield	Pointer	A pointer to the subfield to add
field type	Longint	Type of the subfield
position	Longint	Position in which to add the item (optional)

The *Fnd_Find_AddSubfield* routine lets the developer add a subfield to the Find dialog. The field type must be specified because we cannot procedurally determine its type like we can for fields. If a position is not specified, the field is added to the end of the list.

The field type parameter can be one of 4D's Field Type constants:

4D Constant	Type	Value
Is Alpha Field	Longint	0
Is Boolean	Longint	6
Is Date	Longint	4
Is Integer	Longint	8
Is LongInt	Longint	9
Is Real	Longint	1

Is Text	Longint	2
Is Time	Longint	11

Fnd_Find_AddTable

Fnd_Find_AddTable (->table{; position})

Parameter	Type	Description
table	Pointer	A pointer to the table to add
position	Longint	Position in which to add the item (optional)

The *Fnd_Find_AddTable* routine loads the visible, searchable fields for the specified table into the field pop-up menu.

```
Fnd_Find_AddTable (->[Contacts])
```

This command relies on the table returned by *Fnd_Gen_CurrentTable*. If you want to use *Fnd_Find_AddTable* from your own process, you must set the current table with *Fnd_Gen_CurrentTable*. Remember to reset the current table when you are finished with your process.

```
$currentTable_ptr:=Fnd_Gen_CurrentTable   ` get the current setting
Fnd_Gen_CurrentTable($yourTablePointer_ptr)   ` set it to your working table
Fnd_Find_AddTable ($yourTablePointer_ptr)
Fnd_Find_AddField ($yourFieldPointer_ptr)
Fnd_Find_Display   ` show the find dialog
Fnd_Gen_CurrentTable($currentTable_ptr)   ` reset it back to the table it was on
```

Fnd_Find_Display

Fnd_Find_Display

Parameter	Type	Description
No parameters required.		

The *Fnd_Find_Display* routine displays the Find dialog. If the user clicks OK, the component performs the query and the current selection is modified.

Fnd_Find_Info

Fnd_Find_Info (info requested) ➔ Text

Parameter	Type	Description
info requested	Text	Info desired
Function Result	Text	Response

This function returns the requested information about the component.

`$version_t:=`*`Fnd_Find_Info`*` (“version”)`

The *Fnd_Find_Info* method will respond to these requests:

Request	Response	Example
name	The component’s full name	Foundation Find
version	The component’s version number	4.0.4 beta 2

This routine can also be called using the *Fnd_Gen_ComponentInfo* method without first testing to see if the component is installed:

`$version_t:=`*`Fnd_Gen_ComponentInfo`*` ("Fnd_Find";"version")`

See the *Fnd_Gen_ComponentInfo* method for more information.

Fnd_Find_SetEditorButton

Fnd_Find_SetEditorButton (label{; method})

Parameter	Type	Description
label	Text	New label for the Query Editor button
method	Text	Method to execute (optional)

The Fnd_Find_SetEditorButton routine allows the developer to hide or modify the Query Editor button in the Find dialog. Pass an empty string to label to hide the button.

`Fnd_Find_SetEditorButton`` ("")`

If method is specified, it will be executed instead of displaying 4D's Query Editor.

`Fnd_Find_SetEditorButton`` ("Query with SQL";"MySQLSelectMethod")`

General Component

(Fnd_Gen)

The "Fnd_Gen" component is the only required Foundation component. That is because it contains the core functions that are used by all of the other Foundation components. This component requires the Foundation Extras plugin.

The components here were designed specifically for use by the other Foundation components. If there is something here you can use in your routines, feel free.

Language Reference

Here is the list of routines in Foundation's General component:

Fnd_Gen_BugAlert
Fnd_Gen_ButtonText
Fnd_Gen_CancelQuit
Fnd_Gen_CenterWindow
Fnd_Gen_ComponentAvailable
Fnd_Gen_ComponentCheck
Fnd_Gen_ComponentData
Fnd_Gen_GetString
Fnd_Gen_GetDatabaseInfo
Fnd_Gen_GetString
Fnd_Gen_Info
Fnd_Gen_LaunchAsNewProcess
Fnd_Gen_MenuBar
Fnd_Gen_Platform

Fnd_Gen_ComponentInfo	Fnd_Gen_PlugInAvailable
Fnd_Gen_ComponentReport	Fnd_Gen_QuitNow
Fnd_Gen_CurrentFormType	Fnd_Gen_RemoveActivationCode
Fnd_Gen_CurrentTable	Fnd_Gen_Reset
Fnd_Gen_FileName	Fnd_Gen_SelectionChanged
Fnd_Gen_FormMethod	Fnd_Gen_SetDatabaseInfo

Fnd_Gen_BugAlert

Fnd_Gen_BugAlert (method{; details})

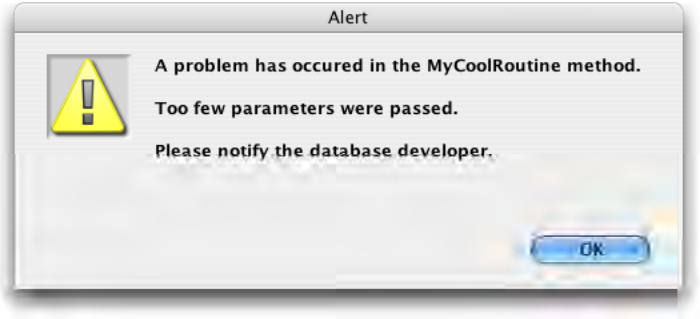
Parameter	Type	Description
method	Text	Method name
details	Text	Details about the problem (optional)

The *Fnd_Gen_BugAlert* method provides a simple method for the developer to notify the end-user of an problem that can happen only as a result of a programming error.

For example, if you create a method that requires two parameters, you can use 4D's **Count parameters** function to determine if two parameters have been passed. If the wrong number of parameters has been passed, you would use the *Fnd_Gen_BugAlert* method to alert the end-user to the problem. The *Fnd_Gen_BugAlert* method will describe the problem and ask the user to notify the database developer.

```
If(Count parameters<2)
  Fnd_Gen_BugAlert (Current method name;"Too few parameters were passed.")
End if
```

If you want to provide a more specific error message, pass a complete sentence as the second parameter. This sentence will be displayed following the first sentence in the error message as shown below.



A problem has occured in the MyCoolRoutine method.

Too few parameters were passed.

Please notify the database developer.

Do not use 4D's **Current method name** function as the first parameter to this method if you are calling it from a 4D component designed for 4D 2003.2 or earlier. With these earlier versions of 4D 2003, 4D does not return the method's name if **Current method name** is called from a method in a component. Instead, you must hard-code the current method name.

This method is intended to display errors in the code, not runtime errors that occur because a file is not where you expect it to be, or a value is not entered properly. Put this method only in places where you assume it will never be called.

After displaying the alert, this routine calls 4D's **TRACE** command to help you debug the error. The **TRACE** command is ignored if the database is compiled.

Fnd_Gen_ButtonText

Fnd_Gen_ButtonText (button; label; alignment{; ->obj1..->objN}) ➔ Number

Parameter	Type	Description
button	Pointer	A pointer to the button
label	Text	New button text
alignment	Longint	Which end to keep stationary
obj1..objN	Pointer	Other form objects to move (optional)
Function result	Number	The increase in the button size

Sets the button text and resizes the button if necessary. The side passed as the alignment parameter remains stationary, while the other side will be modified if necessary to allow the text to fit. The button will only be enlarged if necessary, never made smaller than its original form size.

4D Constant	Type	Value
Align Left	Longint	2
Center	Longint	3
Align Right	Longint	4

You can optionally pass pointers to other form objects. If the button is resized, these objects will be moved an equal amount so that they maintain an equal distance from the resized button. If you pass Align Left as the alignment, then pass pointers to the objects located directly to the right side of the button (the side that might be moved).

The amount of the increase in the button size is returned as the result of this function. This value can be used if you need to procedurally modify aspects of other form objects.

For example, imagine we have a **Find** and **Cancel** button set located in the lower-right portion of a dialog. If we localize the button text using 4D's **BUTTON TEXT** command, the button labels change, but the buttons remain the same size.

```
BUTTON TEXT(bFind;"Rechercher")
BUTTON TEXT(bCancel;"Annuler")
```

By replacing 4D's **BUTTON TEXT** command with the *Fnd_Gen_ButtonText* method, the buttons are now resized to fit the new labels.

```
Fnd_Gen_ButtonText (->bFind;"Rechercher";Align Right )
Fnd_Gen_ButtonText (->bCancel;"Annuler";Align Right )
```

However, now the **Find** button has expanded so that the **Cancel** button is no longer properly positioned. So we can pass a pointer to the **Cancel** button to *Fnd_Gen_ButtonText* when we move the **Find** button. This tells the *Fnd_Gen_ButtonText* method to shift the **Cancel** button so that it retains its original relationship with the **Find** button.

```
Fnd_Gen_ButtonText (->bFind;"Rechercher";Align Right ;->bCancel)
Fnd_Gen_ButtonText (->bCancel;"Annuler";Align Right )
```

Fnd_Gen_CancelQuit

Fnd_Gen_CancelQuit

Parameter	Type	Description
No parameters required.		

When you call Foundation's *Fnd_Gen_QuitNow* routine to quit the database, Foundation spends about 30 seconds waiting for all of the processes to end. If you want Foundation to stop telling the other processes to quit, call this method. This command can be called from any process to interrupt Foundation's attempt to quit.

Fnd_Gen_CenterWindow

Fnd_Gen_CenterWindow (width; height{; type{; position{; title{; close}}}})

Parameter	Type	Description
width	Longint	Width of the new window
height	Longint	Height of the new window
type	Longint	Window type (optional)
position	Longint	Window position (optional)
title	Text	Window title (optional)
close	Boolean	Has a close box (optional)
Function result	Longint	Window reference number

The *Fnd_Gen_CenterWindow* routine opens a new window centered on the screen or over the front most window (depending on the value of position parameter). It is included primarily for use by other components that do not wish to require the presence of the Windows component. If the Foundation Windows component is installed in the database, use the *Fnd_Wnd_OpenWindow* or *Fnd_Wnd_OpenFormWindow* routine instead.

Fnd_Gen_ComponentAvailable

Fnd_Gen_ComponentAvailable (component name) ➔ Boolean

Parameter	Type	Description
component name	Text	Name of the component
Function result	Boolean	True if the component is available

Use the *Fnd_Gen_ComponentAvailable* routine to determine if a specific Foundation component is available. If it is, you can then call the component using 4D's **EXECUTE** command. This allows your method to behave differently if a component is available, yet still compile without the component installed.

You can also make your own routines compatible with this routine by creating an initialization routine that ends with "_Init." For example, if you create a component named "MyComponent," just create a routine named "MyComponent_Init." It does not matter what it does — it is only important that it does not cause an error, or at least resets 4D's Error variable to 0. You can then test for the component this way:

```
If(Fnd_Gen_ComponentAvailable ("MyComponent"))
  EXECUTE("MyComponentMethod")
End if
```

See the source code to Foundation's *Fnd_Gen_MenuBar* method to see how this function can be used.

Fnd_Gen_ComponentCheck

Fnd_Gen_ComponentCheck (calling component; required component)

Parameter	Type	Description
calling component	Text	Internal name of the calling component
required component	Text	Internal name of the required component

The *Fnd_Gen_ComponentCheck* routine displays an alert if the specified component is not available. It then ends execution by calling 4D's **ABORT** command. Use this command in the initialization of a component that requires another Foundation component.

```
Fnd_Gen_ComponentCheck("My Cool Component";"Fnd_Dlg")
Fnd_Gen_ComponentCheck("My Cool Component";"Fnd_Wnd")
```

Fnd_Gen_ComponentData

Fnd_Gen_ComponentData (->codes array{; ->inst vers {; ->cur vers}})

Parameter	Type	Description
codes array	Pointer	A text array to receive the component prefixes
inst vers	Pointer	A text array to receive the installed version numbers (optional)
cur vers	Pointer	A text array to receive the current version numbers (optional)

This method returns in the passed arrays information about the Foundation components. All known Foundation components are returned in the arrays, even if the component is not currently installed. If a component is not installed, empty text strings are returned in the current versions array.

```
ARRAY TEXT(aComponentsPrefixes;0)
ARRAY TEXT(aInstalledVers;0)
ARRAY TEXT(aCurrentVers;0)
Fnd_Gen_ComponentData (->aComponentsPrefixes;->aInstalledVers;->aCurrentVers)
```

To get a component's full name from the prefix, call the component's _Info method:

```
ARRAY TEXT(aComponentsPrefixes;0)
Fnd_Gen_ComponentData (->aComponentsPrefixes)
ARRAY TEXT(aComponentNames;Size of array(aComponentsPrefixes))
For($i;1;Size of array(aComponentNames))
  aComponentNames{$i}:=Fnd_Gen_ComponentInfo (aComponentsPrefixes{$i};”name”)
End for
```

Fnd_Gen_ComponentInfo

Fnd_Gen_ComponentInfo (component name; info requested) ➔ Text

Parameter	Type	Description
component name	Text	A component prefix
info requested	Text	Desired information
Function result	Text	Information requested or an error message

Many of the Foundation components include a method designed to return information about the component or the current status of the component. These information methods are named with the component's prefix, followed by "_Info" (e.g. *Fnd_Loc_Info*).

In most cases, you will want to call these information methods directly. However, you may want to make a component optional. In this case, you cannot call the method directly, because your structure would not compile if you include a call to a method that does not exist. You can avoid this problem by using the *Fnd_Gen_ComponentInfo* method. It will call another component's _Info method, after first checking to see if the component is installed.

Pass this method a component name and name for the information you want. If the component exists, Foundation will call its _Info routine, pass it the info label, and return the result. This allows you to get information from a component in a single call, rather than having to first test to see if the component is available.

```
$language:=Fnd_Gen_ComponentInfo ("Fnd_Loc";"language")
```

If the component is not installed, "Component Not Available" will be returned.

```
$language_t:=Fnd_Gen_ComponentInfo ("Fnd_Loc";"language")
If ($language_t="Component Not Available")
  $language_t:="EN"
End if
```

If the component does not have an _Info routine, "Component Did Not Respond" will be returned.

We recommend that if a component does not recognize a request it should return the text "Component Did Not Recognize Request". These values will not be localized so the calling method can test for them.

Fnd_Gen_ComponentReport

Fnd_Gen_ComponentReport

Parameter	Type	Description
No parameters required.		

This routine displays Foundation's Component Report window. The window displays a list of all of the known Foundation components, along with the most recent version numbers known to the Foundation General component, and the version numbers of any components currently installed in the database.

This routine is available from Menu Bar #1 (Fnd_Shell) in the Foundation shell. See Tools menu item Component Report.

Fnd_Gen_CurrentFormType

Fnd_Gen_CurrentFormType ({form type}) ➔ Number

Parameter	Type	Description
form type	Longint	Form type constant
Function result	Number	Form type constant

If no parameters are passed, *Fnd_Gen_CurrentFormType* returns the type of the currently displayed form. Here are the values that this routine might return:

Value	Constant	Description
0	Fnd_Gen_NoForm	No window
1	Fnd_Gen_UnknownForm	Unknown form type
2	Fnd_Gen_OutputForm	An output form
3	Fnd_Gen_InputForm	An input form
4	Fnd_Gen_PreferencesForm	The Preferences window
5	Fnd_Gen_AboutForm	The About box

Use this method to determine the contents of the front most window. This gives you a way to determine if an input form or an output form is front most. Constants are also included to determine if other Foundation windows are front most.

Use this routine from methods called from the Administration dialog, the Print dialog, and the Special Functions dialog to determine if the user is viewing a record or selection of records. For example, it may be important to a printing routine to know at runtime if the user is viewing an input form or an output form.

You can pass a number to this routine from your own non-modal windows if you want to later be able to determine if they are front most.

Fnd_Gen_CurrentTable

Fnd_Gen_CurrentTable ({->table}) ➔ Pointer

Parameter	Type	Description
table	Pointer	Pointer to the table to make current (optional)
Function result	Pointer	Current table

Rather than using 4D's **DEFAULT TABLE** command and **Current default table** function, Foundation maintains a variable to a pointer to the current table for each process.

You can get the value of this pointer by calling this method without any parameters. If no table is current, a nil pointer is returned.

Pass a pointer to a table to this routine to set it as Foundation's current table for the current process.

```
$tablePtr:=Fnd_Gen_CurrentTable
DELETE RECORD($tablePtr->)
```

or

```
DELETE RECORD(Fnd_Gen_CurrentTable ->)
```

Fnd_Gen_FileName

Fnd_Gen_FileName (path) ➔ Text

Parameter	Type	Description
path	Text	Full path name
Function result	Text	File name at the end of the pathname

To get the name of a file from a full file path, pass the path to this function.

```
$title:=Fnd_Gen_FileName (Document)
SET WINDOW TITLE($title)
```

Fnd_Gen_FormMethod

Fnd_Gen_FormMethod

Parameter	Type	Description
No parameters required.		

This method is designed to be called from any non-modal forms in the application. It checks to see if the close box has been clicked, or if another process has requested the window to close.

Any non-modal forms added to a Foundation database should call this method with the following form events enabled:

On Activate
On Outside Call
On Close Box

Fnd_Gen_GetDatabaseInfo

Fnd_Gen_GetDatabaseInfo (info type) ➔ Text

Parameter	Type	Description
info type	Text	Info type label
Function result	Text	Value

4D's component architecture does not allow variables to be shared between components. That is a good thing, but it requires us to find another way to share information between components. In Foundation, we do this with the *Fnd_Gen_SetDatabaseInfo* and *Fnd_Gen_GetDatabaseInfo* methods.

Foundation maintains a set of text arrays to store general information about the shell. Information is added to these arrays using the *Fnd_Gen_SetDatabaseInfo* method, and you can get at this information using the *Fnd_Gen_GetDatabaseInfo* method.

By default, Foundation's General component stores just one bit of information: the version number of the Fnd_Gen component.

In the Foundation Shell, the *Fnd_Hook_Shell_Setup* hook uses the *Fnd_Gen_SetDatabaseInfo* method to set up four other pieces of information:

Label	Data
DatabaseName	The name of the database
DatabaseVersion	The version number of the database
DatabaseCopyright	The database's copyright information
DatabaseURL	The URL for the About dialog

You can store any text information in these variables that you need to share between components. These routines are designed for storing hard-coded text strings for communication between components. To store user preference related text strings, use Foundation's Preferences component.

Fnd_Gen_GetString

Fnd_Gen_GetString (module; lookup code{; param 1{; param 2...}}) ➔ Text

Parameter	Type	Description
module	Text	The module name
lookup code	Text	The lookup code
param 1 .. param 2	Text	Replacement parameters (optional)
Function Result	Text	The localized string

This routine offers a way to get localized **OK** and **Cancel** button for a dialog without requiring Foundation's Localization component. Pass it the internal string label, and this routine gets the associated string from the 4D application. Optional replacement strings may be passed as parameters 3 through 5. See documentation for *Fnd_Loc_GetString* for complete description.

Here is a list of the labels this command accepts, and the English versions of the associated button labels:

Internal Label	Button Label
OK	OK
Cancel	Cancel
DontSave	Don't Save
Stop	Stop

```
$windowTitle_t:=Fnd_Gen_GetString ("Fnd_Rec";"ModWindowTitle")
```

Fnd_Gen_Info

Fnd_Gen_Info (info requested) ➔ Text

Parameter	Type	Description
info requested	Text	Info desired
Function Result	Text	Response

This function returns the requested information about the General component.

```
$version_t:=Fnd_Gen_Info ("version")
```

The *Fnd_Gen_Info* method will respond to these requests:

Request	Response	Example
name	The component's full name	Foundation General
version	The component's version number	4.1.4

See the *Fnd_Gen_ComponentInfo* method for more information.

Fnd_Gen_LaunchAsNewProcess

Fnd_Gen_LaunchAsNewProcess (method; process name) ➔ Boolean

Parameter	Type	Description
name	Text	The name of the method to launch
process name	Text	Name to give the new process
Function result	Boolean	True if we are in a new process

The *Fnd_Gen_LaunchAsNewProcess* method launches the specified process as a new process. If the process is already running, it is just brought to the front.

This process is designed so a method can launch itself in a new process. It is used like this:

```
If(Fnd_Gen_LaunchAsNewProcess ("ThisMethodName";"$New process name"))
    ` Any code placed here will run in the new process.
End if
```

See the source to Foundation's *Fnd_Art_About* method to see a more complete example.

Fnd_Gen_MenuBar

Fnd_Gen_MenuBar

Parameter	Type	Description
No parameters required.		

Instead of calling 4D’s **MENU BAR** command, call the *Fnd_Gen_MenuBar* routine to install and update Foundation’s menu bar when you display a window in a new process. In addition to calling the **MENU BAR** command, this routine will localize the menus (using the Localization component, if available), enable and disable the appropriate menu items, and update the **Window** menu.

Fnd_Gen_Platform

Fnd_Gen_Platform ➔ Number

Parameter	Type	Description
Function result	Longint	Platform number

This function returns the platform number from 4D’s **PLATFORM PROPERTIES** command. This allows you to easily distinguish between Mac OS X and Windows.

```
Case of
  : (Fnd_Gen_Platform =Windows )
    ` We are using Windows
  : (Fnd_Gen_Platform =Mac OS)
    ` We are using Mac OS X

End case
```

Fnd_Gen_PlugInAvailable

Fnd_Gen_PlugInAvailable (plugin name) ➔ Boolean

Parameter	Type	Description
plugin name	Text	Name of the plugin
Function result	Boolean	True if the plugin is available

This function returns **True** if the specified plugin is installed. The plugin name must be passed exactly as shown in the list below.

This routine is only aware of a limited number of plugins. Please contact us if you would like support for a specific plugin added to this routine.

This routine can currently test for the following plugins:

Foundation Extras
4D View
4D Write
4D Draw
ToolboxPack

Currently this routine can only detect the English versions of the above 4D plugins (4D View, 4D Write, and 4D Draw).

Fnd_Gen_QuitNow

Fnd_Gen_QuitNow ({quit?}) ➔ Boolean

Parameter	Type	Description
quit?	Boolean	Cause Foundation to quit now? (optional)
Function result	Boolean	True if it is time to quit

This routine has two different formats.

If you create a background process that runs until the user selects **Quit**, you should call this method on a regular basis. If you do not pass a parameter to this method, it will return **True** if the user is trying to quit the application. When Foundation tries to quit, it will wake any paused or delayed processes so they can check the value returned by this function.

```
While(Not(Fnd_Gen_QuitNow ))
  ` Do background stuff here.
End if
```

The second way to use this routine is to use it to tell Foundation you want to quit the application, just as if the user had selected **Quit** from the **File** menu. Just pass **True** to the method, and Foundation will attempt to quit all processes, and then call **QUIT 4D** (if the database is compiled).

```
Fnd_Gen_QuitNow (True)
```

Fnd_Gen_RemoveActivationCode

Fnd_Gen_RemoveActivationCode

Parameter	Type	Description
No parameters required.		

This method removes the activation code from the database structure file so the developer can share the source code with the Foundation components installed. This method can be called from Menu Bar #1 in the Custom Menus environment if you are using the Foundation Shell. This is found under the Tools menu.

Fnd_Gen_Reset

Fnd_Gen_Reset

Parameter	Type	Description
No parameters required.		

Fnd_Gen_Reset resets all of the available Foundation components. This routine does not actually clear any variables, it just flags them as needing to be re-initialized, so the next time each component is used, it will reset its own variables.

The Foundation Shell calls this method when you quit, so the variables are all reset when you restart Foundation during development.

Fnd_Gen_SelectionChanged

Fnd_Gen_SelectionChanged

Parameter	Type	Description
No parameters required.		

If your code changes the current selection of records displayed in an output form, call this routine at the end of your method to let Foundation know about the change. This routine will ensure that the menus and toolbar buttons are properly updated for the contents of the current selection.

This method calls the *Fnd_Hook_IO_SelectionChanged* hook.

Fnd_Gen_SetDatabaseInfo

Fnd_Gen_SetDatabaseInfo (info type; value)

Parameter	Type	Description
info type	Text	Info type label
value	Text	Info value

Use this routine to set text values that can be later tested from other components using the *Fnd_Gen_GetDatabaseInfo* method. You can set and get any type of text information using this routine. Set values are available to all processes.

```
Fnd_Gen_SetDatabaseInfo("SecretCodeWord";"firefox")
```

See the *Fnd_Gen_GetDatabaseInfo* method for more information.

Input/Output Component

(Fnd_IO)

The Foundation IO component is responsible for displaying input for designed specifically for use with the Foundation Shell. It is not intended to be used in other database projects.

Installation

It contains no tables or forms, and no additional steps are required after installation.

The IO component stores information in a database table.

4D v11 does not allow tables in components. To work around this problem, the Foundation components do not include any 4D tables. Instead, you will need to add any required tables to your structure before using the component. Internally the components create pointers to the structure's tables and fields and then uses these throughout the code.

Create the [Fnd_Pref] Table

To use the Foundation Preferences component, you will first need to add a table to your structure. You can simply copy the [Fnd_Pref] table from the Product Sales.4DB sample file, or you can create it manually in 4D (this is a good opportunity to reuse an unused or deleted table). The table must be named "Fnd_Pref" and must contain the fields listed below. The order of the fields is not important, and it is okay if the table contains other unused fields (in case you are reusing a table). However, the field names and types must be set up exactly as shown below.

Field Name	Type	Attributes
[Fnd_Pref]ID	Long Integer	
[Fnd_Pref]Owner	Alpha 80	Indexed
[Fnd_Pref]Name	Alpha 80	Indexed
[Fnd_Pref]Type	Integer	
[Fnd_Pref]Value	Text	

Install the Component

The Foundation Preferences component requires the following components (shown with minimum required version numbers). It also requires version 4.2 or later of the Foundation Extras plugin.

Component	Minimum Version
Fnd_Gen	4.2

Updating the Component

If you later upgrade (or reinstall) the component, you will be asked if you want to update the public Fnd_Pref_Preferences form. Click **Yes** if you have not customized this form for your application, or if you wish to revert to the default layout. Otherwise click **No** to preserve any changes you may have made.

Updating the Component

The component's "Fnd_Menu" menu bar has public access, so that you can modify it to meet your needs. You can safely add and delete menu items using the 4D Menu Editor. So be sure to backup your structure before upgrading this component to ensure you can restore your changes if necessary.

Language Reference

Here is the list of routines in Foundation's IO component:

- Fnd_Hook_IO_DisplayRecord
- Fnd_Hook_IO_DisplayTable
- Fnd_Hook_IO_InputFormButton
- Fnd_Hook_IO_SelectionChanged
- Fnd_IO_AddMultipleRecords
- Fnd_IO_DisplayRecord
- Fnd_IO_DisplayNavButtons
- Fnd_IO_InputFormMethod
- Fnd_IO_InputFormName
- Fnd_IO_MultiWindow
- Fnd_IO_OutputFormMethod
- Fnd_IO_OutputFormName
- Fnd_IO_RecordEdited
- Fnd_IO_ToolbarIconGroup

Fnd_IO_DisplayTable	Fnd_IO_UpdateToolbar
Fnd_IO_Info	

Fnd_Hook_IO_DisplayRecord

Fnd_Hook_IO_DisplayRecord

Parameter	Type	Description
No parameters required.		

This hook is called just before an input window is displayed. Here you can set the window title or position, or specify the input form to use.

Use the *Fnd_Gen_CurrentTable* method to determine the current table.

The following example uses this hook to set the window title to something other than the default window title selected by Foundation.

```
Case of
  : (Fnd_Gen_CurrentTable=(->[Products])
    Fnd_Wnd_Title ([Products]Product Name)
End case
```

Fnd_Hook_IO_DisplayTable

Fnd_Hook_IO_DisplayTable

Parameter	Type	Description
No parameters required.		

This hook is called just before an output window is displayed. Here you can set the window title or position, or specify the output form to use.

Use the *Fnd_Gen_CurrentTable* method to determine the current table.

Fnd_Hook_IO_InputFormButton

Fnd_Hook_IO_InputFormButton ({button}) ➔ Boolean

Parameter	Type	Description
enable	Boolean	True to enable adding multiple records (optional)
Function result	Boolean	True if this feature is enabled

This hook is called when a button on the inherited form named Fnd_IO_InputForm is clicked. It receives as the a parameter the name of the button that was clicked. The name will be one of these text values:

Button Names
save
cancel
first
previous
next
last

Return **True** in $0 to allow Foundation to handle the click and perform the standard action. Return **False** to prevent Foundation from acting on the click.

```
C_BOOLEAN($0;$allow)
C_TEXT($1;$action)

$action:=$1
$allow:=True
Case of
  : (Fnd_Gen_CurrentTable =(->[Invoices]))
    Case of
      : ($action="save")
        If(Not(User in group(Current user;"Accounting"))
          Fnd_Dlg_Alert ("You are not authorized to modify invoice records.")
          $allow:=False
        End if
    End case
End case

$0:=$allow
```

Use the *Fnd_Gen_CurrentTable* method to determine the current table.

Fnd_Hook_IO_SelectionChanged

Fnd_Hook_IO_SelectionChanged

Parameter	Type	Description
No parameters required.		

This hook is called any time the current selection of records displayed in an output list has changed. You can filter the selection here (remove records the user should not see), sort the selection, etc. from this hook.

Use the *Fnd_Gen_CurrentTable* method to determine the current table.

The following example uses this hook to remove discontinued items from the selection of products displayed in the output form.

```
Case of
  : (Fnd_Gen_CurrentTable =(->[Products]))
    QUERY SELECTION([Products];[Products]Discontinued=False)
End case
```

Fnd_IO_AddMultipleRecords

Fnd_IO_AddMultipleRecords ({enable}) ➔ Boolean

Parameter	Type	Description
enable	Boolean	True to enable adding multiple records (optional)
Function result	Boolean	True if this feature is enabled

By default, Foundation's *Fnd_IO_DisplayRecord* command adds one record, then either closes the input window or returns to the output list (depending on the current configuration — see the *Fnd_IO_MultiWindow* command). This command can be used to modify this behavior so that Foundation behaves similarly to 4D's User environment.

Passing **True** to this command will cause Foundation to continuously add new records until the input button's **Cancel** button is clicked. This command can be called from the *Fnd_Hook_IO_DisplayRecord* hook:

```
Fnd_IO_AddMultipleRecords (True)
```

This command will affect only the current process.

This routine can also be called as a function to determine the current configuration:

```
$addingMultipleRecs:=Fnd_IO_AddMultipleRecords
```

Fnd_IO_DisplayRecord

Fnd_IO_DisplayRecord ({->table{; record number}})

Parameter	Type	Description
table	Pointer	Pointer to the record's table (optional)
record number	Longint	Record number (optional)

Displays the specified record of the table in a new process. If no table is passed, then the current form's table is used (if available). If no record number is passed, then the current record will be displayed (if there is one). Pass 4D's New record constant as the record number parameter to create a new record.

Just after this routine launches a new process, and just before it opens a window, the *Fnd_Hook_IO_DisplayRecord* hook is called.

Fnd_IO_DisplayNavButtons

Fnd_IO_DisplayNavButtons ({enable}) ➔ Boolean

Parameter	Type	Description
enable	Boolean	True to display the navigation buttons(optional)
Function result	Boolean	True if enabled

Fnd_IO_DisplayNavButtons allows the developer to display or hide the input form record navigation buttons for the current process.

```
Fnd_IO_DisplayNavButtons (False)  ` Do not display the navigation buttons.
```

This command will affect only the current process. It can be called from the *Fnd_Hook_IO_DisplayRecord* hook.

This routine can also be called as a function to get the current state of this setting.

```
$displayNavButtons:=Fnd_IO_DisplayNavButtons
```

Fnd_IO_DisplayTable

Fnd_IO_DisplayTable (->table{; force new window})

Parameter	Type	Description
table	Pointer	Pointer to the table to display
force new window	Boolean	Open a new window (optional)

Pass a table pointer to the *Fnd_IO_DisplayTable* method whenever you want to display a new table window. If a window already exists for the specified table, it will be brought to the front. If a window is not already open to display the table, this routine will start a new process, open a new window, and display the specified table using the table's output form:

```
Fnd_IO_DisplayTable (->[People])
```

You can optionally force the *Fnd_IO_DisplayTable* method to open a new table window—even if one is already open—by passing a second parameter. If the second parameter evaluates to **True**, then a new process will be created to display the table. Otherwise, a new process will be created only if one does not already exist:

```
Fnd_IO_DisplayTable (->[People];True)
```

If you want to open a table window at startup, call this method from the *Fnd_Hook_Shell_Startup* method.

Foundation's Navigation Palette buttons and the Open Table Dialog call the this method to display table windows.

By default, the selection of records for the new window will contain all of the table's records. You can change this selection before the window is displayed in the *Fnd_Hook_IO_DisplayTable* hook. The hook is called just after this routine launches a new process, and just before it opens a window.

Fnd_IO_Info

Fnd_IO_Info (info requested) ➔ Text

Parameter	Type	Description
info requested	Text	Info desired
Function Result	Text	Response

Returns the requested information about the component.

```
$version_t:=Fnd_IO_Info ("version")
```

The *Fnd_IO_Info* method will respond to these requests:

Request	Response	Example
name	The component's full name	Foundation General
version	The component's version number	4.1

This routine can also be called using the *Fnd_Gen_ComponentInfo* method without first testing to see if the component is installed:

```
$version_t:=Fnd_Gen_ComponentInfo ("Fnd_Gen";"version")
```

See the *Fnd_Gen_ComponentInfo* method for more information.

Fnd_IO_InputFormMethod

Fnd_IO_InputFormMethod

Parameter	Type	Description
No parameters required.		

The *Fnd_IO_InputFormMethod* method must be called from the form method of each input form that will be displayed in the Custom Menus environment. This method makes sure that the menus get updated when the form is displayed and when its window is brought to the front from behind other windows.

It also ensures that the window gets closed properly when the user quits from the database or when the **Close All Windows** command is selected from the **Window** menu.

This method assumes these form events are enabled:

On Activate

On Close Box
On Deactivate
On Load
On Outside Call
On Resize

Fnd_IO_InputFormName

Fnd_IO_InputFormName ({form name}) ➔ Text

Parameter	Type	Description
form name	Text	The new input form name (optional)
Function result	Text	The input form name

This method allows the developer to get and set the name of the form to use for any input screens opened by the current process. To set the name of the next input form name to use, pass the input form name to this routine from the *Fnd_Hook_IO_DisplayRecord* hook.

```
Fnd_IO_InputFormName ("AdminInput")
```

No parameter is needed when using this method as a function to determine the currently set form name:

```
$currentFormName:=Fnd_IO_InputFormName
```

Fnd_IO_MultiWindow

Fnd_IO_MultiWindow ({enable}) ➔ Boolean

Parameter	Type	Description
form name	Text	True to enable multiple window mode (optional)
Function result	Boolean	True if multiple window mode is enabled

By default, Foundation displays input forms in a new process and window. This routine allows the developer to change this behavior so that input forms are displayed in the same process and window as the output form when double-clicking on a record in an output list.

Pass **True** to enable multi-window mode, or pass False to disable multi-window mode.

```
Fnd_IO_MultiWindow (True)
```

This command will affect only the current process. It can be called from the *Fnd_Hook_IO_DisplayTable* hook.

This routine can also be called without any parameters to determine the current setting:

```
$multiWindowMode:=Fnd_IO_MultiWindow
```

Fnd_IO_OutputFormMethod

Fnd_IO_OutputFormMethod

Parameter	Type	Description
No parameters required.		

The *Fnd_IO_OutputFormMethod* method must be called from the form method of each output form that will be displayed in the Custom Menus environment. This method makes sure that the menus get updated when the form is opened and when its window is brought to the front from behind other windows. It also ensures that the window gets closed properly when the user quits from the database, or when the **Close All Windows** command is selected from the **Window** menu.

It can be called as the only line of code in an output form, or before or after your own form method code.

This method assumes these form events are enabled:

On Load
On Unload
On Clicked
On Outside Call
On Double Clicked
On Activate
On Deactivate
On Resize
On Close Box
On Header

Fnd_IO_OutputFormName

Fnd_IO_OutputFormName (form name) ➔ Text

Parameter	Type	Description
name	Text	Output form name (optional)
Function result	Text	The current output form name

Call this method to set the name of the next output form to use. If you are using the Foundation Shell, you can call this method from the *Fnd_Hook_IO_DisplayTable* hook.

```
Fnd_IO_OutputFormName ("AccountingOutput")
```

This routine may also be called as a function to get the current output form name.

```
$outputForm:=Fnd_IO_OutputFormName
```

Fnd_IO_RecordEdited

Fnd_IO_RecordEdited ({->table; calling process}) ➔ Longint

Parameter	Type	Description
table	Pointer	A pointer to the new record’s table
calling process	Longint	The process number of the output form
Function result	Longint	The record number that should be added

This command is designed primarily for Foundation’s internal use, but it has been made available for use by replacements for Foundation’s default input and output handling routines.

Call this when a new record has been added or modified. Pass it the new record number and the number of the process that should be notified about this updated record. This method will call the other process, which can then get the value by calling this method without any parameters.

When this method is called without any parameters (due to an On Outside Call event usually), it will clear the value, so another outside call will not return the same number again.

If no record has been added, this routine will return -1, which is 4D’s No Current Record constant.

See the Foundation Construction Set for an example.

Fnd_IO_ToolbarIconGroup

Fnd_IO_ToolbarIconGroup ({style name}) ➔ Text

Parameter	Type	Description
style name	Text	The button style name to use (optional)
Function result	Text	The current style

This command changes the icons used when displaying the output form toolbar. The default style is “bold.” Use this command to switch the icon group to any of the other groups:

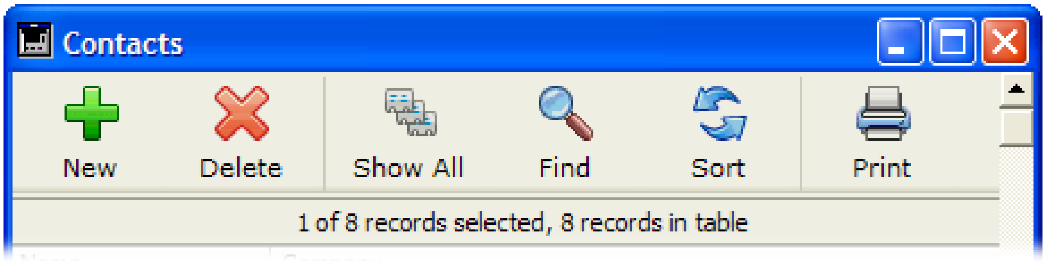
Icon Groups
Bold
Card
Native
Mac
Win

For example, this would switch the output toolbar icon group to “Card”:

```
Fnd_IO_ToolbarIconGroup ("Card")
```

This command will affect only the current process. It can be called from the *Fnd_Hook_IO_DisplayTable* hook.

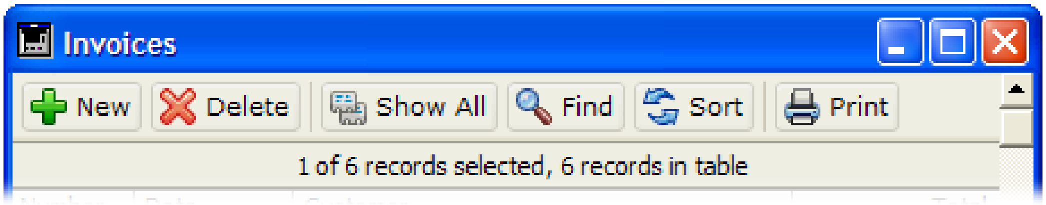
Here is what the different groups look like:



1 of 8 records selected, 8 records in table

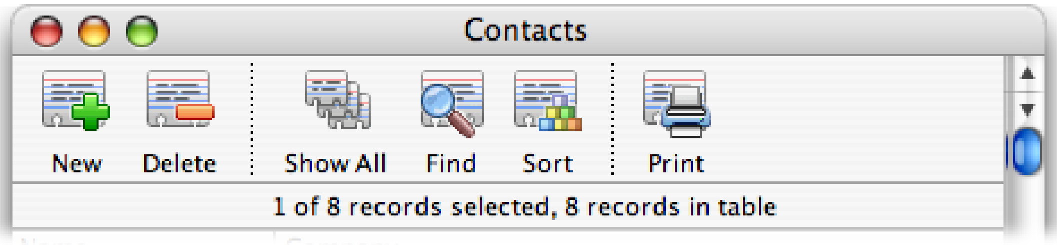
Icon Group: Bold

The icon group is not affected by the toolbar size. This command can be combined with calls to the *Fnd_Tlbr_Style* and *Fnd_Tlbr_Platform* commands.



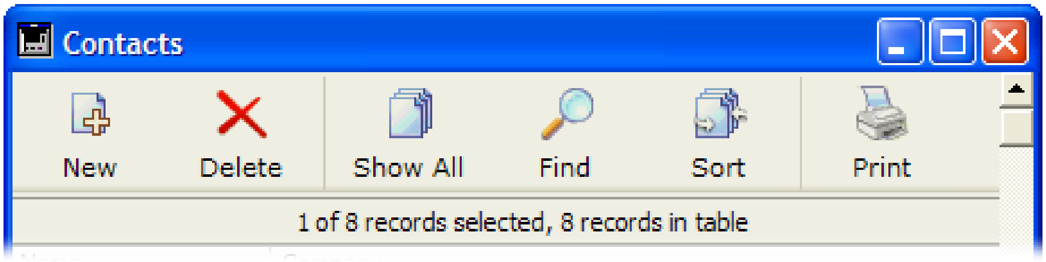
1 of 6 records selected, 6 records in table

Icon Group: Bold
Style: Small2



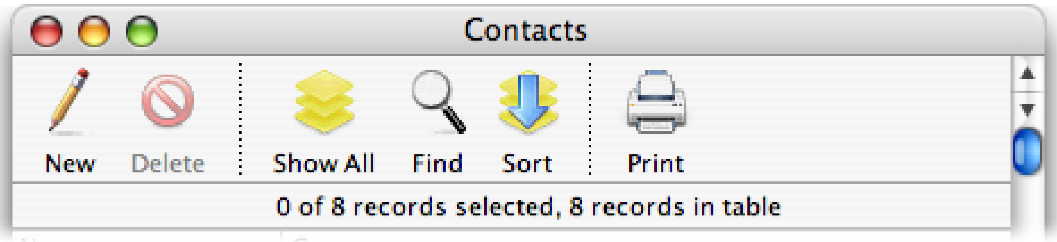
1 of 8 records selected, 8 records in table

Icon Group: Card



1 of 8 records selected, 8 records in table

Icon Group: Native or Win



0 of 8 records selected, 8 records in table

Icon Group: Native or Mac

The “Native” platform will automatically select either the “Win” or “Mac” group for you, depending on the current platform. This toolbar style works only with the large toolbar size.

This routine can also be called without any parameters to determine the current setting:

`$iconGroupName:=`*`Fnd_IO_ToolbarIconGroup`*

Fnd_IO_UpdateToolbar

Fnd_IO_UpdateToolbar

Parameter	Type	Description
No parameters required.		

Enables and disables the toolbar buttons based on the current selection of records and whether or not any records are highlighted. Generally there will be no need to ever call this method. It is available primarily so that it can be used by the optional Foundation Grid component.

Lists Component

Fnd_List

There are three main parts of Foundation's Lists component. First, the component can automatically update 4D lists when a new structure is delivered, so the user's changes are not lost. The component also provides editors for these lists so they can easily be modified in the Custom Menus environment. And finally, the Lists component offers two easy to use choice list dialogs.

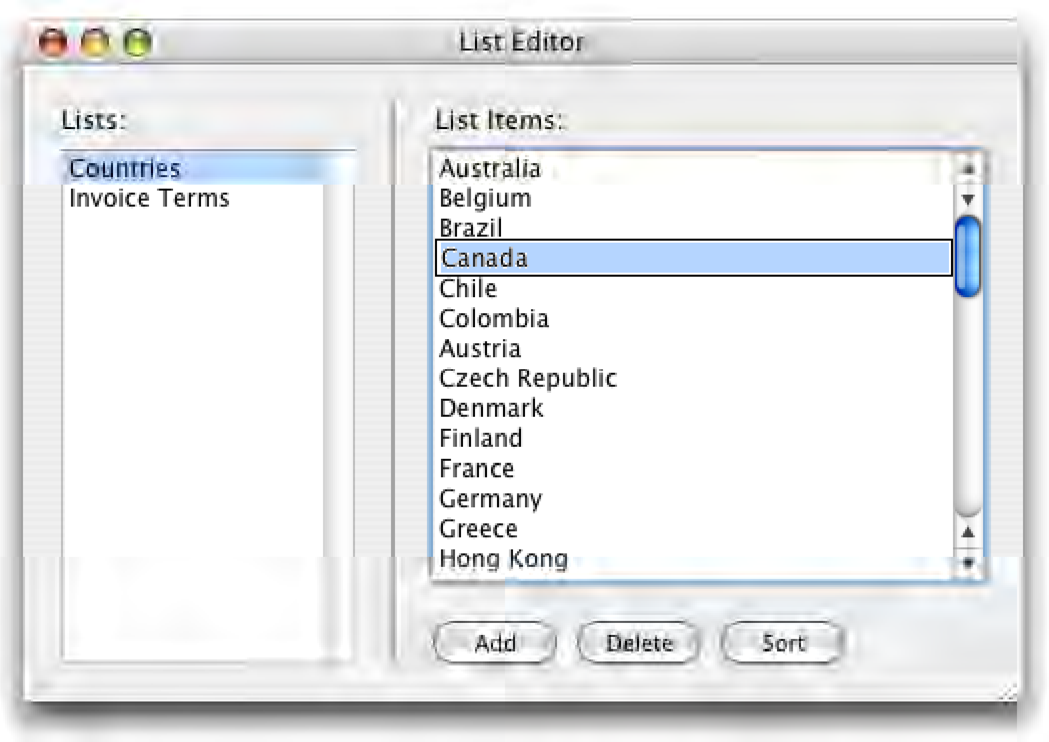
Automatic 4D List Updating

Since 4D lists are stored in the database's structure file, any changes made by end users are usually lost when a new structure is delivered. Foundation solves this problem by copying a lists contents to records in the data file. The next time the database is launched, Foundation compares a date/time stamp in the structure file with a date/time stamp in the data file. If they do not match, Foundation updates the lists in the structure file with the lists in the data file.

To take advantage of this feature, you must call the *Fnd_List_OnStartup* method in the database's startup method, and then call the *Fnd_List_OnExit* method in the *On Exit* database method. If you are using the Foundation Shell, these methods are called for you automatically. Then pass the name of any list you want Foundation to manage to the *Fnd_List_AddToListEditor* method. This makes the list available in Foundation's List Editor window, and tells Foundation to preserve the user's version of the list when the structure is updated. If you are using the Foundation Shell, you can do this in the *Fnd_Host_List_SetEditableLists* hook.

4D Lists Editor

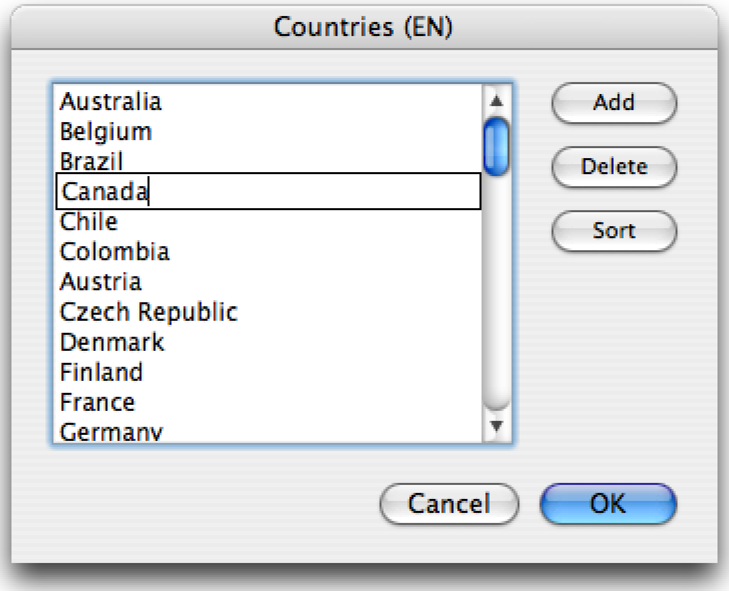
Foundation Lists offers a list editor to allow end users to edit multiple 4D lists in a single non-modal window. To display a list of all of the end user editable lists in the database, call the *Fnd_List_Editor* method. This will display the List Editor window:



The user can select a list to modify, then add, delete, and edit list items. To add items to this list, pass the name of the 4D list to the *Fnd_List_AddToListEditor* method.

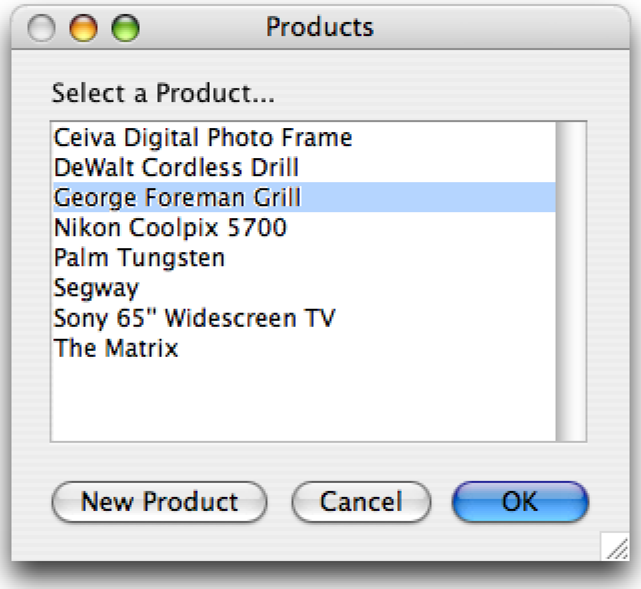
Single 4D List Editor

You can also display a single list for the user to edit by calling the *Fnd_List_EditOne* method.



Choice List Dialog

The Lists component also offers a basic choice list dialog that can be used to present a list of options from which the user can select one item. Here is an example of Foundation's Choice List dialog:



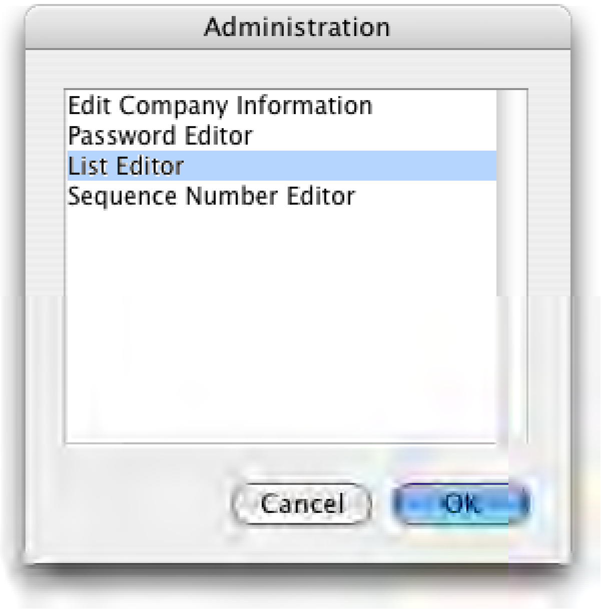
To display this dialog, just pass a text array to the *Fnd_List_ChoiceList* function, and it will return the selected element number.

Here is the code that was used to create this dialog:

```
ALL RECORDS([Products])
SELECTION TO ARRAY([Products]ID;$productIDs_ai;[Products]Name;Products_at)
SORT ARRAY(Products_at;$productIDs_ai)

Fnd_Wnd_Title ("Products")
Fnd_List_AddButton ("New Product")
$element_i:=Fnd_List_ChoiceList ("Select a Product...";->Products_at)

Case of
  : (OK=1)  ` A product was selected.
    CREATE RECORD([Line Items])
    [Line Items]ID:=Sequence number([Line Items])
    [Line Items]Product_ID:=$productIDs_ai{$element_i}
    SAVE RECORD([Line Items])

  : (OK=2)  ` The "New" button was clicked.
    Fnd_IO_DisplayRecord (->[Products];New record )
End case
```

Notice that the **New Product** button shown here is both optional and completely under the developer's control. This optional third button can have any label, and do anything when clicked (although it will always close the choice list dialog first).

Command Dialog



The *Fnd_List_CommandDialog* allows you to present the user with a list of actions to take (it uses the *Fnd_List_ChoiceList* function), and then executes the selected action. The Foundation Shell uses this feature to present the Administration, Print, and Special Functions dialogs. Just call the *Fnd_Host_List_SetEditableLists* method once for each item you want to add to the list, then call the *Fnd_List_CommandDialog* method.

Here is the code that was used to display the above dialog:

```
Fnd_List_AddItem ("Edit Company Information";"Company_EditSettings")
Fnd_List_AddItem ("Password Editor";"EDIT ACCESS")
Fnd_List_AddItem ("List Editor";"Fnd_List_Editor")
Fnd_List_AddItem ("Sequence Number Editor";"Fnd_SqNo_Editor")
Fnd_List_CommandDialog
```

Installation

This section describes how to install the Fnd_List component into a database that is not based on the Foundation Shell. If you started your project using the Foundation Shell, the Sequence Numbers component is already installed and integrated into your database.

The Sequence Numbers component stores information in a database table.

4D v11 does not allow tables in components. To work around this problem, the Foundation components do not include any 4D tables. Instead, you will need to add any required tables to your structure before installing the component. Internally the components create pointers to the structure's tables and fields and then uses these throughout the code.

Create the [Fnd_List] Table

To use the Foundation Lists component, you will first need to add a table to your structure. You can simply copy the [Fnd_List] table from the Product Sales.4DB sample file, or you can create it manually in 4D (this is a good opportunity to reuse an unused or deleted table). The table must be named "Fnd_List" and must contain the fields indicated below. The order of the fields is not important, and it is okay if the table contains other unused fields (in case you are reusing a table). However, the field names and types must be set up exactly as shown below.

Field Name	Type	Attributes
[Fnd_List]ID	Long Integer	Indexed
[Fnd_List]List_Name	Alpha 80	Indexed
[Fnd_List]List_Blob	BLOB	

Install the Component

The Foundation Lists component requires the following components (shown with minimum required version numbers). It also requires version 4.2 or later of the Foundation Extras plugin.

Component	Minimum Version
Fnd_Gen	4.2
Fnd_Dlg	4.1.3
Fnd_Wnd	4.1.4

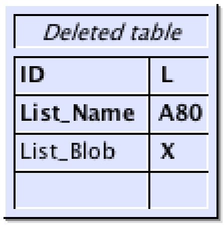
After you have added the [Fnd_List] table to your structure, install the Fnd_List component.

Updating the Component

If you later upgrade (or reinstall) the component, you will be asked if you want to update the public Fnd_List_Preferences form. Click **No** to preserve any changes you may have made to this form. Click **Yes** only if you have not customized this form for your application, or if you wish to revert to the default layout.

If you are upgrading a database that currently has version 4.1.4 or earlier of the Foundation Lists component, you will need to first uninstall that component, then install the 4.2 or later version. The 4.2 and later versions cannot directly update an existing older version of the component.

Removing the earlier version of the component (which included a table) will leave a table named "Deleted table" in your structure. After installing the new version of the component, launch 4D and switch to the Design environment (ignore the error messages that will be displayed). Change the name of the deleted table that looks like the one shown here back to "Fnd_List." Then quit and relaunch 4D. This time the database should launch without errors.



Deleted table

ID	**L**
List_Name	**A80**
List_Blob	**X**

Language Reference

Here are the routines in the Foundation Lists component:

Fnd_Host_List_SetEditableLists
Fnd_List_AddButton
Fnd_List_AddItem
Fnd_List_AddToListEditor
Fnd_List_ChoiceList
Fnd_List_CommandDialog
Fnd_List_EditOne
Fnd_List_Editor
Fnd_List_Info
Fnd_List_OnExit
Fnd_List_OnStartup

Fnd_Host_List_SetEditableLists

Fnd_Host_List_SetEditableLists

Parameter	Type	Description
No parameters required.		

This hook gives you a place to call the *Fnd_List_AddToListEditor* method. This hook is called during the startup process by Foundation.

```
Fnd_List_AddToListEditor ("Invoice Terms")
Fnd_List_AddToListEditor ("Country Names")
```

Fnd_List_AddButton

Fnd_List_AddButton (button label)

Parameter	Type	Description
button label	Text	The new button's label

The *Fnd_List_AddButton* routine adds a third button to Foundation's choice list or command dialog. Pass it the button label to display. If the button is clicked, the dialog will be closed and the OK variable will be set to 2.

```
ALL RECORDS([Products])
SELECTION TO ARRAY([Products]ID;$productIDs_ai;[Products]Name;Products_at)
SORT ARRAY(Products_at;$productIDs_ai)

Fnd_List_AddButton ("New Product")
$element_i:=Fnd_List_ChoiceList ("Select a Product...";->Products_at)

Case of
  : (OK=1)  ` A product was selected.
    ...  ` Use $element_i to determine the selection.
  : (OK=2)  ` The "New" button was clicked.
    ...
End case
```

Fnd_List_AddItem

Fnd_List_AddItem (command name; method name)

Parameter	Type	Description
command name	Text	Command name to add to the dialog
method name	Text	Method name to run if command name is selected

The *Fnd_List_AddItem* routine adds a command to the Command Dialog. Pass it the command name to display to the user, and the 4D method to call if the command is selected by the user. This routine can be called multiple times before calling the *Fnd_List_CommandDialog* command.

```
Fnd_List_AddItem ("Print Sales Report";"PrintSalesReport")
Fnd_List_AddItem ("End of Month Report";"PrintEOMReport")
Fnd_List_CommandDialog
```

Fnd_List_AddToListEditor

Fnd_List_AddToListEditor (list)

Parameter	Type	Description
list	Text	Name of the list to make available

The *Fnd_List_AddToListEditor* routine should called from the *Fnd_Host_List_SetEditableLists* method. Any list names passed to this method will become editable by the database administrator in the Edit Lists dialog.

```
Fnd_List_AddToListEditor ("Country Names")
```

Foundation will also ensure these lists are not replaced when the database structure is updated. It does this by keeping a copy of the list in the data file. If Foundation determines that the structure file may have been updated, it copies the stored copy of the list back into the structure file.

See the *Fnd_Host_List_SetEditableLists* command for an example of this command.

Fnd_List_ChoiceList

Fnd_List_ChoiceList (prompt; ->text array) ➔ Number

Parameter	Type	Description
prompt	Text	Prompt
text array	Pointer	Pointer to a text array
Function result	Number	Selected element number

The *Fnd_List_ChoiceList* allows the user to select an element from a list.

```
LIST TO ARRAY("MyList";MyTextArray)
Fnd_Wnd_Title ("Choices")
$element_i:=Fnd_List_ChoiceList ("Pick One";->MyTextArray)
If(OK=1)
  [Field]Value:=MyTextArray{$element_i}
End if
ARRAY TEXT(MyTextArray;0)  ` Clear the array.
```

Fnd_List_CommandDialog

Fnd_List_CommandDialog

Parameter	Type	Description
No parameters required.		

Fnd_List_CommandDialog displays the choice list dialog composed by calling the *Fnd_List_AddToListEditor* routine. See that command for an example.

The window's title and position can be set before calling this command using the Foundation Windows component's *Fnd_Wnd_Title* and *Fnd_Wnd_Position* commands.

```
Fnd_List_AddItem ("Print Sales Report";"PrintSalesReport")
Fnd_List_AddItem ("End of Month Report";"PrintEOMReport")
Fnd_List_CommandDialog
```

Fnd_List_EditOne

Fnd_List_EditOne (list name)

Parameter	Type	Description
list name	Text	Name of the list

The *Fnd_List_EditOne* routine allows the developer to edit the specified list in a semi-modal dialog.

```
Fnd_List_EditOne ("Country Names")
```

Fnd_List_Editor

Fnd_List_Editor

Parameter	Type	Description
No parameters required.		

The *Fnd_List_Editor* routine displays the List Editor window in a new process. Only 4D lists previously passed to the *Fnd_List_AddToListEditor* method will be displayed.

Fnd_List_Info

Fnd_List_Info (info requested) ➔ Text

Parameter	Type	Description
info requested	Text	Info desired
Function Result	Text	Response

Returns the requested information about the component.

```
$version_t:=Fnd_List_Info ("version")
```

The *Fnd_List_Info* method will respond to these requests:

Request	Response	Example
name	The component's full name	Foundation Lists
version	The component's version number	4.0.3 alpha 1

This routine can also be called using the *Fnd_Gen_ComponentInfo* method without first testing to see if the component is installed:

```
$version_t:=Fnd_Gen_ComponentInfo ("Fnd_List";"version")
```

See the *Fnd_Gen_ComponentInfo* method for more information.

Fnd_List_OnExit

Fnd_List_OnExit

Parameter	Type	Description
No parameters required.		

The *Fnd_List_OnExit* routine stores the current date and time in both the structure file (as a list) and the data file (as a record). We can compare these at startup to determine if the structure file has been updated.

Normally this command should be added to the *On Exit* and the *On Server* Shutdown database methods.

If you are using the Foundation Shell, this method is called automatically by the Fnd_Shell component.

Fnd_List_OnStartup

Fnd_List_OnStartup

Parameter	Type	Description
No parameters required.		

The *Fnd_List_OnStartup* routine is called when the database is launched to determine if the lists in the structure file need to be updated from the lists in the data file. Normally this command should be called from the *On Startup* and *On Server Startup* database methods.

If you are using the Foundation Shell, this method is called automatically by the Fnd_Shell component.

Localization Component

Fnd_Loc

The Foundation Localization component allows the Foundation Shell and other Foundation Components to display buttons, labels, and messages in languages other than English. It looks up localized strings, resizes buttons and other form objects to display text of any length, and more.

The Foundation Localization component requires only the Foundation General component if installed in a database other than the Foundation Shell.

How Foundation Handles Localization

4th Dimension allows the developer to use Macintosh string resources to localize menus, button labels, and other objects in the 4D form editor. However, working with Macintosh string resources is not always easy for Windows developers. And string resources are better suited towards static localization (creating one structure file for each language) rather than creating a single structure that is localized on-the-fly.

Instead, Foundation's localization strings are stored as 4D Lists. Each component maintains its own set of localized strings in a set of 4D lists. At least two lists are required for localization. One list of lookup codes, and one list of localized codes for English. Any number of additional lists can also be used to localize the component for another language.

The list of lookup codes and the list of localized strings must remain synchronized. That is, for each item in the lookup code list, there must be a matching item in the localized string list.

The lookup code list generally shares the component’s short name. For example, the lookup list for the Foundation Art component is named “Fnd_Art.” The localized string list starts with the name of the lookup code list, plus an underscore and a two letter language code. “EN” is used for English.

For example, Foundation’s Dialog component contains a lookup code list named “Fnd_Dlg” that looks something like this:

“Fnd_Dlg” list

OK
Cancel
Stop
Save
DontSave

Notice that the entries in the lookup code list do not contain spaces. They are generally kept short, too. So even if a localization list contained a long message, the lookup code would be less than ten characters. It is also important that you do not duplicate a lookup code within a single lookup code list. It is not a problem if two different components use the same lookup code.

The lookup code list has a matching localization string list named “Fnd_Dlg_EN” that looks like this:

“Fnd_Dlg_EN” list

OK
Cancel
Stop
Save
Don’t Save

The Foundation Dialog component also include another list of French localizations. It is named “Fnd_Dlg_FR” and contains these list items:

“Fnd_Dlg_FR” list

OK
Annuler
Arrêter
Enregistrer
Ne pas enregistrer

During runtime, the *Fnd_Loc_GetString* method is used to return localized strings from these lists. Just pass it the name of the lookup code list, plus the specific lookup code you are interested in:

```
$localized_t:=Fnd_Loc_GetString ("Fnd_Dlg";"DontSave")
```

In the example above, “Ne pas enregistrer” is returned if you are using the French version of 4th Dimension, or “Don’t Save” is returned if you are using any other version of 4D. The default is always the English translation if no list is available for the current language.

To make the lists easier to understand during development, comments can be inserted. These are just list items that start with “ ` ” (two spaces, the 4D comment character, and another space) and any text.

Comments like these should be repeated in each list in the same position. For example, we can add a “Button Labels” comment to the Foundation Dialogs lists:

“Fnd_Dlg” list	“Fnd_Dlg_EN” list	“Fnd_Dlg_FR” list
` Button Labels	` Button Labels	` Button Labels
OK	OK	OK
Cancel	Cancel	Annuler
Stop	Stop	Arrêter
Save	Save	Enregistrer
DontSave	Don’t Save	Ne pas enregistrer

The localization strings used by Foundation’s components are stored as Private objects. So you will not see them when using the components, but you will see them when working with the Foundation source code.

Modules, Groups, and Lookup Codes

You do not need to have a lookup code list for each component. In fact, you can have lookup code lists that are not related to a component, or multiple lookup code lists for a component. You could create a lookup code list for buttons and name it “MyButtons,” and another one for messages and call it “MyMessages.”

The combination of the lookup code list, plus the localized strings list are be referred to as a localization module. So “MyButtons,” “MyButtons_EN,” and “MyButtons_FR” are referred to as the “MyButtons” module. The module name is passed as the first parameter to the Fnd_Loc_GetString routine. The second parameter is the lookup code.

When we add comments to the list, the comments break the lists up into groups. In the Foundation Dialogs module described earlier, the “Button Labels” comment creates the group “Button Labels.” Groups are not used at runtime, but they make the lists easier to manage during development. Groups are also used by the Foundation Localization Editor.

Localization Editor

To help you work with localization lists, the Foundation Localization component includes a Localization Editor. This dialog can be displayed by selecting Localization Editor from the Tools menu of the Fnd_Shell menu bar (menu bar #1).

If you have upgraded your database from Foundation 4.0.2 or earlier, you may need to add this menu item to your Fnd_Shell menu using 4D’s Menu Bar editor. Just have the new menu item call the Fnd_Loc_Editor method.

The Localization Editor displays the localization modules for all of the Foundation components installed in your structure file. You can use the editor to modify any of these lists. The editor will actually copy the private Foundation localization lists in your structure file, then edit the copies. This will allow you to easily revert to the original localizations just by deleting the copies. The copies have the same names as the original lists, plus an asterisk.

You can also use the Localization Editor to add your own localization modules to a database. Just click the + button at the bottom of the Modules list. Give the new module a unique name, and then create groups and lookup codes for your project.

Currently Foundation's Localization Editor supports only English and French localizations. However, the Localization component's methods can handle any number of languages. You will just need to create the lists for other languages manually, using the 4D List Editor. The *Fnd_Loc_DuplicateList* method can be helpful when creating other localization lists.

Important Notes

Keep in mind that the Localization Editor is intended to be used only as a development tool. It does not contain the extensive error checking required for use by end users.

A lookup code cannot be duplicated within a module, even if it is in a separate group. Having duplicated lookup codes in different modules is not a problem.

Foundation's Localization Editor cannot be used when the 4D List Editor window is open, or if there are other users connected to 4D Server. This is to prevent potential list corruption problems.

You can add an existing module name (it is defined in the 4D lists editor) to the Localization Editor without losing the existing list. The Localization Editor will copy the contents of the existing list into the new localization list it creates.

The list editor will always add an asterisk to your list names when saving them. You can leave them with the asterisk, or remove the asterisk. If you remove the asterisk you will have a backup copy in case something goes wrong with the list editor.

Any list beginning with "Fnd_" and ending with an asterisk can be safely deleted. It is a copy of a Foundation list. If the asterisk version is deleted, the component's Private list will be used.

Although the editor will allow you to add and remove items from the Foundation localization lists, this is generally considered a bad idea.

Foundation's *Fnd_Loc_GetString* method will always use lists with asterisks if they are available. Because of this, any Foundation lists you modify with the list editor will override Foundation's private lists. If a list with an asterisk is available, the routine will not look for additional information from the list name without the asterisk.

It is safe to delete the "Fnd_Loc_EditorModules*" list also. It will be recreated the next time the Localization Editor is used. This list is used only by the Localization Editor — it is not used at runtime.

Localized lists should not be passed to the *Fnd_List_AddToListEditor* method from the *Fnd_Hook_List_SetEditableLists* hook, since this would allow users to unsynchronize the lists.

Here is the list of routines in Foundation's Localization component:

- Fnd_Loc_DuplicateList
- Fnd_Loc_Editor
- Fnd_Loc_FixButtonWidths
- Fnd_Loc_GetString
- Fnd_Loc_Info
- Fnd_Loc_LanguageCode

Language Reference

Here are the routines in Foundation's Localization component:

Fnd_Loc_DuplicateList

Parameter	Type	Description
No parameters required.		

This routine can be used to duplicate a 4D list. It is designed to be called from Foundation's Fnd_Shell menu by the developer to assist in creating new localizations. It is not intended to be called procedurally.

To call this method, select Duplicate a 4D List... from Foundation's Tools menu. A dialog will ask you for the name of the list to duplicate. Enter the name of an existing 4D list and click OK.

In the next dialog, enter in the name of the new list to create and click OK. The first list will be duplicated and given this name.

Fnd_Loc_Editor

Parameter	Type	Description
No parameters required.		

Displays the Localization List Editor window in a new process. See the beginning of this chapter for more information.

Fnd_Loc_FixButtonWidths

Fnd_Loc_FixButtonWidths (->button1; ->button2{; ->buttonN})

Parameter	Type	Description
button1..buttonN	Pointer	Pointers to two or more buttons

Adjusts all of the button widths of two or more vertically aligned buttons so they all have the same left and right coordinates of the widest button of the group. Designed to be called after calling *Fnd_Loc_DuplicateList* on each of the buttons in the group.

For example, imagine we have three buttons as shown here:

If we use the *Fnd_Loc_DuplicateList* method to change the button labels to French, the size of the buttons will all be increased as needed. But after the call, each button will have a different length.

So we pass pointers to each of these buttons to the *Fnd_Loc_FixLabelWidths*, and each button is set to the length of the longest button:

```
Fnd_Loc_FixButtonWidths (->bNew;->bView;->bDelete)
```

Fnd_Loc_FixLabelWidths

Fnd_Loc_FixLabelWidths (->form object; ->enterable object{; ..repeat..})

Parameter	Type	Description
$1, $3, $5..$N	Pointer	The label to adjust
$2, $4, $6..$N+1	Pointer	The field to its right

One of the problems with localizing field labels is that if you make the label large enough for one language, it may appear too large for other languages. And even if you resize the label object, you must also reposition the field next to it. However, if you do this for each label/field combination individually, you will end up with fields that are no longer aligned.

One solution is to use multiple pages or multiple forms — one for each language. But this can be a maintenance problem for complex layouts. Another solution is to use the *Fnd_Loc_FixLabelWidths* method to resize multiple labels and fields in a single call.

Pointers to 4D form objects are passed to this routine in sets of two. One for the label, and one for the field. This routine will move the right side of the label so that the entire label text fits. It will then adjust the left side of the field the same amount, so that the space between the label and the field remains unchanged.

If multiple sets of label and field pointers are passed, the widest label is used for adjusting all of the objects.

For example, imagine we have this English set of fields on a form:

We start by replacing the static text labels with variables, so we can pass pointers to these objects to the *Fnd_Loc_FixLabelWidths* method. Then we localize the labels using Foundation's *Fnd_Loc_GetString* function.

Now the labels are correct, but the colon has been truncated on some of the labels. So we pass the three labels on the left, along with their associated fields, to the *Fnd_Loc_FixLabelWidths* method in a single call:

```
Fnd_Loc_FixLabelWidths (->vNumberLabel;->[Products]Number;
          ->vNameLabel;->[Products]Name;->vDescLabel;->[Products]Desc)
```

Now the labels on the left are properly displayed. The width of the label objects has been increased slightly, and the width of the corresponding fields has been reduced by the same amount. The amount of the change is based on the content of the labels.

Now we just need to apply the fix to the List Price label and field:

```
Fnd_Loc_FixLabelWidths (->vPriceLabel;->[Products]ListPrice)
```

See the Product Sales example database to see this command in action.

Fnd_Loc_GetString

Fnd_Loc_GetString (module; lookup code{; param 1{; param 2...}}) ➜ Text

Parameter	Type	Description
module	Text	The module name
lookup code	Text	The lookup code
param 1..param 2	Text	Replacement parameters (optional)
Function Result	Text	The localized string

This routine returns a localized string based on the internal string or label. Foundation's Language Code is used to determine which language string to return. See the *Fnd_Loc_LanguageCode* method for more information.

Foundation stores localization strings as 4D lists. These lists can be edited directly in the 4D List Editor, or using Foundation's Localization Editor. See the detailed description at the beginning of this chapter.

To use this method, pass it the module name and the lookup code:

```
$localized_t:=Fnd_Loc_GetString ("MyModule";"ProductNameLable")
```

You can optionally pass additional text parameters to insert into the string. For example, you may want to localize a string like "You have 10 days remaining" where the number of days is calculated, rather than fixed. To do this, create a localized string with "<<1>>" in it. This will be replaced by the first optional parameter to this call.

In this example, imagine we have a localized string named "DaysRemaining" with this content:

"You have <<1>> days remaining."

Just pass the number of days (as a string) as the third parameter to Fnd_Loc_GetString:

```
$days:=String(vDays)
$localized_t:=Fnd_Loc_GetString ("MyModule";"DaysRemaining";$days)
```

You can do this replacement for any number of optional parameters. For the second optional parameter use "<<2>>," then "<<3>>" for the third, etc.

Fnd_Loc_Info

Fnd_Loc_Info (info requested) ➜ Text

Parameter	Type	Description
info requested	Text	Info desired
Function Result	Text	Response

Returns the requested information about the component.

```
$language_t:=Fnd_Loc_Info ("language")
```

The Fnd_Loc_Info method will respond to these requests:

Request	Response	Example
name	The component's full name	Foundation Localization
version	The component's version number	4.0.3 beta 1
language	The currently selected language code	EN

This routine can also be called using the *Fnd_Gen_ComponentInfo* method without first testing to see if the component is installed:

```
$language_t:=Fnd_Gen_ComponentInfo ("Fnd_Loc";"language")
```

See the *Fnd_Gen_ComponentInfo* method for more information.

Fnd_Loc_LanguageCode

Fnd_Loc_LanguageCode ({code}) ➜ Text

Parameter	Type	Description
code	Text	The language code to use (optional)
Function Result	Text	The current language code

If no parameters are passed, this routine returns the current language code that Foundation is using for localization.

```
$languageCode:=Fnd_Loc_LanguageCode
```

If a language code is passed to this routine, it is set as the new localization code.

```
Fnd_Loc_LanguageCode ("FR")
```

If you are working with a database that may not have the Foundation Language component installed, you can just call the *Fnd_Gen_ComponentInfo* routine like this:

```
$languageCode:=Fnd_Gen_ComponentInfo ("Fnd_Loc";"language")
```

The Foundation General component will then test to see if the Localization component is installed, and if so, return the current language code. If the Localization component is not installed, “Component Not Available” will be returned.

Log Component

Fnd_Log

The Log component provides logging capabilites to track any events in your database. You can call these commands from anywhere in your database. The log file name will consist of the name of your Structure file.log on Mac OS X and Structure file.txt on Windows. It will be created in the Get 4D folder on Windows and Logs folder inside your 4D preferences folder on Mac OS X.

Language Reference

Here are the routines in Foundation's Log component:

Fnd_Log_AddEntry
Fnd_Log_Enable
Fnd_Log_Info

Fnd_Log_AddEntry

Fnd_Log_AddEntry (text1{; text2..textN})

Parameter	Type	Description
text1	Text	Text to be placed in the log

The *Fnd_Log_AddEntry* routine places the text parameters into an external text file. It is safe to call this routine even if we are not currently logging activity.

```
Fnd_Log_AddEntry("An error has occurred in method: "+ Current method name+".")
```

Fnd_Log_Enable

Fnd_Log_Enable ({enable?}) ➜ Boolean

Parameter	Type	Description
enable	Boolean	True to enable logging (optional)
Function result	Boolean	True if logging is enabled

This function allows the developer to turn logging on or off. This function returns true if logging is currently on.

```
$isLoggingOn_b:=Fnd_Log_Enable ` Also turns logging on
```

Fnd_Log_Info

Fnd_Log_Info (info requested) ➜ Text

Parameter	Type	Description
info requested	Text	Info desired
Function Result	Text	Response

Returns the requested information about the component.

```
$language_t:=Fnd_Log_Info (“language”)
```

The Fnd_Log_Info method will respond to these requests:

Request	Response	Example
name	The component’s full name	Foundation Log
version	The component’s version number	4.0.3 beta 1
language	The currently selected language code	EN

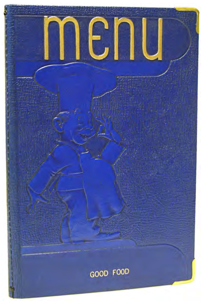
This routine can also be called using the *Fnd_Gen_ComponentInfo* method without first testing to see if the component is installed:

```
$language_t:=Fnd_Gen_ComponentInfo ("Fnd_Log";"language")
```

See the *Fnd_Gen_ComponentInfo* method for more information.

Menus Component

(Fnd_Menu)



The Menu component has been designed specifically for use with the Foundation Shell. It is not intended to be used in other database projects. However, there are some useful commands you may wish to use when working with the Foundation Shell.

Installation

The Menus component is designed for use only with the Foundation Shell. Therefore, it requires version 4.2 or later of the Fnd_Shell component.

Updating the Component

The component's "Fnd_Menu" menu bar has public access, so that you can modify it to meet your needs. You can safely add and delete menu items using the 4D Menu Editor. But be sure to backup your structure before upgrading this component to ensure you can restore your changes, if necessary.

Language Reference

The following methods are simply marked as protected so they can be called from the Foundation menu bar (the menu named “Fnd_Menu”). Although they are not really intended to be called directly, you can call them to duplicate the behavior of selecting a menu item.

Fnd_Menu_Administration
Fnd_Menu_Delete
Fnd_Menu_Find
Fnd_Menu_ModifyRecords
Fnd_Menu_NavPalette
Fnd_Menu_NewRecord
Fnd_Menu_OmitSubset
Fnd_Menu_OpenTable
Fnd_Menu_Preferences
Fnd_Menu_Print
Fnd_Menu_QueryEditor
Fnd_Menu_Quit
Fnd_Menu_ShowAll
Fnd_Menu_ShowSubset
Fnd_Menu_Sort
Fnd_Menu_SortOrderEditor
Fnd_Menu_SpecialFunctions
Fnd_Menu_WindowItem

Here are the more useful (and documented) routines in Foundation’s Menus component:

Fnd_Menu_DisableAll
Fnd_Menu_MenuBar
Fnd_Menu_Window_Add
Fnd_Menu_Info
Fnd_Menu_MenuBarName
Fnd_Menu_Window_Remove

Fnd_Menu_DisableAll

Fnd_Menu_DisableAll

Parameter	Type	Description
No parameters required.		

This routine temporarily disables all of the currently displayed menu items. Call it before displaying a dialog or window which will not respond to menu selections. The menus will be automatically restored the next time the menu bar is updated. To ensure that the menus remain disabled when the window is displayed, call this method from the form’s On Activate event after calling *Fnd_Gen_FormMethod*.

```
Fnd_Gen_FormMethod
Case of
  : (Form event=On Activate )
    Fnd_Menu_DisableAll
End case
```

Fnd_Menu_Info

Fnd_Menu_Info (info requested) ➔ Text

Parameter	Type	Description
info requested	Text	Info desired

Function Result	Text	Response

This function returns the requested information about the Menu component.

```
$version_t:=Fnd_Menu_Info ("version")
```

The *Fnd_Menu_Info* method will respond to these requests:

Request	Response	Example
name	The component's full name	Foundation Menus
version	The component's version number	4.1.2

See the *Fnd_Gen_ComponentInfo* method for more information.

Fnd_Menu_MenuBar

Fnd_Menu_MenuBar

Parameter	Type	Description
No parameters required.		

This routine installs and updates Foundation's menu bar. In addition to calling the **MENU BAR** command, this routine will localize the menus (using the Localization component, if available), enable and disable the appropriate menu items, and update the **Window** menu.

If you create a component that requires the Foundation Menus component, you can call this method directly. Otherwise, it is best just to call *Fnd_Gen_MenuBar*, which will call this method if it is available.

Fnd_Menu_MenuBarName

Fnd_Menu_MenuBarName ({menu name}) ➔ Text

Parameter	Type	Description
menu name	Text	Menu bar name (optional)
Function result	Text	Menu bar name to use next

Fnd_Menu_MenuBarName allows the developer to specify the menu bar to display the next time the menu bar is displayed by a Foundation component. If a blank name is specified, the default Foundation menu bar name ("Fnd_Menu") is restored. If no parameter is passed, nothing is changed.

This routine also returns the name of the menu bar name that will be used the next time the *Fnd_Menu_MenuBar* command is called.

In the original Foundation 4.0 release, this method was named *Fnd_Menu_SetMenuBarName*.

Fnd_Menu_Window_Add

Fnd_Menu_Window_Add ({item text})

Parameter	Type	Description
item text	Text	Item text to add to the menu (optional)

Adds the specified title to the **Window** menu. If no title is specified, the window title of the current process is used. If the current process is already represented in the **Window** menu, the menu item text is updated.

```
Fnd_Wnd_Title ("Invoice Designer")  ` Fnd_Menu_Window_Add will use this.
Fnd_Wnd_OpenFormWindow (->[Invoices];"InvoiceDesignerForm")
Fnd_Menu_Window_Add
DIALOG([Invoices];"InvoiceDesignerForm")
Fnd_Menu_Window_Remove
CLOSE WINDOW
```

Call *Fnd_Menu_Window_Remove* just before closing the window to remove the item from the **Window** menu.

Fnd_Menu_Window_Remove

Fnd_Menu_Window_Remove ({process number})

Parameter	Type	Description
process number	Longint	Process number to remove

The *Fnd_Menu_Window_Remove* routine removes the specified process from the **Window** menu. If the process number is not specified, the current process is removed from the menu bar.

It is important to call this method before calling 4D's **CLOSE WINDOW** command. Otherwise the window's name may not get removed from the **Window** menu.

See the *Fnd_Menu_Window_Add* command for an example.

Message Component

Fnd_Msg

The Message component provides messaging between processes. You can call these commands from anywhere in your database.

Language Reference

Here are the routines in Foundation's Message component:

Fnd_Msg_Broadcast
Fnd_Msg_Get
Fnd_Msg_GetParameter
Fnd_Msg_Info
Fnd_Msg_PackParameters
Fnd_Msg_QuitBackgroundProcess
Fnd_Msg_Send

Fnd_Msg_Broadcast

Fnd_Msg_Broadcast (process name; message)

Parameter	Type	Description
process name	Text	Process name to send message
message	Text	Message to send to process

The *Fnd_Msg_Broadcast* routine sends the message out to all processes with the specified name. The process name can contain wildcard characters. Does not broadcast to 4D created processes.

```
Fnd_Msg_Broadcast (processname; "Hello, World")
```

Fnd_Msg_Get

Fnd_Msg_Get ({->blob}) ➔ Text

Parameter	Type	Description
blob	Pointer	Pointer to blob (optional)
Function result	Text	The message

This function returns the next message queued up for the current process.

```
$processmessage_t:=Fnd_Msg_Get (->messsage_blob)
```

Fnd_Msg_GetParameter

Fnd_Msg_GetParameter (parameters string; num) ➔ Text

Parameter	Type	Description
parameters string	Text	String of parameters
num	Longint	Parameter number to get
Function result	Text	Parameter value

This function returns the parameter in position "num" out of the message. See *Fnd_Msg_PackParameters* to find out more about how to pack the parameters into your message.

```
$State_t:=Fnd_Msg_GetParameter ("CityState";2)
```

Fnd_Msg_Info

Fnd_Msg_Info (info requested) ➜ Text

Parameter	Type	Description
info requested	Text	Info desired
Function Result	Text	Response

Returns the requested information about the component.

```
$language_t:=Fnd_Msg_Info (“language”)
```

The Fnd_Msg_Info method will respond to these requests:

Request	Response	Example
name	The component’s full name	Foundation Message
version	The component’s version number	4.0.3 beta 1
language	The currently selected language code	EN

This routine can also be called using the *Fnd_Gen_ComponentInfo* method without first testing to see if the component is installed:

```
$language_t:=Fnd_Gen_ComponentInfo ("Fnd_Msg";"language")
```

See the *Fnd_Gen_ComponentInfo* method for more information.

Fnd_Msg_PackParameters

Fnd_Msg_PackParameters (msg; param 1 {... param N}) ➜ Text

Parameter	Type	Description
msg	Text	Message to append parameter
param 1 - n	Text	Parameters to add to the message
Function result	Text	Delimited parameters

This function returns the delimited parameters. This function adds the text parameters to the message using the same delimiter expected by the *Fnd_Msg_GetParameter* routine.

```
$delimitedparams_t:=Fnd_Msg_PackParameters ("Location_msg";"City";"State")
```

Fnd_Msg_QuitBackgroundProcess

Fnd_Msg_QuitBackgroundProcess

This method ends the message loop background process. If there are messages still waiting to be delivered, this process waits until a timeout period is reached so hopefully the messages will be delivered.

```
Fnd_Msg_QuitBackgroundProcess
```

Fnd_Msg_Send

Fnd_Msg_Send (target process; message; {->blob})

Parameter	Type	Description
target process	Longint	Process number to send the message
message	Text	Message to send
blob	Pointer	Pointer to a blob (optional)

This method sends a message to another process.

```
Fnd_Msg_Send (Process number(◇YourProcessName);"Hello, World")
```

Navigation Component

Fnd_Nav

The Foundation Navigation component allows you to display a floating palette of buttons that the user can use to perform actions. By default, the Navigation Palette will display buttons for each visible table in the structure file. Clicking one of the buttons will call Foundation's *Fnd_IO_DisplayTable* method to display the selected table's output form.

You can optionally change the buttons that are displayed in the palette, and add additional buttons to perform other functions. Once you call *Fnd_Nav_AddButtonMethod* or *Fnd_Nav_AddButtonTable* to add a button to the table, it will be added as the first button. The default buttons will no longer be displayed. Each additional call to *Fnd_Nav_AddButtonMethod* or *Fnd_Nav_AddButtonTable* will add another button to the palette.

When the palette window is closed, the settings are reset. You must once again add the buttons the next time the palette is displayed. In the Foundation Shell, this should be done in the *Fnd_Hook_Shell_NavigationPalette* hook.

You can also reset the palette by deleting all of the buttons with the *Fnd_Nav_Delete* method. This routine allows you to rebuild the palette on-the-fly. The palette window will resize if necessary. This command can also be used to procedurally close the palette window.

The Navigation Palette is limited to 40 buttons. Attempting to add more than 40 buttons will have no effect.

This component requires the Foundation General (Fnd_Gen) component. This routine will use the Foundation Preferences, Virtual Structure, and IO components if they are available.

Language Reference

Here are the routines in Foundation's Navigation component:

Fnd_Nav_AddButtonMethod
Fnd_Nav_AddButtonTable
Fnd_Nav Delete
Fnd_Nav_Display
Fnd_Nav_Info

Fnd_Nav_AddButtonMethod

Fnd_Nav_AddButtonMethod (method; label{; foreground color; background color}})

Parameter	Type	Description
method	Text	Method name to execute
label	Text	Button label
foreground color	Longint	Text color (optional)
background color	Longint	Background color (optional)

This routine adds a new button to the palette and assigns the method as its action. When the button is clicked, the first parameter, method, will be executed.

```
Fnd_Nav_AddButtonMethod ("MyCoolMethod"; "Run Cool Method")
```

You can include parameters or any other executable code as the first parameter. For example, the following example will pass a text and a numeric parameter to MyCoolMethod:

```
Fnd_Nav_AddButtonMethod ("MyCoolMethod(\"one\";2)"; "Run Cool Method")
```

If a button with an identical label already exists, it is updated with the new method name.

Buttons are always added to the palette after existing palette buttons. To add a button in any other position, just use the *Fnd_Nav_Delete* to delete all of the existing palette buttons, then recreate the palette using *Fnd_Nav_AddButtonMethod* and *Fnd_Nav_AddButtonTable*.

```
Fnd_Nav_Delete    ` Remove all of the buttons.
Fnd_Nav_AddButtonMethod ("MyCoolMethod";"Run Cool Method")
Fnd_Nav_AddButtonTable (->[Invoices])
Fnd_Nav_AddButtonTable (->[People];"Customers")
```

You can optionally specify the text color and background color to use for the new button. See 4D's **SET RGB COLORS** command documentation for more information about specifying colors. Pass the

4D constant Default foreground color and Default background color to restore the palette's default colors.

```
  ` Add a button with a light green background color.
Fnd_Nav_AddButtonMethod ("MyMethod";"Run My Method";Default foreground
color ;0xE2FFDF)
```

Fnd_Nav_AddButtonTable

Fnd_Nav_AddButtonTable (->table{; label{; foreground color; background color}})

Parameter	Type	Description
table	Pointer	Table
label	Text	Button label (optional)
foreground color	Longint	Text color (optional)
background color	Longint	Background color (optional)

This method adds a new button to the palette and assigns the specified table to it.

```
Fnd_Nav_AddButtonTable (->[Invoices])
Fnd_Nav_AddButtonTable (->[People];"Customers")
```

This routine calls Foundation's IO component to display the table. If the IO component is not available, an error will occur when the button is clicked.

If the Foundation Virtual Structure component is available, it will be used to determine the table name if no label is specified. Otherwise the actual table name will be used.

Buttons are always added to the palette after existing palette buttons. To add a button in any other position, just use the *Fnd_Nav_Delete* to delete all of the existing palette buttons, then recreate the palette using *Fnd_Nav_AddButtonMethod* and *Fnd_Nav_AddButtonTable*. See the *Fnd_Nav_AddButtonMethod* routine for an example.

You can optionally specify the text color and background color to use for the new button. See 4D's **SET RGB COLORS** command documentation for more information about specifying colors. Pass the 4D constant Default foreground color and Default background color to restore the palette's default colors.

```
  ` Add a button with blue text and a light blue background color.
Fnd_Nav_AddButtonTable (->[Invoices];"";0x00FF;0x00C6D4FC)
```

Fnd_Nav_Delete

Fnd_Nav_Delete ({label})

Parameter	Type	Description
label	Text	Button label to delete (optional)

Fnd_Nav_Delete deletes the button with the specified label from the navigation palette. If no parameter is passed, all buttons are deleted.

If all of the buttons are deleted from the palette, and no new buttons are immediately added, the palette window will close.

Fnd_Nav_Display

Fnd_Nav_Display

Parameter	Type	Description
No parameters required.		

Fnd_Nav_Display displays the navigation palette window. If *Fnd_Nav_AddButtonMethod* or *Fnd_Nav_AddButtonTable* have not been previously called, the palette will contain a button for each visible table in the database. However, these buttons will work only if the Foundation IO component is available.

```
Fnd_Nav_AddButtonMethod ("MyCoolMethod";"Run Cool Method")
Fnd_Nav_AddButtonTable (->[Invoices])
Fnd_Nav_AddButtonTable (->[People];"Customers";Default foreground color ;0x00E2FFDF)
Fnd_Nav_Display
```

Fnd_Nav_Info

Fnd_Nav_Info (info requested) ➜ Text

Parameter	Type	Description
info requested	Text	Info desired
Function Result	Text	Response

Returns the requested information about the component.

```
$language_t:=Fnd_Nav_Info ("language")
```

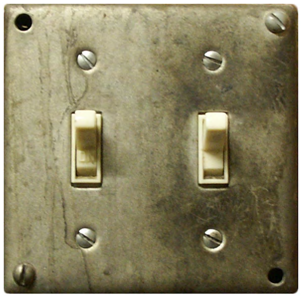
The Fnd_Nav_Info method will respond to these requests:

Request	Response	Example
name	The component's full name	Foundation Navigation
version	The component's version number	4.0.3 beta 1
language	The currently selected language code	EN

This routine can also be called using the *Fnd_Gen_ComponentInfo* method without first testing to see if the component is installed:

```
$language_t:=Fnd_Gen_ComponentInfo ("Fnd_Nav";"language")
```

See the *Fnd_Gen_ComponentInfo* method for more information.



Preferences Component

Fnd_Pref

The Foundation Preferences component provides a very easy way to manage user preferences. With this component, you can easily store strings, numbers, Boolean values, and even window coordinates.

Using this component, a preference value can be stored either locally in a preferences file on the user's computer, or in the 4D database in the [Fnd_Pref] table. Any combination of local and server preferences can be used. Additionally, this component lets you create any number of shared server preferences that can be used by all database users.

Preference Scope

The 4.2 update adds the ability to define any preference value as either a local, server, or shared preference. The methods for getting and setting each of these preference types is identical, but differs in the new "scope" parameter that can be passed to the methods when setting a preference value.

Preferences are set using one of this component's "set" routines. Just pass a preference name and value:

```
Fnd_Pref_SetText ("My Favorite Color";"green")
```

This stores the preference in the user's external preference file. This call can be modified to instead save the preference as a record in the database by adding the Fnd_Pref_Server constant as a third parameter:

```
Fnd_Pref_SetText ("My Favorite Color";"green";Fnd_Pref_Server )
```

The scope parameter is not used when getting a preference value. So the function to get the user's favorite color is the same regardless of where it is stored:

```
$color_t:=Fnd_Pref_GetText ("My Favorite Color")
```

However, since you can pass a default value to use when getting a preference, you can also now pass the scope for the default value. If the preference does not already exist, it will be created using the default value and scope.

```
$color_t:=Fnd_Pref_GetText ("My Favorite Color";"White";Fnd_Pref_Server)
```

Although it would be possible to store a preference with the same name both locally and in the data file, this should be avoided, since there is no way to control which preference (local or server) would be returned when getting the value.

Local Preferences

Local preferences are stored in an external file on the user's hard drive. The file is stored in the user's preferences folder in an XML format. Local preference values cannot be seen or manipulated by any other user in a multiuser system.

Local preferences are ideal for machine specific settings, such as window positions and default directory paths.

This is the default preferences setting type if no scope parameter is provided when calling the "set" preference routines. Earlier versions of the Preferences component offered this type of preference setting.

The preferences are stored in an XML file in the user's directory:

OS	Directory
Mac OS 9:	Hard Disk:System Folder:Preferences:DatabaseName Prefs
Mac OS X:	~/Library/Preferences/DatabaseName.plist
Windows:	C:\Documents and Settings\User Name\Application Data\4D\DatabaseName.xml

The "DatabaseName" in the paths shown above is based on the structure file name, not on the name you assign the database in Foundation's *Fnd_Hook_Shell_Setup* method.

The file is stored as a Mac OS X preference file, regardless of the platform in use. A simple Foundation preference file might look something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/
DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
  <dict>
    <key>Fnd_Nav: Open at Startup</key>
    <true/>
    <key>Fnd_Nav: Navigation Palette Window Position</key>
    <dict>
      <key>left</key>
      <integer>1356</integer>
      <key>top</key>
      <integer>34</integer>
      <key>right</key>
      <integer>1440</integer>
      <key>bottom</key>
      <integer>53</integer>
    </dict>
  </dict>
</plist>
```

Server Preferences

Like local preferences, server preferences are unique to the current user, but the value is stored in the 4D data file. This allows a user to use the same preference values when working at different workstations. Server preferences for any user can be procedurally controlled from another workstation in a client/server environment.

By default, server preferences are stored using the current user's name from the 4D password system. The *Fnd_Pref_UserName* method can be used to change the current user name for administration purposes or if you are not using 4D's password system.

Server preferences are ideal for setting user or data specific settings, such as the user's e-mail address or e-mail message signature.

Shared Preferences

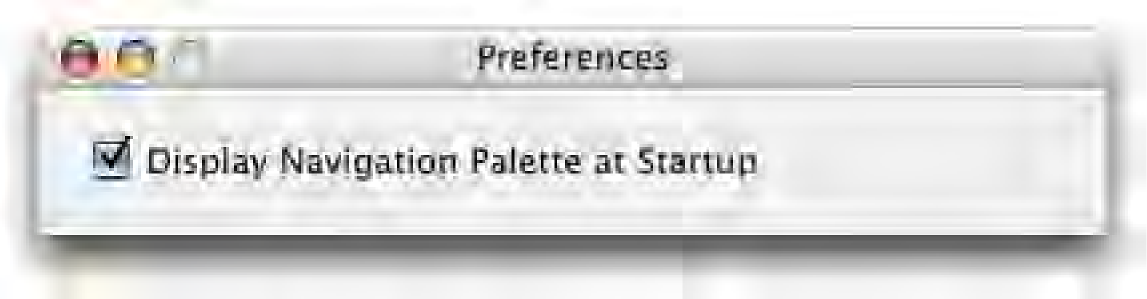
Shared preferences are also stored in the data file, but can be set and accessed by any user. The value stored is identical for all users, and can also be used by triggers, web processes, and stored procedures on the server.

To set a shared preference, use the Fnd_Pref_Shared constant when setting the preference value:

```
Fnd_Pref_SetText ("SMTP Mail Server";"smtp.ourcompany.com";Fnd_Pref_Shared )
```

The Preferences Form

This component includes a form that is used to display the Preferences window.



☑ Display Navigation Palette at Startup

Unlike most forms, this form is Public rather than Private, allowing you to modify it. By default, it has just a single option: Display Navigation Palette at Startup.

You can add additional objects to this form and wire it into the Preferences component using the commands below. Each preference setting requires a name and a value. The name can be just about anything you want, as long as each name is unique.

Installation

The Prefs component stores information in a database table. 4D v11 components do not allow tables to be included in a component.

Therefore, the Foundation components do not include any 4D tables. Instead, you will need to add any required tables to your structure before installing the component. Internally the components create pointers to the structure's tables and fields and then uses these throughout the code.

Create the [Fnd_Pref] Table

To use the Foundation Preferences component, you will first need to add a table to your structure. You can simply copy the [Fnd_Pref] table from the Product Sales.4DB sample file, or you can create it manually in 4D (this is a good opportunity to reuse an unused or deleted table). The table must be named "Fnd_Pref" and must contain the fields listed below. The order of the fields is not important, and it is okay if the table contains other unused fields (in case you are reusing a table). However, the field names and types must be set up exactly as shown below.

Field Name	Type	Attributes
[Fnd_Pref]ID	Long Integer	
[Fnd_Pref]Owner	Alpha 80	Indexed
[Fnd_Pref]Name	Alpha 80	Indexed
[Fnd_Pref]Type	Integer	
[Fnd_Pref]Value	Text	

Install the Component

The Foundation Preferences component requires the following components (shown with minimum required version numbers). It also requires version 4.2 or later of the Foundation Extras plugin.

Component	Minimum Version
Fnd_Gen	4.2

Updating the Component

If you later upgrade (or reinstall) the component, you will be asked if you want to update the public Fnd_Pref_Preferences form. Click **Yes** if you have not customized this form for your application, or if you wish to revert to the default layout. Otherwise click **No** to preserve any changes you may have made.

Language Reference

Here are the routines in Foundation's Preferences component:

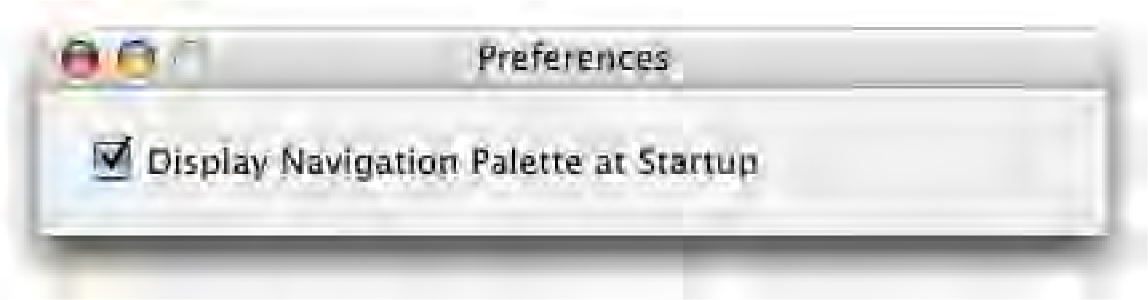
Fnd_Pref_Display
Fnd_Pref_GetBoolean
Fnd_Pref_GetLongInt
Fnd_Pref_GetReal
Fnd_Pref_GetText
Fnd_Pref_GetWindow
Fnd_Pref_Info
Fnd_Pref_SetBoolean
Fnd_Pref_SetLongInt
Fnd_Pref_SetReal
Fnd_Pref_SetText
Fnd_Pref_SetWindow
Fnd_Pref_UserName

Fnd_Pref_Display

Fnd_Pref_Display

Parameter	Type	Description
No parameters required.		

This method displays the Preferences window in a new process.



☑ Display Navigation Palette at Startup

You can add additional options to this window by modifying the component's Fnd_Pref_Preferences public form. Just add similar code to the existing code that gets and sets the Navigation Palette option.

Fnd_Pref_GetBoolean

Fnd_Pref_GetBoolean (name{; default {; scope}}) ➔ Boolean

Parameter	Type	Description
name	Text	Name of the preference item
default	Boolean	Default value (optional)
scope	Longint	The scope if the default is used (optional)
Function result	Boolean	The saved value

Returns the Boolean value of the preference with the specified name:

```
$displayToolbar_b:=Fnd_Pref_GetBoolean ("Toolbar")
```

If an item with the specified name is not found, **False** is returned. To specify a different default value, pass it as the second parameter:

```
$displayToolbar_b:=Fnd_Pref_GetBoolean ("Toolbar";True)
```

If the preference value already exists, the second parameter will be ignored. If it does not exist, it will be created and set to the value of the second parameter.

If no scope parameter is specified, the value will be saved as a local preference. A third parameter can be used to specify the preference's scope:

```
$displayToolbar_b:=Fnd_Pref_GetBoolean ("Toolbar";True;Fnd_Pref_Server )
```

The scope value will be used only if the parameter does not already exist.

Fnd_Pref_GetLongInt

Fnd_Pref_GetLongInt (name{; default {; scope}}) ➔ Longint

Parameter	Type	Description
name	Text	Name of the preference item
default	Longint	Default value (optional)
scope	Longint	The scope if the default is used (optional)
Function result	Longint	The saved value

Returns the long integer value of the preference with the specified name:

```
$timeout_i:=Fnd_Pref_GetLongint ("TimeoutInSeconds")
```

If an item with the specified name is not found, 0 is returned. To specify a different default value, pass it as the second parameter:

```
$timeout_i:=Fnd_Pref_GetLongint ("TimeoutInSeconds";30)
```

If the preference value already exists, the second parameter will be ignored. If it does not exist, it will be created and set to the value of the second parameter.

If no scope parameter is specified, the value will be saved as a local preference. A third parameter can be used to specify the preference's scope:

```
$timeout_i:=Fnd_Pref_GetLongint ("TimeoutInSeconds";30;Fnd_Pref_Server )
```

The scope value will be used only if the parameter does not already exist.

Fnd_Pref_GetReal

Fnd_Pref_GetReal (name{; default {; scope}}) ➔ Longint

Parameter	Type	Description
name	Text	Name of the preference item
default	Real	Default value (optional)
scope	Longint	The scope if the default is used (optional)
Function result	Real	The saved value

Returns the real number value of the preference with the specified name:

```
$rate_r:=Fnd_Pref_GetReal ("ExchangeRate")
```

If an item with the specified name is not found, 0 is returned. To specify a different default value, pass it as the second parameter:

```
$rate_r:=Fnd_Pref_GetReal ("ExchangeRate";1.35)
```

If the preference value already exists, the second parameter will be ignored. If it does not exist, it will be created and set to the value of the second parameter.

If no scope parameter is specified, the value will be saved as a local preference. A third parameter can be used to specify the preference's scope:

```
$rate_r:=Fnd_Pref_GetReal ("ExchangeRate";1.35;Fnd_Pref_Shared )
```

The scope value will be used only if the parameter does not already exist.

Fnd_Pref_GetText

Fnd_Pref_GetText (name{; default {; scope}}) ➔ Longint

Parameter	Type	Description
name	Text	Name of the preference item
default	Text	Default value (optional)
scope	Longint	The scope if the default is used (optional)
Function result	Text	The saved value

Returns the text value of the preference with the specified name:

```
$email_t:=Fnd_Pref_GetText ("EmailAddress")
```

If an item with the specified name is not found, 0 is returned. To specify a different default value, pass it as the second parameter:

```
$email_t:=Fnd_Pref_GetText ("EmailAddress";"you@OurCompany.com")
```

If the preference value already exists, the second parameter will be ignored. If it does not exist, it will be created and set to the value of the second parameter.

If no scope parameter is specified, the value will be saved as a local preference. A third parameter can be used to specify the preference's scope:

```
$email_t:=Fnd_Pref_GetText ("EmailAddress";"you@OurCompany.com";Fnd_Pref_Server )
```

The scope value will be used only if the parameter does not already exist.

Fnd_Pref_GetWindow

Fnd_Pref_GetWindow (name; ->left; ->top; ->right; ->bottom)

Parameter	Type	Description
name	Text	Name of the preference item
left	Pointer	Left coordinate
top	Pointer	Top coordinate
right	Pointer	Right coordinate
bottom	Pointer	Bottom coordinate

Use this method to retrieve the coordinates (left, top, right, and bottom) of a window position that has been stored with Foundation's *Fnd_Pref_SetWindow* method. Pass pointers to four variables, and this routine will modify the values of the variables.

If the window position has not been previously saved, the variables will be left untouched, so you can set them to default values before calling this method. For example:

```
vLeft:=10  ` Set the default position first.
vTop:=40
vRight:=510
vBottom:=340
Fnd_Pref_GetWindow ("My Window";->vLeft;->vTop;->vRight;->vBottom)
$winRef:=Open window(vLeft;vTop;vRight;vBottom;Movable dialog box )
Fnd_Pref_SetWindow ("My Window";->vLeft;->vTop;->vRight;->vBottom)
```

Window coordinates are always stored locally. There is no option to store this information as a server or shared preference.

Fnd_Pref_Info

Fnd_Pref_Info (info requested) ➔ Text

Parameter	Type	Description
info requested	Text	Info desired
Function Result	Text	Response

Returns the requested information about the component.

```
$version_t:=Fnd_Pref_Info ("version")
```

The *Fnd_Pref_Info* method will respond to these requests:

Request	Response	Example
name	The component's full name	Foundation Preferences
version	The component's version number	4.2

This routine can also be called using the *Fnd_Gen_ComponentInfo* method without first testing to see if the component is installed:

```
$version_t:=Fnd_Gen_ComponentInfo ("Fnd_Pref";"version")
```

See the *Fnd_Gen_ComponentInfo* method for more information.

Fnd_Pref_SetBoolean

Fnd_Pref_SetBoolean (name; value {; scope})

Parameter	Type	Description
name	Text	Name of the preference item
value	Boolean	The new preference value
scope	Longint	The preference's scope (optional)

This routine stores a Boolean value.

```
Fnd_Pref_SetBoolean ("Toolbar";$displayToolbar_b)
```

If no scope is specified, the preference is stored as a local preference. Pass a third parameter to specify the scope of the preference:

```
Fnd_Pref_SetBoolean ("Toolbar";$displayToolbar_b;Fnd_Pref_Shared )
```

A preference's scope should not be changed. Doing so may cause unpredictable results.

Fnd_Pref_SetLongInt

Fnd_Pref_SetLongInt (name; value {; scope})

Parameter	Type	Description
name	Text	Name of the preference item
value	Longint	The new preference value
scope	Longint	The preference's scope (optional)

This routine stores a long integer value.

```
Fnd_Pref_SetLongInt ("TimeoutInSeconds";$timeout_i)
```

If no scope is specified, the preference is stored as a local preference. Pass a third parameter to specify the scope of the preference:

```
Fnd_Pref_SetLongInt ("TimeoutInSeconds";$timeout_i;Fnd_Pref_Server )
```

A preference's scope should not be changed. Doing so may cause unpredictable results.

Fnd_Pref_SetReal

Fnd_Pref_SetReal (name; value {; scope})

Parameter	Type	Description
name	Text	Name of the preference item
value	Real	The new preference value
scope	Longint	The preference's scope (optional)

This routine stores a real number value.

```
Fnd_Pref_SetReal ("ExchangeRate";$rate_r)
```

If no scope is specified, the preference is stored as a local preference. Pass a third parameter to specify the scope of the preference:

```
Fnd_Pref_SetReal ("ExchangeRate";$rate_r;Fnd_Pref_Shared )
```

A preference's scope should not be changed. Doing so may cause unpredictable results.

Fnd_Pref_SetText

Fnd_Pref_SetText (name; value {; scope})

Parameter	Type	Description
name	Text	Name of the preference item
value	Text	The new preference value
scope	Longint	The preference's scope (optional)

This routine stores a real number value.

```
Fnd_Pref_SetText ("EmailAddress";$email_t)
```

If no scope is specified, the preference is stored as a local preference. Pass a third parameter to specify the scope of the preference:

```
Fnd_Pref_SetText ("EmailAddress";$email_t;Fnd_Pref_Server )
```

A preference's scope should not be changed. Doing so may cause unpredictable results.

Fnd_Pref_UserName

Fnd_Pref_UserName (name; value {; scope})

Parameter	Type	Description
name	Text	Name of the preference item
value	Text	The new preference value
scope	Longint	The preference's scope (optional)

This routine gets and sets the current user name used by the Preferences component. When setting a server preference, the preference is stored with the current user's name. By default the value returned by 4D's **Current user** function is used. But if you do not use 4D's password system then everybody's name becomes "Designer," and everyone shares the same preferences (use this component's shared preferences feature if you want to do this). So to specify a unique name for each user, you can pass the user's name (or any other unique text identifier) to this function:

```
Fnd_Pref_UserName (String([User]ID))
```

This method can also be called as a function to get the current user name setting:

```
$userName_t:=Fnd_Pref_UserName
```

Fnd_Pref_SetWindow

Fnd_Pref_SetWindow ({{name}; window ref})

Parameter	Type	Description
name	Pointer	Name of the preference item (optional)
window ref	Longint	Window to save (optional)

Fnd_Pref_SetWindow (name; left; top; right; bottom)

Parameter	Type	Description
name	Text	Name of the preference item
left	Longint	Left coordinate
top	Longint	Top coordinate
right	Longint	Right coordinate
bottom	Longint	Bottom coordinate

This routine saves the coordinates of the current window in the user's preferences file. The values can be set either by using the window reference or the left, top, right, and bottom coordinates.

To save the values using the window reference, pass a preference name and the window reference number that has been returned by one of 4D's window opening commands. If no name is specified, the current process name is used. If no window is specified, the frontmost window of the current process is used.

See the example in *Fnd_Pref_GetWindow*.

To save the values using the left, top, right, and bottom coordinates, pass a preference name and the coordinates. There are no optional parameters when using the window coordinates.

```
$left_i:=20
$top_i:=50
$right_i:=320
$bottom_i:=250
Fnd_Pref_SetWindow ("MonitorWindow";$left_i;$top_i;$right_i;$bottom_i)
```

Window coordinates are always stored locally. They cannot be stored as server or shared preferences.

Password Component

Fnd_Pswd

The Password component provides utility routines for generating passwords. You can call these commands from anywhere in your database.

Language Reference

Here are the routines in Foundation's Password component:

Fnd_Pswd_BulletEntry
Fnd_Pswd_CustomCharacters
Fnd_Pswd_ExcludeCharacters
Fnd_Pswd_GeneratePassword
Fnd_Pswd_GeneratorDialog
Fnd_Pswd_Info
Fnd_Pswd_MaxLength
Fnd_Pswd_MinLength
Fnd_Pswd_UseLowercase
Fnd_Pswd_UseNumbers
Fnd_Pswd_UseSymbols
Fnd_Pswd_UseUppercase

Fnd_Pswd_BulletEntry

Fnd_Pswd_BulletEntry (->visible object; ->text variable)

Parameter	Type	Description
visible object	Pointer	Pointer to the displayed variable
text variable	Pointer	Entered value

The *Fnd_Pswd_BulletEntry* routine replaces typed characters with bullet characters for password entry. Requires the field's On Before Keystroke event to be enabled.

```
Fnd_Pswd_BulletEntry(->[Contact]Password;->password_t)
```

Fnd_Pswd_CustomCharacters

Fnd_Pswd_CustomCharacters ({characters}) ➜ Text

Parameter	Type	Description
characters	Text	Custom characters to use to generate passwords (optional)
Function result	Text	Currently specified custom characters

This function returns the currently specified custom characters to use to generate passwords. Optionally, you may pass any custom characters that should be used to generate passwords.

```
$passwordcharacters_t:=Fnd_Pswd_CustomCharacters ("abcxyz123890")
```

Fnd_Pswd_ExcludeCharacters

Fnd_Pswd_ExcludeCharacters ({characters}) ➜ Text

Parameter	Type	Description
characters	Text	Custom characters to exclude from passwords (optional)
Function result	Text	Currently excluded characters

This function returns the currently excluded custom characters cannot be used to generate passwords. Optionally, you may pass any custom characters that should be excluded from generated passwords.

`$excludedcharacters_t:=Fnd_Pswd_ExcludeCharacters ("1234567890")`

Fnd_Pswd_GeneratePassword

Fnd_Pswd_GeneratePassword ➜ Text

Parameter	Type	Description
Function result	Text	Generated password

This function returns a password.

`$password_t:=Fnd_Pswd_GeneratePassword`

Fnd_Pswd_GeneratorDialog

Fnd_Pswd_GeneratorDialog ➜ Text

Parameter	Type	Description
Function result	Text	Password

This function displays a password generator dialog and returns the password.

`$password_t:=Fnd_Pswd_GeneratorDialog`

Fnd_Pswd_Info

Fnd_Pswd_Info (info requested) ➔ Text

Parameter	Type	Description
info requested	Text	Info desired
Function Result	Text	Response

Returns the requested information about the component.

`$version_t:=Fnd_Pswd_Info ("version")`

The *Fnd_Pswd_Info* method will respond to these requests:

Request	Response	Example
name	The component's full name	Foundation Password
version	The component's version number	4.2

This routine can also be called using the *Fnd_Gen_ComponentInfo* method without first testing to see if the component is installed:

```
$version_t:=Fnd_Gen_ComponentInfo ("Fnd_Pswd";"version")
```

See the *Fnd_Gen_ComponentInfo* method for more information.

Fnd_Pswd_MaxLength

Fnd_Pswd_MaxLength ({max chars to use{; max limitation}}) ➜ Longint

Parameter	Type	Description
max chars to use	Longint	Maximum password length setting (optional)
max limitation	Longint	Maximum value displayed in the pop-up menu (optional)
Function result	Longint	Currently set maximum password length

This function returns the current value of the maximum password length. Optionally, allows the developer to set the maximum password length, and, optionally, the maximum password length defined in the Generator dialog.

```
$maxpasswordlength_i:=Fnd_Pswd_MaxLength (12;8)
```

Fnd_Pswd_MinLength

Fnd_Pswd_MinLength ({min chars to use{; min limitation}}) ➜ Longint

Parameter	Type	Description
min chars to use	Longint	Minimum password length setting (optional)
min limitation	Longint	Minimum value displayed in the pop-up menu (optional)
Function result	Longint	Currently set minimum password length

This function returns the current value of the minimum password length. Optionally, allows the developer to set the minimum password length, and, optionally, the minimum password length defined in the Generator dialog.

```
$minpasswordlength_i:=Fnd_Pswd_MinLength (6;6)
```

Fnd_Pswd_UseLowercase

Fnd_Pswd_UseLowercase ({use lowercase?}) ➜ Boolean

Parameter	Type	Description
use lowercase	Boolean	True to use lowercase in passwords
Function result	Boolean	Current setting of whether to use lowercase

This function returns current setting of whether to use lowercase in passwords. Optionally, pass True to tell the password generator to use lowercase letters in passwords.

`$uselowercase_b:=Fnd_Pswd_UseLowercase (True)`

Fnd_Pswd_UseNumbers

Fnd_Pswd_UseNumbers ({use numbers?}) ➔ Boolean

Parameter	Type	Description
use lowercase	Boolean	True to use numbers in passwords
Function result	Boolean	Current setting of whether to use numbers

This function returns current setting of whether to use numbers in passwords. Optionally, pass True to tell the password generator to use numbers in passwords.

`$usenumbers_b:=Fnd_Pswd_UseNumbers (True)`

Fnd_Pswd_UseSymbols

Fnd_Pswd_UseSymbols ({use symbols?}) ➔ Boolean

Parameter	Type	Description
use symbols	Boolean	True to use symbols in passwords
Function result	Boolean	Current setting of whether to use symbols

This function returns current setting of whether to use symbols in passwords. Optionally, pass True to tell the password generator to use symbols in passwords.

`$usesymbols_b:=Fnd_Pswd_UseSymbols (True)`

Fnd_Pswd_UseUppercase

Fnd_Pswd_UseUppercase ({use uppercase?}) ➔ Boolean

Parameter	Type	Description
use uppercase	Boolean	True to use uppercase in passwords
Function result	Boolean	Current setting of whether to use lowercase

This function returns current setting of whether to use uppercase in passwords. Optionally, pass True to tell the password generator to use uppercase letters in passwords.

`$useuppercase_b:=Fnd_Pswd_UseUppercase (True)`

Records Component

Fnd_Rec

Foundation's Records component give you the tools to modify a selection of records. These are commands that the shell calls to affect records displayed in an output form.

Language Reference

Here are the routines in Foundation's Records component:

- Fnd_Hook_Rec_Delete
- Fnd_Hook_Rec_New
- Fnd_Rec_DeleteUserSet
- Fnd_Rec_Info
- Fnd_Rec_Info
- Fnd_Rec_OmitSubset
- Fnd_Rec_ShowAll
- Fnd_Rec_ShowSubset

Fnd_Hook_Rec_Delete

Fnd_Hook_Rec_Delete ➜ Boolean

Parameter	Type	Description
No parameters required.		
Function result	Boolean	Allow Foundation to handle the request?

This hook gets called before Foundation attempts to delete any records. This hook returns True to allow Foundation to display a warning and delete the records. Return False if you want to handle the process yourself.

Call *Fnd_Gen_CurrentTable* to determine the table in use. The “UserSet” set contains the records the user wants to delete.

Fnd_Hook_Rec_New

Fnd_Hook_Rec_New (->table) ➜ Boolean

Parameter	Type	Description
table	Pointer	Pointer to the table for which to create a new record
Function result	Boolean	Allow Foundation to handle the request?

This hook gets called when the user wants to add a new record. It returns True to allow Foundation to handle the request. Return False if you want to handle the request yourself.

The first parameter passed to this method is the table for which the new record will be created.

Fnd_Rec_DeleteUserSet

Fnd_Rec_DeleteUserSet

Parameter	Type	Description
No parameters required.		

Fnd_Rec_DeleteUserSet deletes the highlighted records in the current selection (the “UserSet”). This method asks the user first if they are sure the records should be deleted. It was designed to be called from the Delete button on the toolbar or from a Delete menu item.

Fnd_Rec_Info

Fnd_Rec_Info (info requested) ➜ Text

Parameter	Type	Description
info requested	Text	Info desired
Function Result	Text	Response

Returns the requested information about the component.

```
$version_t:=Fnd_Rec_Info ("version")
```

The *Fnd_Rec_Info* method will respond to these requests:

Request	Response	Example
name	The component's full name	Foundation Records
version	The component's version number	4.0.2 beta 8

This routine can also be called using the *Fnd_Gen_ComponentInfo* method without first testing to see if the component is installed:

```
$version_t:=Fnd_Gen_ComponentInfo ("Fnd_Rec";"version")
```

See the *Fnd_Gen_ComponentInfo* method for more information.

Fnd_Rec_NewRecord

Fnd_Rec_Info ({->table})

Parameter	Type	Description
table	Pointer	Pointer to the table to use (optional)

This method creates a new record for the specified table (if any). Otherwise a new record is created for the current table.

```
Fnd_Rec_Info (->[Contacts])
```

Fnd_Rec_OmitSubset

Fnd_Rec_OmitSubset

Parameter	Type	Description
No parameters required.		

This method omits the highlighted records from the current selection. Assumes the frontmost window is displaying a 4D output form.

Fnd_Rec_ShowAll

Fnd_Rec_ShowAll ({->table})

Parameter	Type	Description
table	Pointer	Pointer to the table to use (optional)

This method updates the selection to include all records. If the Macintosh Option key or Windows Alt key is down, the selection is reduced to no records. Assumes the frontmost window is displaying a 4D output form.

Fnd_Rec_ShowSubset

Fnd_Rec_ShowSubset

Parameter	Type	Description
No parameters required.		

This method reduces the current selection to just the highlighted records. Assumes the frontmost window is displaying a 4D output form.

Registration Component

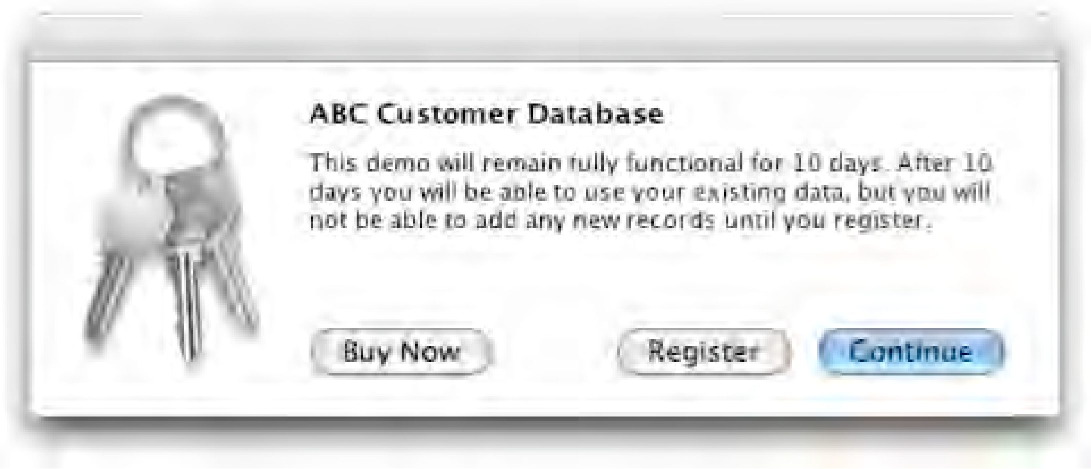
Fnd_Reg & Fnd_RegG

The Foundation Registration component is a 4th Dimension component designed to help you create demonstration or limited versions of your product that can later be converted by the end user to a full working version by entering a personalized user name and unlock code.

This component can easily be added to a database that has been created with the Foundation Shell, or it can be added to any database by combining it with a few other Foundation components (see the Installation section for details).

There are actually two separate components you will be using. One is the Fnd_Reg component, which you will add to the database you will be distributing. The other is the Fnd_RegG component (Registration Generator), which you will add to your private customer database. This component will generate the activation codes used by the Fnd_Reg component to unlock the public database.

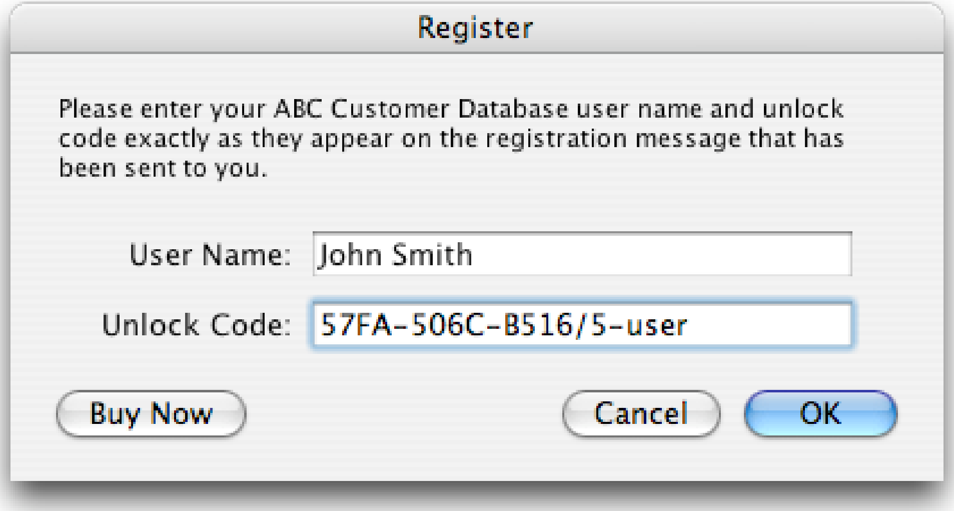
When an end-user launches your application, they will see a message at startup that indicates that the program is in demo mode, and will have limited functionality until an unlock code is entered.



ABC Customer Database

This demo will remain fully functional for 10 days. After 10 days you will be able to use your existing data, but you will not be able to add any new records until you register.

When the user purchases a license to the product, they can click the **Register** button to enter in their personalized user name and unlock code. This will enable some or all of the features of your program. Only the unlock code that matches the user name will unlock the program.



Register

Please enter your ABC Customer Database user name and unlock code exactly as they appear on the registration message that has been sent to you.

User Name: John Smith

Unlock Code: 57FA-506C-B516/5-user

The unlock code is a short string consisting of some hexadecimal numbers and dashes. It can optionally contain additional license information as shown in the screen shot. The unlock code can be generated by the Fnd_RegG component installed in your customer database. The number is generated using a public algorithm and a private keyword, so it can be also be created by almost any other program using any computer language. For example, you could generate a personalized unlock code using PHP from your web site.

This system can provide registration for an entire application or a specific feature of an application. A product's demo can be limited by time (it stops working after X number of days) or by feature (functionality is limited until the product is registered). Any combination of these two can be used. You can use the routines in this component to create almost any type of demo.

For example, you could provide any combination of these demo behaviors:

- The application is fully functional for 15 days after the first launch. After 15 days the user can still view and modify previously entered data, but cannot create new records.
- The application is fully functional, but limits the number of records that can be entered until an activation code is entered.
- The application is partially functional with no time limitation. Entering an activation code enables additional features.
- The application is partially functional for 60 days, then cannot be launched without an activation code.

You can completely customize the look and content of the registration dialogs. The optional **Buy Now** button opens a URL in the user's default browser.

Multiple Features

In addition to registration for the overall product, the Foundation Registration system can also be used to register any number of optional features in an application. For example, you can offer the ability to "add-on" accounting features or enable integration with another program by entering additional unlock codes.

Most of the routines in this component accept a feature code as the first parameter. If you omit this parameter or pass an empty string, the component will modify the "APPL" feature. This is the default feature code of the application.

If you want to offer additional optional features, you will need to specify a unique feature code for each one to differentiate it from the default application feature code. Any text value will work, such as "invoicing" or "INV" or "My really cool invoice feature." The feature code is never displayed to the end user. However, a feature name is displayed in the registration dialogs. The feature name can be specified using the *Fnd_Reg_FeatureName* method. This allows the feature name to be localized.

Additional License Information

The registration system is also able to track license limitations, such as the number of concurrent users in a multi-user system. This is done by adding a your own custom string to the unlock code. For example, instead of just entering an unlock code like "BA90-E129-C9F6," the user would enter "57FA-506C-B516/5-user" to indicate that the license is valid for 5 concurrent users.

The "5-user" string above is just an example. You can chose to add any custom string to any license you generate, then retrieve this string at runtime to determine how your program will behave. The contents of this string are included in the checksum that is used to generate the registration number, so the unlock code will fail if the user attempts to change it. For example, if "57FA-506C-B516/5-user" would work, but "57FA-506C-B516" or "57FA-506C-B516/7-user" would not work.

The registration information is stored in the data file. If a new data file is created, the demo period starts over and the unlock code must be re-entered.

Generating Unlock Codes

Unlock codes are generated right in your 4D-based customer tracking database using the Foundation Registration Generator (Fnd_RegG) component. The simple unlock code generation algorithm can also be easily incorporated into other languages such as PHP, ASP, AppleScript, C++, etc. This allows you to offer automated online registration or use non-4D databases to generate the unlock codes. For details on the algorithm for creating unlock codes, see the How it Works topic.

The Secret Key

Since the unlock code generated by the Foundation Registration component is based on an public algorithm, what prevents other 4D developers (or anyone else with access to this documentation) from generating an unlock code for your application? You provide one secret string value that will be known only to the application you distribute and to the customer database that generates the activation codes. This string will be hidden within the code of your application, so the end user will never be able to see it.

You can use almost any word or phrase as your secret key. It can be as short as "dog" or as long as "this is my secret key and you cannot do anything about it." As with any password, it is important to use a secret key that cannot be guessed easily. Your secret key can include any punctuation or special characters, although there are some cross-platform issues involved in using characters with high-ascii values.

Generally, you will want to use a different secret key for each product you distribute. If you do not, a user would be able to purchase an unlock code for one product and then successfully use it to unlock your other products.

Feature Codes

The Registration component has built-in support for different features within one application, so you do not need a different secret key for the application and each feature. You will simply provide a feature code for each special feature you wish to handle. The application itself is just a feature that has been pre-named "Application." When working with the application you can either pass this name as the feature code or a blank string.

Examples of feature codes might include "Payroll," "Printing," or "Advanced Features." It is not important that these feature codes be difficult to guess. They are simply for you to identify different features, so it is best to select easy to remember words or phrases.

The User Interface

When a user downloads and launches your application, the first thing they see will be the Demo Dialog. Before the demo period starts (*Fnd_Reg_StartDemoPeriod* has never been called) no message indicating the remaining number of days remaining is displayed.

After the demo period starts (*Fnd_Reg_StartDemoPeriod* has been called) an additional line of text displays the number of days remaining before the trial period ends. Now the **Continue** button in the registration dialog is disabled for the first 7 seconds. A countdown is displayed to show the user how long before the button will become enabled.

During the final week of the trial period, the message is displayed in red.

Installation

This section describes how to install the Fnd_Reg component into a database that you will distribute as a demonstration version. You should install only the Fnd_Reg component into this database. Later you will install the Fnd_RegG component into your customer database for creating personalized unlock codes.

The Registration component stores information in a database table. To work around this problem, the Foundation Registration component uses a new technique for accessing the database structure. No tables are included with the component. Instead, you will need to add the tables to your structure before installing the component. Internally the component creates pointers to the structure's tables and fields and then uses these throughout the code.

Create the [Fnd_Reg] Table

To use the Foundation Registration component, you will first need to add a table to your structure. The table must be named "Fnd_Reg" and must contain the fields listed below. The order of the fields is not important, and it is okay if the table contains other unused fields (in case you are reusing a table). However, the field names and types must be set up exactly as shown below.

Field Name	Type	Attributes
[Fnd_Reg]ID	Long Integer	Indexed
[Fnd_Reg]Feature_Code	Alpha 80	Indexed
[Fnd_Reg]Demo_Days	Integer	
[Fnd_Reg]Demo_Start_Date	Date	
[Fnd_Reg]Unlock_Code	Alpha 40	
[Fnd_Reg]User_Name	Alpha 80	
[Fnd_Reg]License_Info	Alpha 80	
[Fnd_Reg]Checksum	Alpha 40	

Install the Component

The Foundation Registration component requires the following components (shown with minimum required version numbers). It also requires version 4.2 or later of the Foundation Extras plugin.

Component	Minimum Version
Fnd_Gen	4.2
Fnd_Text	4.1.2
Fnd_Wnd	4.1.4

Updating the Component

If you later upgrade (or reinstall) the component, you will be asked if you want to update the two public forms. Click **Yes** if you have not customized these forms for your application, or if you wish to revert to the default layouts. Otherwise click **No** to save any changes you may have made.

Quick Start

After you have installed the Fnd_Reg component to your database, think of a secret keyword or phrase for your product. This is the password for creating registration numbers, so make sure it is not easy to guess. See The Secret Key for details.

Add the code below to your database's *On Startup* database method. Generally you will want to do this before calling any other code (including other Foundation routines, including *Fnd_Shell_Startup*).

```
Fnd_Reg_SetSecretKey ("MySecretKey")
Fnd_Reg_BuyNowURL ("";"http://www.Mere-Mortal-Software-com/tracker/buynow.php")
Fnd_Reg_DemoDialog
```

Replace "MySecretKey" above with your own secret keyword or phrase that you have selected for this application. You must call *Fnd_Reg_SetSecretKey* before calling any other Fnd_Reg routine. A Foundation Bug Alert dialog will be displayed if this routine is not called before any other Fnd_Reg routine. For more information about the secret keyword, see the How it Works section.

Decide how many days you want your program to operate before features become disabled or the application refuses to launch. By default this is 30 days. If you want to set a different trial period, add a call to *Fnd_Reg_DemoPeriod* just before the call to *Fnd_Reg_DemoDialog*:

```
Fnd_Reg_SetDemoPeriod (15)  ` Set a 15 day trial period.
```

Decide what action must be taken to start the demo period. The trial period can start when the program is first launched, or by any other action, such as when the first record is saved. Add a call to *Fnd_Reg_StartDemoPeriod* wherever you want to start the demo period. There is no need to determine if this routine has already been called, since it will have an affect only the first time it is called.

```
Fnd_Reg_StartDemoPeriod
```

Keep in mind that the demo start period and unlock code are stored in the data file, so the trial period will start over if a new data file is created.

Finally, install the Fnd_RegG component to your customer database. To generate an unlock code for a customer, pass the customer name (it can be an individual's name or a company name) to the *Fnd_RegG_GenerateUnlockCode* method:

```
$unlock_t:=Fnd_Reg_GenerateUnlockCode ("MySecretKey";"";[Customer]ComanyName)
```

The second parameter (feature code) is optional, so just pass a blank string for now. To unlock the application, the user will need to enter both the unlock code and the customer name that was passed to this routine.

That is it. You are ready to distribute your database as a fully functional, time limited demo. See the next section if you would like to offer different trial options.

Techniques

Feature Codes

All of the registration routines accept an optional "feature code" parameter. This allows you to set up trial periods for additional program features. If you do not pass a feature code to a routine, internally the component uses "APPL" as the feature code. All of the registration routines behave identically but independently from other feature codes. For example, you could offer a demo of your advanced charting features separately from the application. So a user could try the charting routines long after registering and unlocking the application.

Throughout this documentation the word "feature" can refer to a specific optional feature or the default "APPL" feature code.

Determining the Current Registration State

The *Fnd_Reg_RegistrationState* routine will return the current state of the registration routines. You can call it at any time to determine how your program should behave. It will return one of these constant values:

Fnd_Reg_PreDemo
Fnd_Reg_Demo
Fnd_Reg_Expired
Fnd_Reg_Registered

Fnd_Reg_PreDemo: The feature has not been registered yet and the demo period has not been started (there is no expiration date).

Fnd_Reg_Demo: The feature has not been registered and the demo period has been started (there is an expiration date), but has not yet expired. Use the *Fnd_Reg_GetExpirationDate* or *Fnd_Reg_GetDemoDaysRemaining* functions to determine when the trial period will expire.

Fnd_Reg_Expired: The feature has not been registered and the demo period's expiration date has passed. Call *Fnd_Reg_GetExpirationDate* to determine when the trial expired.

Fnd_Reg_Registered: The feature has been registered.

Selecting a Secret Product Key

The secret key is basically the password to your product registration, so it is important to treat it with the same care you would treat any password. In addition to keeping it safe in your office, you should also do your best to obscure it in your database structure file.

It is possible to view the contents of a 4D structure file with a text editor. If your password is a simple string such as "MyPassword," it is easy to spot. On the other hand, if your password is "xtX4ptÂz>?" then it is much more difficult to find.

This applies to compiled databases too. In fact, the 4D compiler is so smart it can even hinder your efforts to obscure the password. For example, it is easy to spot this line of code in your compiled structure with a text editor:

```
Fnd_Reg_SetSecretKey ("MyPassword")
```

In the text editor, the string looks like this:

```
MyPassword
```

So, in an attempt to hide it, you instead enter it like this:

```
Fnd_Reg_SetSecretKey ("M"+"y"+"P"+"a"+"s"+"s"+"w"+"o"+"r"+"d")
```

The problem is that the 4D compiler is very smart. It is smart enough to know that it can combine these strings at compile time rather than at runtime. So in your compiled code it also looks like this in the text editor:

```
MyPassword
```

Okay, so we need to get tricky to fake-out this smart compiler. Let us try this:

```
Fnd_Reg_SetSecretKey (Char(77)+Char(121)+Char(80)+Char(97)+Char(115)+
  Char(115)+Char(119)+Char(111)+Char(114)+Char(100))
```

That is a little better, but we have only made a small change. It looks like this in the text editor:

```
* M * y * P * a * s * w * o * r * d
```

So, unless you want to get really fancy (which is certainly possible) it may be best to simply pick an obscure secret key, such as "xtX4ptÂz>?." Although it is still stored as a simple text string, it is much more difficult to spot when the structure file is viewed with a text editor.

Custom Implementation Techniques

Instead of using the registration component to limit the number of days your unregistered product can be used, you can instead use the component to enforce other types of limitations. Here are just some of the optional registration techniques that you can implement using the Foundation Registration component.

Limit the Number of Records that can be Created

To limit the demo by the number of records rather than the number of days (similar to the way the 4D demo works), do not start the demo period. Instead, just modify Foundation's *Fnd_Hook_Rec_NewRecord* hook so that it checks to see if the application is unlocked, and if not, displays an error message when the user tries to add more records than the demo allows.

Limit the Demo Period to Minutes

Another way to offer a trial version is to limit the number of minutes the program can be used after it is launched. If the user wants to continue using the database, they must quit and relaunch it. This is the way the Foundation demo works.

To do this, you will need to get the current time when the application is launched, and then compare this to the current time when performing some standard operations. It is best to use the **Tickcounts** function, so that the program behaves as expected if launched around midnight.

Immediately after the demo dialog is displayed, determine when the demo should end:

```
Fnd_Reg_SetSecretKey ("MySecretKey")
$msg_t:="This database can be used for 15 minutes per launch until it is purchased."
Fnd_Reg_DemoMessage ("FancyGraphs";$msg_t)
Fnd_Reg_BuyNowURL ("";"http://www.Mere-Mortal-Software.com/tracker/buynow.php")
Fnd_Reg_DemoDialog
◇DemoEndTickcount_i:=Tickcount+(15*3600)  ` Quit after 15 minutes.
```

Then create a routine that determines if it is time to quit:

```
  ` Project Method: CheckDemoTimeout
If (Fnd_Reg_RegistrationState #Fnd_Reg_Registered )
  Fnd_Alert ("Your session has timed-out. Please relaunch the application to continue.")
  Fnd_Gen_QuitNow (True) ` Or insert your custom quit routine here.
End if
```

Now just call this method liberally throughout your code. For example, you could call it from some of Foundation's hooks so the time-out is checked whenever the user searches or displays an input form.

Creating a Trial Feature that Never Expires

Instead of creating a demo that times-out after a specific number of days, you may want your application (or a feature) to operate in demo mode indefinitely. This is ideal for upgraded features that you want your existing customers to purchase.

To do this, just do not call the *Fnd_Reg_StartDemoPeriod* method. Then in your code, offer limited functionality unless the *Fnd_Reg_StartDemoPeriod* returns Fnd_Reg_Registered:

```
ALL RECORDS([Invoices])
ORDER BY([Invoices];[Invoices]InvoiceDate;<)
$msg_t:="This is an optional feature, so only 20 records will be used until it is purchased."
Fnd_Reg_DemoMessage ("FancyGraphs";$msg_t)
Fnd_Reg_DisplayDialog ("FancyGraphs")  ` Will not appear if unlocked.
If (Fnd_Reg_RegistrationState ("FancyGraphs")#Fnd_Reg_Registered )
  REDUCE SELECTION([Invoices];20)  ` Use just 20 records in demo mode.
  ORDER BY([Invoices];[Invoices]InvoiceDate;<)  ` Resort them.
End if
DoFancyGraphs
```

Limiting the Number of Concurrent Users

Many 4D projects are licensed per consecutive user. Although 4D Server can control the number of consecutive users, it would be possible for your customer to purchase an additional 4D Client license directly from 4D without also licensing your custom application for another user. To prevent this, you can include the number of licensed users in the unlock code.

To do this, we first need to modify our call to *Fnd_RegG_GenerateUnlockCode* by passing it some additional license information:

```
$personalization_t:=[Customer]CompanyName
$users_t:=String([Customer]LicensedUsers)
$unlock_t:=Fnd_Reg_GenerateUnlockCode ("MySecretKey";"";$personalization_t;$users_t)
```

This license information is added to the unlock code. For example, if the customer is limited to 25 users, the registration code might look something like this:

```
57FA-506C-B516/25
```

This makes it easy to see that the unlock code is valid for 25 users. But because of the way the unlock code is constructed, simply changing the number at the end of the unlock code will render it useless.

Now in your application, you can retrieve this number using the *Fnd_Reg_GetLicenseInfo* function:

```
$licensedUsers_t:=Fnd_Reg_GetLicenseInfo
```

If no unlock code has been entered, the *Fnd_Reg_GetLicenseInfo* function will return an empty string.

Locking the Application to a Specific Computer

You may wish to register your application to a specific computer. To do this, you can get the computer's ethernet ID or serial number (another plugin will be needed for this) and use part of it as the license information (using the entire ethernet ID address may be too much for the average person to enter correctly).

Another technique would be to procedurally add it to the personalization used. Or use it instead of asking the user to enter any personalization information. This would require some minor modifications to the component's source code, since you cannot procedurally supply the personalization information.

Clearing the Registration Data

To allow the user to reset the registration period for a specific feature, call *Fnd_Reg_Reset*. To reset multiple features, you must call this routine once for each feature.

Keep in mind that all registration information is essentially reset when a new data file is created.

How the Unlock Code is Generated

The following information is presented primarily for developers that wish to generate unlock codes using programs other than 4D. If you are using the Fnd_RegG component to generate unlock codes from your 4D-based customer database, you will not need to know any of the information presented in this section.

MD5

The unlock code generated by the Fnd_RegG component is simply the MD5 hash of your secret key, the feature code, the user's name (or company name or other unique information) and any additional license information you may wish to add. By using an industry-standard MD5 hash, it is possible to generate a license code using any language (even AppleScript), but only if you know the secret product key. Since it is not possible to determine the input value from an MD5 hash, this provides more than adequate security for most applications. You can learn more about the MD5 message digest here:

http://www.faqs.org/rfcs/rfc1321.html

An MD5 value works very well for an unlock code since it consists only of numbers and the letters A through G (hexadecimal values). Unlike more complex unlock codes, hexadecimal values do not include the letters I, L, or O, so there is little chance of confusing these letters with the numbers 1 and 0.

Input Values

The input string for generating the MD5 is a combination of these strings:

```
secret product key
feature name
user identification information
license information
```

These values are combined into a single string separated by double colons ("::"). Then the entire string is converted to an ISO 8859-1 format to ensure that an identical unlock code will be generated on both Macintosh and Windows.

Finally, this string is then used to generate an MD5 value.

Formatting

Rather than using the entire 32 character MD5 value, Foundation uses just the first 12 characters. Although this does not offer the security of using all 32 characters, it offers sufficient security for almost any 4D-based application. Dashes are also added between each four characters to make the code easier to read.

Verification

It is not possible to take an MD5 value (especially a partial one) and convert it back to the original string used to create it. So instead the application uses the same process as the registration generator to create an unlock code, and this unlock code is compared to the unlock code entered into the registration dialog by the end user. If the unlock codes match, the entered value is stored in the data file. The application supplies the secret product key and the feature code, and the user types in the personalization information and the license info (as part of the unlock code).

Each time the program is used the unlock code is retrieved from the data file and this process is repeated to verify that the saved unlock code is still valid.

An Example

To explain the process, we will use an example. Let us say we have selected "jay%99west@" as our secret product key, and that we want to generate an application unlock code for a customer named John Doe. Additionally, we want to indicate that the application can be used with 10 clients, so we want to add "10-user" as the license information.

First, we will concatenate the strings into one string we can pass to our MD-5 routine. We need to separate each of the input strings with two colons ("::"). This is done just to make the string slightly easier to read during development. We combine the strings in this order:

```
secret product key::feature name::user identification information::license information
```

So our input string for this example looks like this:

```
jay%99west@::APPL::John Doe::10-user
```

The input string contains "::APPL::" because Foundation uses "APPL" as the feature code for the application.

Even if no license information is used, the two-colon separator is still used after the user name:

```
jay%99west@::APPL::John Doe::
```

Note that any of the input values may already include the "::" character sequence. That is not a problem, since there is no need to parse this data into its separate elements.

Just to avoid problems, none of the input values should contain a carriage return or a line break.

Next, before generating an MD5 value from this string, the string is converted to ISO 8859-1. Since none of our text includes any high-ASCII characters, it still looks the same after this conversion:

```
jay%99west@::APPL::John Doe::10-user
```

And finally, the input string is converted to an MD5 value. The MD5 value for our example looks like this:

```
35bf6fd6b73c1e29ac15a50316c4cc6c
```

Now we take the first 12 characters and toss in a couple of dashes. We also convert the letters to uppercase:

```
35BF-6FD6-B73C
```

Language Reference

Here are the routines in Foundation's Registration component:

- Fnd_Reg_BuyNowButton
- Fnd_Reg_BuyNowURL
- Fnd_Reg_DemoDays
- Fnd_Reg_DemoDialog
- Fnd_Reg_DemoDialogFormMethod
- Fnd_Reg_DemoMessage
- Fnd_Reg_FeatureName
- Fnd_Reg_GetDemoDaysRemaining
- Fnd_Reg_GetExpirationDate
- Fnd_Reg_GetLicenseInfo
- Fnd_Reg_GetUserName
- Fnd_Reg_Info
- Fnd_Reg_RegDialogFormMethod
- Fnd_Reg_RegDialogOKButton
- Fnd_Reg_RegistrationDialog
- Fnd_Reg_RegistrationState
- Fnd_Reg_Reset
- Fnd_Reg_SetSecretKey
- Fnd_Reg_StartDemoPeriod

Fnd_Reg_BuyNowButton

Fnd_Reg_BuyNowButton

Parameter	Type	Description
No parameters required.		

This method is called from the **Buy Now** button's object method in the component's two public forms. It opens the URL specified by a previous call to *Fnd_Reg_BuyNowURL*.

Fnd_Reg_BuyNowURL

Fnd_Reg_BuyNowURL ({feature code{; URL}}) ➔ Text

Parameter	Type	Description
feature code	Text	A registration feature code or "" (optional)
URL	Text	The new Buy Now button URL (optional)
Function result	Text	The current URL

This routine allows the developer to get and set the URL to display when the **Buy Now** button is clicked. Each feature has its own Buy Now URL. In most cases you will call this routine before calling *Fnd_Reg_DemoDialog*.

To set a new URL, pass it as the second parameter to this method. To set the URL for the application, pass a blank string as the feature code:

```
Fnd_Reg_BuyNowURL ("";"http://www.MyWebSite.com/buynow.html")
```

If called without a second parameter, this routine will return the currently set URL. To get the URL for the "APPL" feature you can also omit the feature code parameter:

```
$url_t:=Fnd_Reg_BuyNowURL
```

Fnd_Reg_DemoDays

Fnd_Reg_DemoDays ({feature code{; days}}) ➔ Longint

Parameter	Type	Description
feature code	Text	A registration feature code or "" (optional)
URL	Longint	The new demo period in days (optional)
Function result	Longint	The current number of demo days

This routine allows the developer to get and set the number of demo days for the specified feature. Generally this routine should be called at startup before any calls to *Fnd_Reg_DemoDialog*. Calling this method does not modify the expiration date if it has already been set by a call to *Fnd_Reg_StartDemoPeriod*.

The default demo period for all features is 30 days. To set a different value, pass it as the second parameter to this method. To set the URL for the application, pass a blank string as the feature code. This example sets the demo days for the "invoicing" feature to 15 days.

```
Fnd_Reg_DemoDays ("invoicing";15)
```

If called without a second parameter, this routine will return the currently set demo period:

```
$demoDays_i:=Fnd_Reg_DemoDays ("invoicing")
```

To get the URL for the "APPL" feature you can also omit all of the parameters.

Fnd_Reg_DemoDialog

Fnd_Reg_DemoDialog ({feature code})

Parameter	Type	Description
feature code	Text	A registration feature code or "" (optional)

This routine displays the demo message window if the user has not yet entered an unlock code for the specified feature. To display the application's demo window, call this method without parameters at startup after calling *Fnd_Reg_SetSecretKey*:

```
Fnd_Reg_SetSecretKey ("MySecretKey")
Fnd_Reg_DemoDialog
```

To display the demo dialog before performing an optional feature, call this method with the feature code:

```
Fnd_Reg_DemoDialog ("invoicing")
```

If the user has already entered a valid unlock code for the feature, the dialog will not be displayed and execution of method will continue uninterrupted.

You can call other registration routines before calling this method to configure the look and behavior of the demo dialog. See the Quick Start topic for an example. You can also modify the public “Fnd_Reg_DemoDialog” form to completely customize the appearance of the window.

Fnd_Reg_DemoDialogFormMethod

Fnd_Reg_DemoDialogFormMethod

Parameter	Type	Description
No parameters required.		

This method must be called from the “Fnd_Reg_DemoDialog” form method.

Fnd_Reg_DemoMessage

Fnd_Reg_DemoMessage ({feature code{; message}}) ➔ Text

Parameter	Type	Description
feature code	Text	A registration feature code or “” (optional)
message	Text	The new demo dialog message text (optional)
Function result	Text	The current message text

This routine allows the developer to get and set the demo message for the specified feature code. If this method is not called the default message contained in the localization strings is used, and can be edited using the Foundation Localization Editor. In most cases you will call this routine before calling *Fnd_Reg_DemoDialog*.

To set a new message, pass a second parameter to this method. To set the demo message for the application, pass a blank string as the feature code:

```
Fnd_Reg_DemoMessage ("";"This application will expire in 20 days.")
```

Instead of hard-coding the number of days, you can instead include “<<1>>” in the text passed to this method. When the dialog is displayed, this marker will be replaced by the number of days set by calling *Fnd_Reg_DemoDays*:

```
Fnd_Reg_DemoMessage ("";"This application will expire in <<1>> days.")
```

If called without a second parameter, this routine will return the currently set demo message. To get the demo message for the “APPL” feature you can also omit the feature code parameter:

```
$message_t:=Fnd_Reg_DemoMessage
```

Fnd_Reg_FeatureName

Fnd_Reg_FeatureName ({feature code{; feature name}}) ➔ Text

Parameter	Type	Description
feature code	Text	A registration feature code or “” (optional)
feature name	Text	The feature code’s display name (optional)
Function result	Text	The current feature code name

This feature name is displayed at the top of the Demo Dialog and in the text message of the Registration Dialog. You can call this method to set the display name for the application or any optional feature. By default, Foundation uses the database name set in the *Fnd_Hook_Shell_Setup* hook. If you are not using the Foundation Shell, you will need to supply the application name using this method or using the *Fnd_Gen_SetDatabaseInfo* method to set a “DatabaseName” value.

To set the feature name, pass a second parameter to this method:

```
Fnd_Reg_FeatureName ("INV";"Invoicing")
```

This method can also be used to get a feature name:

```
$featurName_t:=Fnd_Reg_FeatureName ("INV")
```

If the feature does not exist or no name has been specified, a blank string is returned.

Fnd_Reg_GetDemoDaysRemaining

Fnd_Reg_GetDemoDaysRemaining ({feature code}) ➔ Number

Parameter	Type	Description
feature code	Text	A registration feature code or “” (optional)
Function result	Number	The number of demo days remaining

This function returns the number of days left in the demo. This value is calculated using the number of demo days for the feature and the date the demo period was started by a call to *Fnd_Reg_StartDemoPeriod*.

```
$daysLeft_i:=Fnd_Reg_GetDemoDaysRemaining
```

Fnd_Reg_GetExpirationDate

Fnd_Reg_GetExpirationDate ({feature code}) ➔ Number

Parameter	Type	Description
feature code	Text	A registration feature code or "" (optional)
Function result	Date	The date the demo will expire

This function returns the specified feature's expiration date. This value is calculated using the number of demo days for the feature and the date the demo period was started by a call to *Fnd_Reg_StartDemoPeriod*.

```
$expires_d:=Fnd_Reg_GetExpirationDate
```

If the demo has not been started, the number of demo days is added to the current date, and the resulting date is returned.

Fnd_Reg_GetLicenseInfo

Fnd_Reg_GetLicenseInfo ({feature code}) ➔ Text

Parameter	Type	Description
feature code	Text	A registration feature code or "" (optional)
Function result	Text	The feature's license info

This function returns the license information that was included with the unlock code entered by the user. If the feature has not yet been registered, then a blank string is returned.

```
$licenseInfo_t:=Fnd_Reg_GetLicenseInfo ("INV")
```

Fnd_Reg_GetUserName

Fnd_Reg_GetUserName ({feature code}) ➔ Text

Parameter	Type	Description
feature code	Text	A registration feature code or "" (optional)
Function result	Text	The feature's license info

This function returns the user name or other personalization information that was entered by the end user when registering the product or feature. If the feature has not yet been registered, then a blank string is returned.

```
$userName_t:=Fnd_Reg_GetUserName
```

Fnd_Reg_Info

Fnd_Reg_Info (info requested) ➔ Text

Parameter	Type	Description
info requested	Text	Info desired
Function Result	Text	Response

This function returns the requested information about the Registration component.

```
$version_t:=Fnd_Reg_Info ("version")
```

The *Fnd_Reg_Info* method will respond to these requests:

Request	Response	Example
name	The component's full name	Foundation Registration
version	The component's version number	4.2

This routine can also be called using the *Fnd_Gen_ComponentInfo* method without first testing to see if the component is installed:

```
$version_t:=Fnd_Gen_ComponentInfo ("Fnd_Reg";"version")
```

See the *Fnd_Gen_ComponentInfo* method for more information.

Fnd_Reg_RegDialogFormMethod

Fnd_Reg_RegDialogFormMethod

Parameter	Type	Description
No parameters required.		

This method must be called from the “Fnd_Reg_RegisterDialog” form method.

Fnd_Reg_RegDialogOKButton

Fnd_Reg_RegDialogOKButton

Parameter	Type	Description
No parameters required.		

This method is called from the **OK** button’s object method in the component’s “Fnd_Reg_RegistrationDialog” public form. When clicked it verifies the user name and unlock code entered by the end user.

Fnd_Reg_RegistrationDialog

Fnd_Reg_RegistrationDialog ({feature code})

Parameter	Type	Description
feature code	Text	A registration feature code or "" (optional)

This routine displays the registration dialog window if the user has not yet entered an unlock code for the specified feature. It is not necessary to use this method, since the user can display the registration dialog by clicking the Register button in the Demo Dialog.

To request the unlock code for the application, no parameters are necessary:

```
Fnd_Reg_RegistrationDialog
```

To request the unlock code for a specific feature, pass the feature code parameter:

```
Fnd_Reg_RegistrationDialog ("webserver")
```

If the user has already entered a valid unlock code for the feature, the dialog will not be displayed and execution of method will continue uninterrupted.

Fnd_Reg_RegistrationState

Fnd_Reg_RegistrationState ({feature code}) ➔ Number

Parameter	Type	Description
feature code	Text	A registration feature code or "" (optional)
Function result	Number	The feature's registration state

This function returns the current registration state for the specified feature:

```
$state_i:=Fnd_Reg_RegistrationState
```

The value returned will be equal to one of these constant values:

Fnd_Reg_PreDemo
Fnd_Reg_Demo
Fnd_Reg_Expired
Fnd_Reg_Registered

Fnd_Reg_PreDemo: The feature has not been registered yet and the demo period has not been started (there is no expiration date).

Fnd_Reg_Demo: The feature has not been registered and the demo period has been started (there is an expiration date), but has not yet expired. Use the *Fnd_Reg_GetExpirationDate* or *Fnd_Reg_GetDemoDaysRemaining* functions to determine when the trial period will expire.

Fnd_Reg_Expired: The feature has not been registered and the demo period's expiration date has passed. Call *Fnd_Reg_GetExpirationDate* to determine when the trial expired.

Fnd_Reg_Registered: The feature has been registered.

Fnd_Reg_Reset

Fnd_Reg_Reset ({feature code})

Parameter	Type	Description
feature code	Text	A registration feature code or "" (optional)

Call this to clear any existing registration information for the specified feature:

```
Fnd_Reg_Reset ("webserver")
```

If called with no parameters, this routine will clear the registration information for the application.

In addition to removing a previously entered user name and unlock code, this routine clears the start date set by calling *Fnd_Reg_Init*.

Fnd_Reg_SetSecretKey

Fnd_Reg_SetSecretKey (key)

Parameter	Type	Description
key	Text	The secret product key

Use this method to set the secret product key for the application:

```
Fnd_Reg_SetSecretKey ("mySecretProductKey_773")
```

This must be the first Registration component method called. It should be called only once.

See the Techniques section of this chapter for important information about selecting a secret product key for your application.

Fnd_Reg_StartDemoPeriod

Fnd_Reg_StartDemoPeriod ({feature code})

Parameter	Type	Description
feature code	Text	A registration feature code or "" (optional)

Call this method to begin the demo period. If the demo period has already been started, this routine will do nothing.

```
Fnd_Reg_StartDemoPeriod
```

Additional Credits

Thanks to Vincent Tournier of e-Node for the French localization and for his outstanding help with alpha testing this component.

Shell Component

Fnd_Shell

The Foundation Shell component is the heart of the Foundation Shell. This component was not designed to be of much use except with all of the other Foundation components. Except for modifying the hooks, you generally will never need to call any of these components, unless you have a routine that needs to duplicate a shell action, such as displaying the Administration Dialog.

This component requires the following Foundation components:

- Foundation Art Component
- Foundation General Component
- Foundation List Component
- Foundation Preferences Component
- Foundation Virtual Structure Component
- Foundation Dialog Component
- Foundation IO Component
- Foundation Localization Component
- Foundation Records Component
- Foundation Windows Component

Language Reference

Here are the routines in Foundation's Shell component:

- Fnd_Hook_Shell_Setup
- Fnd_Shell_Administration

- Fnd_Hook_Shell_Administration
- Fnd_Hook_Shell_Find
- Fnd_Hook_Shell_InitializePlugIns
- Fnd_Hook_Shell_OpenTable
- Fnd_Hook_Shell_NavPalette
- Fnd_Hook_Shell_Print
- Fnd_Hook_Shell_Quit
- Fnd_Hook_Shell_Sort
- Fnd_Hook_Shell_SpecialFunctions
- Fnd_Hook_Shell_Startup
- Fnd_Shell_ExcludeFromQuit
- Fnd_Shell_Info
- Fnd_Shell_IsRunning
- Fnd_Shell_OpenTableAdd
- Fnd_Shell_NavigationPalette
- Fnd_Shell_OnStartup
- Fnd_Shell_OpenTableAdd
- Fnd_Shell_OpenTableDialog
- Fnd_Shell_Print
- Fnd_Shell_Show4DSplashScreen
- Fnd_Shell_SpecialFunctions

Fnd_Hook_Shell_Setup

Fnd_Hook_Shell_Setup

Parameter	Type	Description
No parameters required.		

Here is where the developer sets up information about this particular database.

```
Fnd_Gen_SetDatabaseInfo ("NAME";"Super Accounting Program")
Fnd_Gen_SetDatabaseInfo ("VERSION";"5.0.7b")
Fnd_Gen_SetDatabaseInfo ("COPYRIGHT";"Copyright ©2009 Amazing Company, Inc.")
Fnd_Gen_SetDatabaseInfo ("URL";"http://www.AmazingCompany.com/")
```

Fnd_Hook_Shell_Administration

Fnd_Hook_Shell_Administration

Parameter	Type	Description
No parameters required.		

This hook is called to display the shell's Administration Dialog. We have preconfigured it to display 4D's Password Editor, Foundation's List Editor and Sequence Number Editor, and 4D's Import and Export Editors. You can remove these options, or add new options by calling Foundation's Fnd_Hook_List_SetEditableLists method.

Fnd_Hook_Shell_Find

Fnd_Hook_Shell_Find

Parameter	Type	Description
No parameters required.		

This hook gives you a place to configure Foundation's Find Dialog if you want to alter the default configuration. You can call the *Fnd_Gen_CurrentTable* method to determine the table in use. See the Find Component chapter for more information.

Fnd_Hook_Shell_InitializePlugIns

Fnd_Hook_Shell_InitializePlugIns

Parameter	Type	Description
No parameters required.		

Use this method to initialize any plugins by calling their activation routines. This hook is called in the On Startup method for single-user and 4D Client, and also from the On Server Startup method when using 4D Server.

Fnd_Hook_Shell_OpenTable

Fnd_Hook_Shell_OpenTable

Parameter	Type	Description
No parameters required.		

This hook sets up Foundation's Open Table dialog before it is displayed. Call *Fnd_Hook_Shell_NavigationPalette* to specify the table names to display.

Fnd_Hook_Shell_NavPalette

Fnd_Hook_Shell_NavPalette

Parameter	Type	Description
No parameters required.		

This hook is called each time the Navigation Palette command is selected from the Tools menu. Use it to configure Foundation's Navigation Palette if you want to alter the default configuration. See the Navigation Component (Fnd_Nav) chapter for more information.

Fnd_Hook_Shell_Print

Fnd_Hook_Shell_Print

Parameter	Type	Description
No parameters required.		

This hook is called to display the shell's Print dialog. We have preconfigured it to display 4D's Quick Report Editor and Label Editor. You can remove these options, or add new options by calling Foundation's *Fnd_Hook_List_SetEditableLists* method.

Fnd_Hook_Shell_Quit

Fnd_Hook_Shell_Quit ➔ Boolean

Parameter	Type	Description
Function result	Boolean	Allow Foundation to quit?

The shell calls this hook when quitting the database, just before calling QUIT 4D (if compiled). Return True to allow Foundation to quit the application normally. If you return False, Foundation will abort any attempts to quit until the next time quit is selected from the File menu.

If you return False, it is your responsibility to let the user know why their attempt to quit the database is being ignored

You can also add any down code of your own in this method.

Fnd_Hook_Shell_Sort

Fnd_Hook_Shell_Sort

Parameter	Type	Description
No parameters required.		

This hook gives you a place to configure Foundation's Sort dialog if you want to alter the default configuration. You can call the *Fnd_Gen_CurrentTable* method to determine the table in use.

Fnd_Hook_Shell_SpecialFunctions

Fnd_Hook_Shell_SpecialFunctions

Parameter	Type	Description
No parameters required.		

This hook is called to display the shell's Special Functions Dialog. Routines in this dialog are available to all database users. By default, there are no options set up in this dialog. You can add new commands by calling Foundation's *Fnd_Hook_List_SetEditableLists* method.

Fnd_Hook_Shell_Startup

Fnd_Hook_Shell_Startup

Parameter	Type	Description
No parameters required.		

Place any startup code here that you want to run while the startup dialog is displayed.

You can also call your startup stuff before or after *Fnd_Shell_OnStartup* is called from the On Startup database method. However, code placed here will not be executed when you relaunch Foundation during development.

Fnd_Shell_Administration

Fnd_Shell_Administration

Parameter	Type	Description
No parameters required.		

Call this method to display the Foundation Shell's Administration Dialog.

Fnd_Shell_ExcludeFromQuit

Fnd_Shell_ExcludeFromQuit (name)

Parameter	Type	Description
name	Text	Name of the process

Normally when the user selects Quit from the File menu, Foundation does a call process to each process that was procedurally created and waits for all of these processes to quit before it calls QUIT 4D. If any of these processes do not end, Foundation displays a message to the user to tell them it cannot quit because a process will not end.

If you have a process that you do not want to end when the user selects quit, or that you are unable to quit (perhaps because it was launched by a plugin) then you can pass that process name to this method, and Foundation will not wait for it to quit before calling QUIT 4D.

Fnd_Shell_Info

Fnd_Shell_Info (info requested) ➔ Text

Parameter	Type	Description
info requested	Text	Info desired
Function Result	Text	Response

Returns the requested information about the component.

```
$version_t:=Fnd_Shell_Info ("version")
```

The *Fnd_Shell_Info* method will respond to these requests:

Request	Response	Example
name	The component's full name	Foundation Shell
version	The component's version number	4.2

This routine can also be called using the *Fnd_Gen_ComponentInfo* method without first testing to see if the component is installed:

```
$version_t:=Fnd_Gen_ComponentInfo ("Fnd_Shell";"version")
```

See the *Fnd_Gen_ComponentInfo* method for more information.

Fnd_Shell_IsRunning

Fnd_Shell_IsRunning ➔ Boolean

Parameter	Type	Description
Function result	Boolean	Is the shell running

This function returns True if the shell is running. This method is only needed during development. It will return False if you have selected Quit from the File menu, and the shell has switched back to menu bar #1.

Fnd_Shell_NavigationPalette

Fnd_Shell_NavigationPallette

Parameter	Type	Description
No parameters required.		

Call this method to display the Navigation Palette.

Fnd_Shell_OnExit

Fnd_Shell_On Exit

Parameter	Type	Description
No parameters required.		

This method must be called from the database's On Exit and On Server Shutdown database methods.

Fnd_Shell_OnStartup

Fnd_Shell_OnStartup

Parameter	Type	Description
No parameters required.		

The startup routine for the Foundation Shell. This method must be called from the database's On Startup and On Server Startup database methods.

Fnd_Shell_OpenTableDialog

Fnd_Shell_OpenTableDialog

Parameter	Type	Description
No parameters required.		

Displays a dialog with the names of all visible tables. If the user selects one and clicks the OK button the table is displayed in a new window (even if that table is already displayed in a window).

You can configure the tables listed in this dialog from the *Fnd_Hook_Shell_OpenTable* hook.

The table names and their order are determined by the Virtual Structure Component (Fnd_VS), if that component is installed.

Fnd_Shell_Print

Fnd_Shell_Print

Parameter	Type	Description
No parameters required.		

Call this method to display the shell's Print Dialog. Configure the contents of the dialog using the *Fnd_Hook_Shell_Print* hook.

Fnd_Shell_Show4DSplashScreen

Fnd_Shell_Show4DSplashScreen

Parameter	Type	Description
No parameters required.		

Allows the developer to display the 4D splash screen, or default window, so the database can be used in the User environment. No longer necessary in 4D v11, as there is no User environment.

Fnd_Shell_SpecialFunctions

Fnd_Shell_SpecialFunctions

Parameter	Type	Description
No parameters required.		

Call this method to display Foundation Shell's Special Functions Dialog. You can configure the contents of this dialog from the *Fnd_Hook_Shell_SpecialFunctions* hook.

Sort Component

Fnd_Sort

The Sort component provides a simplified way for the user to sort the current selection of records. It will automatically configure itself and sort the current selection of records, so no special setup is required. However, you can use these commands to change the default Sort dialog settings. You can call these commands from the *Fnd_Hook_Shell_Sort* hook.

Language Reference

Here are the routines in Foundation's Sort component:

- Fnd_Sort _AddField
- Fnd_Sort_AddSeparator
- Fnd_Sort_AddTable
- Fnd_Sort_Direction
- Fnd_Sort_Display
- Fnd_Sort_Info
- Fnd_Sort_OrderByEditor
- Fnd_Sort_SelectedField

Fnd_Sort_AddField

Fnd_Sort _AddField (->field{; position})

Parameter	Type	Description
field	Pointer	Pointer to the field to add
position	Longint	Position in which to add the item (optional)

The *Fnd_Sort_AddField* routine lets the developer add a field to the Sort dialog. If position is not specified, the field is added to the end of the selection list.

```
Fnd_Sort_AddField (->[Contact]Last Name)
Fnd_Sort_AddField (->[Company]Company Name)
Fnd_Sort_AddField (->[Contact]First Name;1)  ` Put it before the First Name
```

The field can be from the current or related table.

Fnd_Sort_AddSeparator

Fnd_Sort_AddSeparator ({position})

Parameter	Type	Description
position	Longint	Position of the separator (optional)

The *Fnd_Sort_AddSeparator* routine adds a separator line at position to the list of searchable fields to the Sort dialog.

```
Fnd_Sort_AddField (->[Contact]First Name)
Fnd_Sort_AddField (->[Contact]Last Name)
Fnd_Sort_AddSeparator
Fnd_Sort_AddField (->[Company]Company Name)
```

Fnd_Sort_AddTable

Fnd_Sort_AddTable (->table{; position})

Parameter	Type	Description
table	Pointer	A pointer to the table to add
position	Longint	Position in which to add the item (optional)

The *Fnd_Sort_AddTable* routine loads the visible, sortable fields for the specified table into the field pop-up menu.

For example, this will add all of the sortable fields for both the Contact and Company table to the pop-up menu in the Sort dialog:

```
Fnd_Sort_AddTable (->[Contact])
Fnd_Sort_AddTable (->[Company])
```

Fnd_Sort_Direction

Fnd_Sort_Direction ({direction}) ➜ Longint

Parameter	Type	Description
direction	Longint	Sort direction (1 or -1) (optional)
Function result	Longint	Current sort direction setting

This command allows the developer to set or get the currently selected sort direction for the current process. If a valid direction is specified, this routine will make it the default direction for subsequent sorts, until the user changes the sort direction setting.

Pass 1 for an ascending sort, or -1 for a descending sort.

```
Fnd_Sort_Direction (-1)  ` Descending sort.
```

This function also returns the sort direction set for the current process, so it can be stored for later use in another process or the next time the database is used.

Fnd_Sort_Display

Fnd_Sort_Display

Parameter	Type	Description
table	Pointer	Pointer to the table to use (optional)({->table})

This method displays Foundation's simple Sort dialog. If no table is specified, the table returned by the *Fnd_Gen_CurrentTable* function is used.

Fnd_Sort_Info

Fnd_Sort_Info (info requested) ➜ Text

Parameter	Type	Description
info requested	Text	Info desired
Function Result	Text	Response

Returns the requested information about the component.

```
$version_t:=Fnd_Sort_Info ("version")
```

The *Fnd_Sort_Info* method will respond to these requests:

Request	Response	Example
name	The component's full name	Foundation Sort
version	The component's version number	4.0

This routine can also be called using the *Fnd_Gen_ComponentInfo* method without first testing to see if the component is installed:

```
$version_t:=Fnd_Gen_ComponentInfo ("Fnd_Sort";"version")
```

See the *Fnd_Gen_ComponentInfo* method for more information.

Fnd_Sort_OrderByEditor

Fnd_Sort_OrderByEditor

Parameter	Type	Description
table	Pointer	Pointer to the table to use (optional)({->table})

This method displays 4D's built-in Order By dialog. If no table is specified, the table returned by the *Fnd_Gen_CurrentTable* function is used.

This routine calls 4D's **AUTOMATIC RELATIONS** command before displaying the Order By editor to enable automatic relations. **AUTOMATIC RELATIONS** is called again after the dialog is closed to restore the relations to their default state.

Fnd_Sort_SelectedField

Fnd_Sort_SelectedField ({->field}) ➜ Pointer

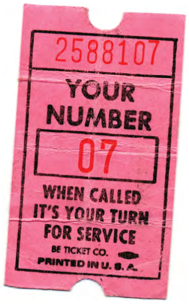
Parameter	Type	Description
field	Pointer	The default field setting (optional)
Function result	Pointer	Current default field

Allows the developer to set or get the currently selected sort field. If a field pointer is specified, this routine will make it the default selection if the field is available in the list. If not, nothing is changed and no error is displayed.

```
Fnd_Sort_SelectedField (->[Contact]Last Name)  ` Set the default sort field.
```

Sequence Numbers Component

(Fnd_SqNo)



Foundation's Sequence Numbers component gives you procedural control over sequential numbers in your database. It includes routines to get sequence numbers and to return those numbers that you have not used, but want to use later.

A Sequence Number Editor window is also included to allow the designer or the database administrator to edit the sequence numbers from the Custom Menus environment.

Installation

This section describes how to install the Fnd_SqNo component into a database that is not based on the Foundation Shell. If you started your project using the Foundation Shell, the Sequence Numbers component is already installed and integrated into your database.

The Sequence Numbers component stores information in a database table.

4D v11 does not allow tables in components. To work around this problem, the Foundation components do not include any 4D tables. Instead, you will need to add any required tables to your structure before installing the component. Internally the components create pointers to the structure's tables and fields and then uses these throughout the code.

Create the [Fnd_SqNo] Table

To use the Foundation Sequence Numbers component, you will first need to add a table to your structure. You can simply copy the [Fnd_SqNo] table from the Product Sales.4DB sample file, or you can create it manually in 4D (this is a good opportunity to reuse an unused or deleted table). The table must be named "Fnd_SqNo" and must contain the fields indicated below. The order of the fields is not important, and it is okay if the table contains other unused fields (in case you are reusing a table). However, the field names and types must be set up exactly as shown below.

Field Name	Type	Attributes
[Fnd_SqNo]ID	Long Integer	Indexed
[Fnd_SqNo]Group_Name	Alpha 80	Indexed
[Fnd_SqNo]Next_Number	Long Integer	
[Fnd_SqNo]Recycle_Bin	BLOB	
[Fnd_SqNo]Designer_Only	Boolean	Indexed

Install the Component

The Foundation Sequence Numbers component requires the following components (shown with minimum required version numbers). It also requires version 4.2 or later of the Foundation Extras plugin.

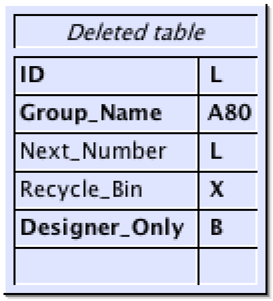
Component	Minimum Version
Fnd_SqNo	4.2
Fnd_Dlg	4.1.3
Fnd_Wnd	4.1.4

Updating the Component

If you later upgrade (or reinstall) the component, you will be asked if you want to update the public Fnd_SqNo_Preferences form. Click **No** to preserve any changes you may have made to this form. Click **Yes** only if you have not customized this form for your application, or if you wish to revert to the default layout.

If you are upgrading a database that currently has version 4.1.4 or earlier of the Foundation Sequence Numbers component, you will need to first uninstall that component, then install the 4.2 or later version. The 4.2 and later versions cannot directly update an existing older version of the component.

Removing the earlier version of the component (which included a table) will leave a table named "Deleted table" in your structure. After installing the new version of the component, launch 4D and switch to the Design environment (ignore the error messages that will be displayed). Change the name of the deleted table that looks like the one shown here back to "Fnd_SqNo." Then quit and relaunch 4D. This time the database should launch without errors.



Deleted table

ID	**L**
Group_Name	**A80**
Next_Number	**L**
Recycle_Bin	**X**
Designer_Only	**B**

Language Reference

Here is the list of routines in Foundation's Sequence Numbers component:

Fnd_Hook_SqNo_SetIDField
Fnd_SqNo_Editor
Fnd_SqNo_Enable
Fnd_SqNo_Fix
Fnd_SqNo_Get
Fnd_SqNo_Info
Fnd_SqNo_Put
Fnd_SqNo_Set
Fnd_SqNo_SetRecordID

Fnd_Hook_SqNo_SetIDField

Fnd_Hook_SqNo_SetIDField (->table) ➔ Pointer

Parameter	Type	Description
table	Pointer	Pointer to the table
Function result	Pointer	Pointer to the table's key field

Foundation's Sequence Number routines assume the first field for every table is the table's key field. If this is not the case, you can show Foundation which field to use for the key field using this hook.

The key field must be a long integer and must be indexed to work properly with the Sequence Numbers component.

This function may return a nil pointer if the table does not contain an ID field.

Fnd_SqNo_Editor

Fnd_SqNo_Editor

Parameter	Type	Description
No parameters required.		

This method displays Foundation's Sequence Number Editor dialog in a new process.

Fnd_SqNo_Enable

Fnd_SqNo_Enable (group name{; start number})

Parameter	Type	Description
group name	Text	Group name
start number	Longint	Starting number (optional)

This method creates the sequence number group if it does not already exist. Optionally, it starts the numbering at the specified number. Otherwise, it starts at 1.

The new group will be editable by the database administrator.

```
Fnd_SqNo_Enable ("Invoice Numbers";1001)
```

Fnd_SqNo_Fix

Fnd_SqNo_Fix (->table)

Parameter	Type	Description
table	Pointer	Pointer to the table to fix

If you ever have a problem with the sequence numbers for record IDs set by the *Fnd_SqNo_SetRecordID* command, this command can fix the problem. It determines the next valid record number for the specified table by analyzing the index file. It then sets the next sequence number for the table. The change is automatically saved by this command.

```
Fnd_SqNo_Fix (->[Contacts])
```

This action can also be performed by clicking the **Fix** button in the Sequence Number Editor dialog.

Fnd_SqNo_Get

Fnd_SqNo_Get (group name) ➔ Number

Parameter	Type	Description
group name	Text	The sequence number group name
Function result	Longint	Next number in the sequence

This function returns the next unique positive long integer based on the group name you pass. This function returns 0 if it is unable to get the next sequence number.

```
[Invoices]Invoice_Number:=Fnd_SqNo_Get ("Invoice Numbers")
If([Invoices]Invoice_Number=0)  ` We were unable to get a sequence number.
  ALERT("Unable to get a new invoice number for this record.")
  CANCEL
End if
```

If the group does not already exist, it is created and 1 is returned as the first number. You can use the *Fnd_SqNo_Enable* method to pre-define a group and set the starting number.

This method can safely be called from within a trigger or transaction.

If you decide you do not need the sequence number, you can return it to the queue using the *Fnd_SqNo_Put* method.

Fnd_SqNo_Info

Fnd_SqNo_Info (info requested) ➔ Text

Parameter	Type	Description
info requested	Text	Info desired
Function Result	Text	Response

Returns the requested information about the component.

```
$version_t:=Fnd_SqNo_Info ("version")
```

The *Fnd_SqNo_Info* method will respond to these requests:

Request	Response	Example
name	The component's full name	Foundation Sequence Numbers
version	The component's version number	4.0.3 beta 2

This routine can also be called using the *Fnd_Gen_ComponentInfo* method without first testing to see if the component is installed:

```
$version_t:=Fnd_Gen_ComponentInfo ("Fnd_SqNo";"version")
```

See the *Fnd_Gen_ComponentInfo* method for more information.

Fnd_SqNo_Put

Fnd_SqNo_Put (group name; number to return)

Parameter	Type	Description
group name	Text	The sequence number group name
number to return	Longint	The number to reuse

Call this method to return a number to the specified sequence number group. If the number is higher than the current next number, it will be ignored. It will also be ignored if it is already in the list of numbers to reuse, or if the group name does not already exist.

```
Fnd_SqNo_Put ("Invoice Numbers";1005)
```

This method always starts a new process to return the sequence number so the calling process does not have to wait for it to finish.

Fnd_SqNo_Set

Fnd_SqNo_Set (group name; next number)

Parameter	Type	Description
group name	Text	A sequence number group name
next number	Longint	New next number in the sequence

This routine sets the next number for the specified group, and deletes any numbers previously allocated for reuse. It can be used to fix problems with existing sequence numbers.

The number passed will be the starting number for the new group:

```
Fnd_SqNo_Set ("Invoice Numbers";1001)  ` Start new invoice numbers at 1001.
```

Unlike the *Fnd_SqNo_Enable* method, this routine updates the next number of the group even if it already exists.

Fnd_SqNo_SetRecordID

Fnd_SqNo_SetRecordID ({->table})

Parameter	Type	Description
table	Pointer	The table for which to return a number (optional)

Although you can use the *Fnd_SqNo_Get* function to get sequence numbers to be used as record IDs, we would like to suggest another method. The *Fnd_SqNo_SetRecordID* method is designed specifically to assign a value for a table. Pass it a pointer to the table for which you want it to assign a new number, and it will assign a unique value to the key field.

Pass this routine a pointer to the table for which you want it to set the unique record ID number. If no table is specified, either the current form table or the trigger table will be used. This method will check to see if the record already has an ID number. If it does not, this routine will assign it a number.

```
Case of
  : (Form event=On Load )
    Fnd_SqNo_SetRecordID
End case
```

This routine will call the *Fnd_Hook_SqNo_SetIDField* hook to determine which field is the key field for the table. It assumes you are using a long integer key field.

One of the benefits of using this routine rather than the *Fnd_SqNo_Get* function is this routine tracks the unique numbers by the table number, not by the table's name. So you can safely change the name of a table without updating any calls to this method. Sequence numbers assigned using this function can also be "fixed" using the **Fix** button in the Sequence Number Editor or by calling the *Fnd_SqNo_Fix* method.

You cannot return an unused ID number for reuse, since ID fields should not contain data that will be important to the end-users. By not returning numbers to the sequence number for reuse, there is less need to worry about causing delays for other users and processes wishing to access this function for the same table.

Text Component

Fnd_Text

The Text component provides utility routines for working with text. You can call these commands from anywhere in your database.

Language Reference

Here is the list of routines in Foundation's Text component:

- Fnd_Text_Base64ToText
- Fnd_Text_Capitalize
- Fnd_Text_CapitalizeExclude
- Fnd_Text_DecodeBase64Blob
- Fnd_Text_EncodeBase64Blob
- Fnd_Text_FormatNumber
- Fnd_Text_Info
- Fnd_Text_PadSpaces
- Fnd_Text_StripSpaces
- Fnd_Text_TextToBase64
- Fnd_Text_TextToMD5
- Fnd_Text_Wrap

Fnd_Text_Base64ToText

Fnd_Text_Base64ToText (text) ➜ Text

Parameter	Type	Description
text	Text	Text to convert from Base64 to ASCII text
Function result	Text	Decoded text

This function returns the ASCII text of the converted Base64 text.

```
$tASCII_t:=Fnd_Text_Base64ToText (tSomeBase64Text_t)
```

Fnd_Text_Capitalize

Fnd_Text_Capitalize (text{; style}) ➜ Text

Parameter	Type	Description
text	Text	Text to capitalize
style	Longint	Capitalization style (optional)
Function result	Text	Capitalized text

This function returns the capitalized text. If no optional style parameter is passed, Book Title capitalization is applied. If style parameter equals 2, the function will not use the excluded word list for capitalization. See *Fnd_Text_CapitalizeExclude* command.

```
$capitalized_t:=Fnd_Text_Capitalize ("please capitalize this phrase.")
```

Fnd_Text_CapitalizeExclude

Fnd_Text_CapitalizeExclude (-> exclude array)

Parameter	Type	Description
exclude array	Pointer	Pointer to text array of excluded words

The *Fnd_Text_CapitalizeExclude* method specifies the words that will not be capitalized using the *Fnd_Text_Capitalize* method. To get a list of words that will be excluded from capitalization, pass an empty text array. To set the words to exclude from the capitalization routine, pass an array with at least one element. To clear the list of words to exclude from the capitalization routines, pass an array with one element consisting of a blank string.

```
Fnd_Text_CapitalizeExclude (->excludedwords_at)
```

Fnd_Text_DecodeBase64Blob

Fnd_Text_DecodeBase64Blob (-> blob)

Parameter	Type	Description
blob	Pointer	Pointer to a blob containing Base64 text

The *Fnd_Text_DecodeBase64Blob* method converts a blob from Base64 to ASCII.

```
Fnd_Text_DecodeBase64Blob (->blob)
```

Fnd_Text_EncodeBase64Blob

Fnd_Text_EncodeBase64Blob (->blob{; line breaks?{; line break character(s)}})

Parameter	Type	Description
blob	Pointer	Pointer to a blob containing ASCII text
line breaks?	Boolean	Add line breaks (optional)
line break character	Text	2 characters for line break (optional)

The *Fnd_Text_EncodeBase64Blob* method converts a blob from ASCII text to Base64. If the optional line breaks is true, line breaks will be added to the blob. If the optional line break character is passed, it will be used as the line break character in the blob.

```
Fnd_Text_EncodeBase64Blob (->blob; true; "LF")
```

Fnd_Text_FormatNumber

Fnd_Text_FormatNumber (string, format) ➜ Text

Parameter	Type	Description
string	Text	Numeric string to format as text
format	Text	Format to use
Function result	Text	Formatted number as text

This function returns formatted string of the numeric string removing any dashes, "E"s, and periods.

```
$formattednumber_t:=Fnd_Text_FormatNumber ("123.45", "$###.##")
```

Fnd_Text_Info

Fnd_Text_Info (info requested) ➜ Text

Parameter	Type	Description
info requested	Text	Info desired
Function Result	Text	Response

Returns the requested information about the component.

```
$language_t:=Fnd_Text_Info (“language”)
```

The Fnd_Text_Info method will respond to these requests:

Request	Response	Example
name	The component’s full name	Foundation Text
version	The component’s version number	4.0.3 beta 1
language	The currently selected language code	EN

This routine can also be called using the *Fnd_Gen_ComponentInfo* method without first testing to see if the component is installed:

```
$language_t:=Fnd_Gen_ComponentInfo ("Fnd_Text";"language")
```

See the *Fnd_Gen_ComponentInfo* method for more information.

Fnd_Text_PadSpaces

Fnd_Text_PadSpaces (text; length)

Parameter	Type	Description
text	Text	The text to pad with spaces
length	Longint	The desired length
Function result	Text	Modified text with spaces

This function pads the passed text with spaces to reach the desired length. If the length of the text is greater than the length parameter, the text will be truncated.

```
$padded_t:=Fnd_Text_PadSpaces ("sometext"; 10)
```

Fnd_Text_StripSpaces

Fnd_Text_StripSpaces (text{; options}) ➔ Text

Parameter	Type	Description
text	Text	Text to strip spaces from
options	Longint	Options to remove spaces (optional)
Function result	Text	Text with spaces stripped

This function returns text with spaces removed. If options = 1, the function will remove only leading spaces. If options = 2, the function will remove only trailing spaces. If options = 4, the function will remove only double spaces within the text.

```
$stripped_t:=Fnd_Text_StripSpaces ("  strip   spaces  ")
```

Fnd_Text_TextToBase64

Fnd_Text_TextToBase64 (text) ➔ Text

Parameter	Type	Description
text	Text	Text to encode in Base64
Function result	Text	Encoded Base64 text

This function returns Base64 encoded text.

```
$base64_t:=Fnd_Text_TextToBase64 (text_t)
```

Fnd_Text_TextToMD5

Fnd_Text_TextToMD5 (text) ➔ Text

Parameter	Type	Description
text	Text	Text to convert to MD5 message digest
Function result	Text	MD5 message digest of text parameter

This function returns MD5 message digest of the text parameter. See RFC 1321 for more information <http://www.faqs.org/rfcs/rfc1321.html>.

```
$MD5message_t:=Fnd_Text_TextToMD5 (text_t)
```

Fnd_Text_Wrap

Fnd_Text_Wrap (text; max line length) ➜ Text

Parameter	Type	Description
text	Text	Text to wrap
max line length	Longint	Max line length to wrap
Function result	Text	Text wrapped to max line length

This function returns text wrapped to the max line length. Routine by Marco Bernasconi <mbernasconi@befund.com>.

```
$wrappedtext_t:=Fnd_Text_Wrap (text_t; 78)
```

Toolbar Component

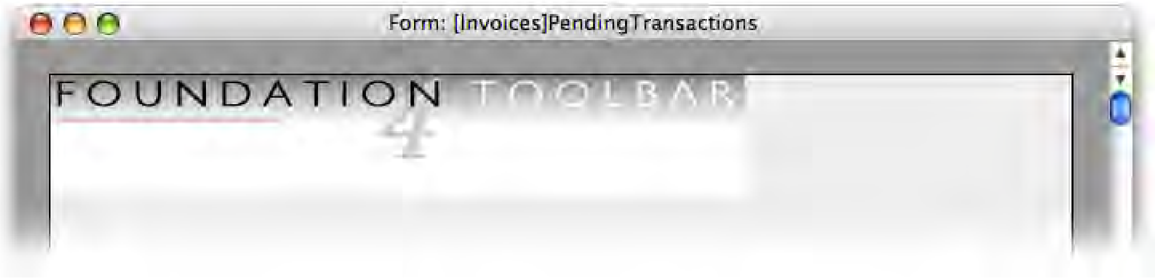
(Fnd_Tlbr)

The Toolbar Component allows you to display a standard Macintosh or Windows toolbar in your forms by inheriting the Fnd_Tlbr_Toolbar form. You can procedurally control the toolbar's icons, labels, and associated methods using the methods listed in this chapter.

This component is based on Mark Mitchenall's free Toolbar Component (http://www.Mitchenall.com/) and is used with permission.

To add a toolbar to your form, simply inherit the Fnd_Tlbr_Toolbar form from your own form. See the 4D Design Reference for more information about using inherited forms.

The inherited form will look similar to this in the 4D Design environment:



To specify the icons and labels to be displayed in the toolbar, call the Fnd_Tlbr_Button_Add method in the form's On Load phase for each button:

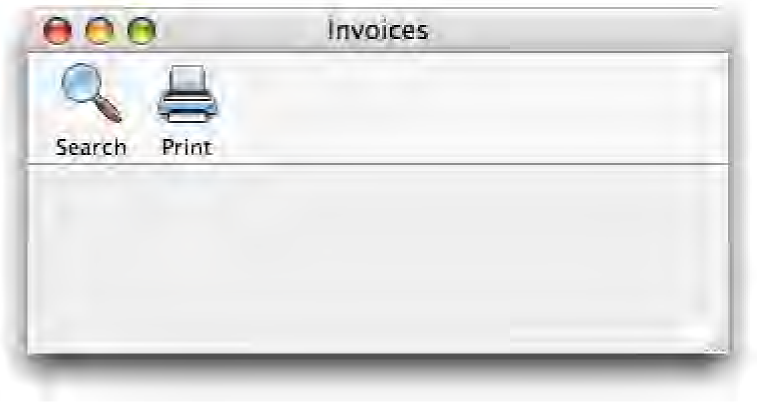
```
Case of
  : (Form event=On Load )
    Fnd_Tlbr_Button_Add ("Button1";"Search";"Fnd_Bttn_Magnifier";"QUERY([Invoices])")
    Fnd_Tlbr_Button_Add ("Button2";"Print";"Fnd_Bttn_Printer";"PrintTransactionsReport")
End case
Fnd_Tlbr_FormMethod
```

The first parameter is a name for the button, the second parameter is the button label, the third parameter is the name of the icon to display, and the last parameter is the code to execute when the button is clicked.

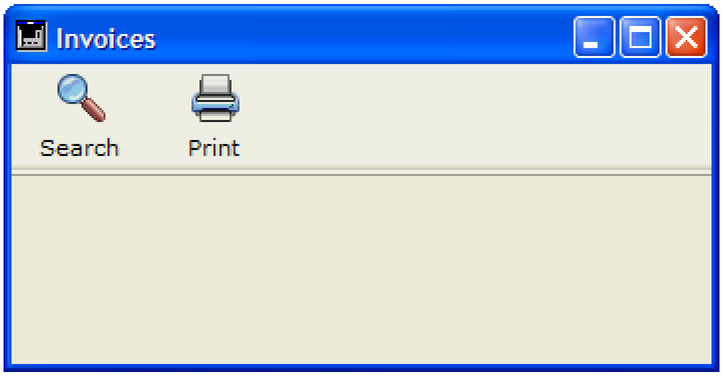
You will also need to call the *Fnd_Tlbr_FormMethod* from your form method, and activate the On Load, On Activate, On Deactivate, and On Resize form events. This method should generally be called after the form's On Load phase, so that it is called after setting up the toolbar buttons with *Fnd_Tlbr_Button_Add*.

If you are using the complete Foundation Shell, you may also want to call the *Fnd_Gen_FormMethod* routine from the form method. This can be called any time before or after the *Fnd_Tlbr_FormMethod* call.

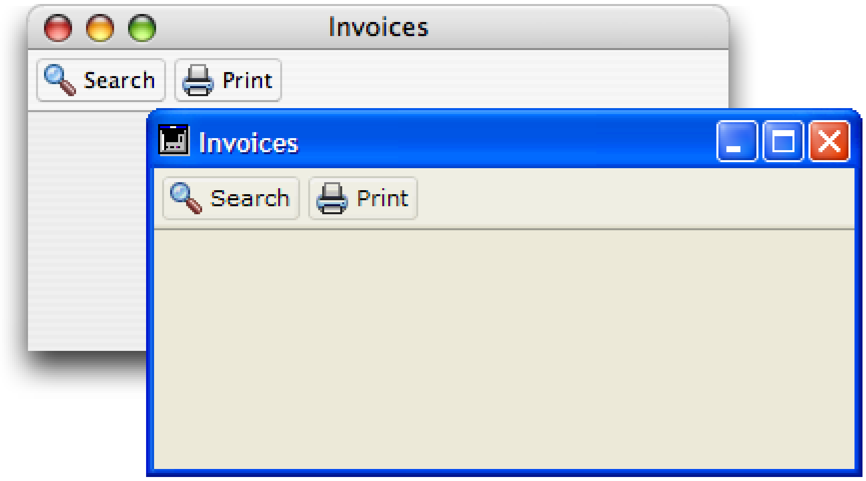
Here is the toolbar the above code will generate when viewed on a Macintosh:



When the same code is run on Windows, the toolbar will look like this:



By default, the "Large" toolbar style is used. The toolbar can also be displayed in a smaller size, with the labels to the right of the icons. See the *Fnd_Tlbr_Style* function for more information.



Any of the icons contained in the Foundation Buttons component can be used when building the toolbar. See the Buttons Component chapter for the complete list of available icons.

Language Reference

Here is the list of routines in the Foundation Toolbar component:

Fnd_Tlbr_Button_Add
Fnd_Tlbr_Button_Enabled
Fnd_Tlbr_Button_Icon
Fnd_Tlbr_Button_Label
Fnd_Tlbr_Button_Menu
Fnd_Tlbr_Button_Method
Fnd_Tlbr_Button_Remove
Fnd_Tlbr_Clear
Fnd_Tlbr_Divider_Add
Fnd_Tlbr_FormMethod
Fnd_Tlbr_Info
Fnd_Tlbr_Redraw
Fnd_Tlbr_StatusMessage
Fnd_Tlbr_Platform
Fnd_Tlbr_Style

Fnd_Tlbr_Button_Add

Fnd_Tlbr_Button_Add (name; button text; icon name; method{; keystroke})

Parameter	Type	Description
name	Text	Unique name for the button object
button text	Text	Button label
icon name	Text	Picture Library name for the icon
method	Text	The method to call when the button is clicked
keystroke	Text	Command/Ctrl keyboard equivalent (optional)

Fnd_Tlbr_Button_Add adds a new button to the toolbar. The button is always appended to the right of any existing toolbar buttons or dividers.

The button name is used to identify the button when calling other Toolbar component methods. It should be a unique string of up to 31 characters.

The second parameter is the text label to be displayed with the button. This label can later be changed on-the-fly by calling the *Fnd_Tlbr_Button_Label* function. This label is limited to 80 characters.

The third parameter is the name of an icon for the new button. Any of the icons built into the Foundation Buttons component can be used. Your own custom icons can also be used. See the Buttons Component (Fnd_Bttn) chapter for the complete list of built-in icons and instructions for using custom icons.

The fourth parameter is the name of a method or any string that will work properly with 4D's **EXECUTE** command. This string will be executed if the button is clicked.

For example, this code will add a button to the toolbar with a label of "Delete" and the Red X icon from the Buttons component. It will call the *DeleteRecords* method when clicked:

```
Fnd_Tlbr_Button_Add ("Button1";"Delete";"Fnd_Bttn_RedX";"DeleteRecords")
```

This line will have the same appearance, but will call 4D's **DELETE RECORDS** command directly:

```
Fnd_Tlbr_Button_Add ("DelBttn";"Delete";"Fnd_Bttn_RedX";"DELETE RECORDS([Invoices])")
```

Note that although the example above may work, it is best to use 4D's **Command name** function to get the localized command name, and to get the table name dynamically, rather than hard-coding it:

```
$execute_t:=Command name(58)+"("+Table name(->[Invoices])+"])"
Fnd_Tlbr_Button_Add ("DelBttn";"Delete";"Fnd_Bttn_RedX";$execute_t)
```

You can optionally assign a key equivalent to the button. The user can then use this keyboard equivalent rather than use the mouse to click the button. On Macintosh, the user will hold down the Command key and press the specified key. On Windows, the Control key is used. This key can be any letter A-Z, any number 0-9, or one of these special values:

Backspace
Delete
Period
UpArrow
DownArrow
RightArrow
LeftArrow

For example, this line lets the user run the *DuplicateRecord* method by typing Command-D on Macintosh or Control-D on Windows:

```
Fnd_Tlbr_Button_Add ("DupBttn";"Dupe Rec";"Fnd_Bttn_Duplicate";"DuplicateRecord";"D")
```

This line lets the user delete records by typing Command-Backspace on Macintosh or Control-Backspace on Windows:

```
Fnd_Tlbr_Button_Add ("DelBttn";"Delete";"Fnd_Bttn_RedX";"DeleteRecords";"Backspace")
```

After calling this command, the *Fnd_Tlbr_Redraw* command should be called to update the toolbar.

Fnd_Tlbr_Button_Count

Fnd_Tlbr_Button_Count ➔ Number

Parameter	Type	Description
Function result	Number	The number of buttons

This function returns the number of buttons displayed in the toolbar.

```
$buttonCount_i:=Fnd_Tlbr_Button_Count
```

Fnd_Tlbr_Button_Enabled

Fnd_Tlbr_Button_Enabled (button name; state)

Parameter	Type	Description
button name	Text	Name of the button to modify
state	Boolean	True to enable the button

Use this routine to enable or disable a toolbar button. Pass True to enable the button or False to disable the button.

```
Fnd_Tlbr_Button_Enabled ("PrintBttn";False)  ` Disable the Print button.
```

It is not necessary to call the *Fnd_Tlbr_Redraw* command after enabling or disabling a toolbar button.

Fnd_Tlbr_Button_Icon

Fnd_Tlbr_Button_Icon (button name{; icon name}) ➔ Text

Parameter	Type	Description
button name	Text	Name of the button to modify

icon name	Text	Name of the picture to use as the icon
Function result	Text	The current button icon name

Call this method to specify the name of a Button component icon to use as the button's icon.

```
Fnd_Tlbr_Button_Icon ("WizardBttn";"Fnd_Bttn_Wand")
```

You can specify the name of a built-in icon or a custom icon. See the Buttons Component (Fnd_Bttn) chapter for the complete list of available icons.

If this method is called after the On Load event, it should be followed by a call to the *Fnd_Tlbr_Redraw* command.

This routine can also be called as a function to get the button's current icon:

```
$icon_t:=Fnd_Tlbr_Button_Icon ("WizardBttn")
```

Fnd_Tlbr_Button_Insert

Fnd_Tlbr_Button_Insert (before button; name; button text; icon name; method{; keystroke})

Parameter	Type	Description
before button	Text	The button in the position to insert the new button
name	Text	Unique name for the button object
button text	Text	Button label
icon name	Text	Picture Library name for the icon
method	Text	The method to call when the button is clicked
keystroke	Text	Command/Ctrl keyboard equivalent (optional)

Use this method to insert a button in the toolbar. This command is identical to the *Fnd_Tlbr_Button_Add* routine, except it has an additional parameter (before button) that lets you specify the name of the button where the new toolbar button should be inserted.

```
If (Current user="Designer")
  Fnd_Tlbr_Button_Insert ("PrintBttn";"ConfBttn";"Configure";"Fnd_Bttn_
Gear";"ConfigMethod")
End if
```

Fnd_Tlbr_Button_Label

Fnd_Tlbr_Button_Label (button name{; label}) ➔ Text

Parameter	Type	Description
button name	Text	Name of the button to modify
label	Text	New button label (optional)
Function result	Text	The current button label

This method sets the label for the specified toolbar button. The label should be no longer than 80 characters.

```
Fnd_Tlbr_Button_Label ("WizardBttn";"Setup Wizard")
```

This routine can also be called as a function to get the button's current label:

```
$label_t:=Fnd_Tlbr_Button_Label ("WizardBttn")
```

If this method is called after the On Load event, it should be followed by a call to the *Fnd_Tlbr_Redraw* command.

Fnd_Tlbr_Button_Menu

Fnd_Tlbr_Button_Menu (button name; ->menu items)

Parameter	Type	Description
button name	Text	The button name
menu items	Pointer	Text array of menu items

Use this command to turn a toolbar button into a drop-down menu. First create the menu using the *Fnd_Tlbr_Button_Add* method. Then create a text array of menu items and call *Fnd_Tlbr_Button_Menu*.

```
$method_t:="PrintRoutine(<FndMenuNumber>)"
Fnd_Tlbr_Button_Add ("PrintBttn";"Print");"Fnd_Bttn_Printer";$method_t)
ARRAY TEXT(PrintOptions_at;4)
PrintOptions_at{1}:="Sales Report"
PrintOptions_at{2}:="-"  ` Add a divider line.
PrintOptions_at{3}:="Report Editor"
PrintOptions_at{4}:="Label Editor"
Fnd_Tlbr_Button_Menu ("PrintBttn";->PrintOptions_at)
```

You will need to use a process array with 4D 2003. A local array can be used with 4D 2004. The array can be cleared immediately after the call.

In the example above, "<FndMenuNumber>" will be replaced with the selected menu item number, then the entire value of the $method_t variable will be executed. So if the Report Editor menu item is selected, the *PrintRoutine* method will be passed a value of 3 as the first parameter.

Then, in the *PrintRoutine* method, we should use a compiler directive to indicate that the first parameter is a number:

```
  ` Project Method: PrintRoutine (menu item)
C_LONGINT($1;$selectedMenuItem_i)
$selectedMenuItem_i:=$1
...
```

The following strings are automatically replaced at runtime with the appropriate values:

String	Value
<FndButtonName>	The name of the button

<FndButtonLabel>	The button's label
<FndMenuNumber>	The selected menu item number
<FndMenuLabel>	The selected menu item label

Note that when using the name or label strings, you will need to include the quotation marks to pass the value as a text value to your button method. This is easily done using the \" escape sequence. For example:

```
$method_t:="PrintRoutine(\"<FndMenuLabel>\")"
Fnd_Tlbr_Button_Add ("PrintBttn";"Print");"Fnd_Bttn_Printer";$method_t)
```

You can pass any combination of these values and other values to your button method.

The “<FndButtonName>” and “<FndButtonLabel>” strings can be used for any toolbar button style. The “<FndMenuNumber>” and “<FndMenuLabel>” strings will be converted only for menu buttons.

Fnd_Tlbr_Button_Method

Fnd_Tlbr_Button_Method (button name{; method name}) ➔ Text

Parameter	Type	Description
button name	Text	Name of the button to modify
method name	Text	Method to execute (optional)
Function result	Text	The current button method

Call this routine to set the associated method for the specified toolbar button. The method will be executed when the button is clicked.

```
Fnd_Tlbr_Button_Method ("WizardBttn";"WizardDialog")
```

This routine can also be called as a function to get the button’s current method:

```
$methodName_t:=Fnd_Tlbr_Button_Method ("WizardBttn")
```

See the *Fnd_Tlbr_Button_Add* routine for more information about associating methods with toolbar buttons.

Fnd_Tlbr_Button_Name

Fnd_Tlbr_Button_Name (index) ➔ Number

Parameter	Type	Description
index	Number	The number of the button
Function result	Text	The name of the button

This function returns the name of the button at the specified position.

For example, to get the name of the last button:

```
$lastButton_i:=Fnd_Tlbr_Button_Count
$buttonName_t:=Fnd_Tlbr_Button_Name ($lastButton_i)
```

Fnd_Tlbr_Button_Remove

Fnd_Tlbr_Button_Remove (button name)

Parameter	Type	Description
button name	Text	Name of the button to modify

This command removes the button with the specified name from the toolbar.

```
Fnd_Tlbr_Button_Remove ("WizardBttn")
```

If this method is called after the On Load event, it should be followed by a call to the *Fnd_Tlbr_Redraw* command.

Fnd_Tlbr_Clear

Fnd_Tlbr_Clear

Parameter	Type	Description
No parameters required.		

Fnd_Tlbr_Clear removes all objects from the toolbar so it can be rebuilt from scratch.

```
Fnd_Tlbr_Clear
Fnd_Tlbr_Button_Add ("Button1";"Search";"Fnd_Bttn_Magnifier";"QUERY([Invoices])")
Fnd_Tlbr_Button_Add ("Button2";"Print";"Fnd_Bttn_Printer";"PrintTransactionsReport")
Fnd_Tlbr_Redraw
```

If this method is called after the On Load form event has run, it should be followed by a call to the *Fnd_Tlbr_Redraw* command.

Fnd_Tlbr_Divider_Add

Fnd_Tlbr_Divider_Add

Parameter	Type	Description
No parameters required.		

Fnd_Tlbr_Divider_Add adds a divider line to the toolbar object list. The divider is added after any other objects in the toolbar.

```
Fnd_Tlbr_Button_Add ("Button1";"Search";"Fnd_Bttn_Magnifier";"QUERY([Invoices])")
Fnd_Tlbr_Divider_Add
Fnd_Tlbr_Button_Add ("Button2";"Print";"Fnd_Bttn_Printer";"PrintTransactionsReport")
Fnd_Tlbr_Redraw
```

Each toolbar is limited to 20 objects. Each divider counts as one object toward this limit. So, if you create a toolbar with 17 buttons, it could include only three dividers.

If this method is called after the On Load form event has run, it should be followed by a call to the *Fnd_Tlbr_Redraw* command.

Fnd_Tlbr_FormMethod

Fnd_Tlbr_FormMethod

Parameter	Type	Description
No parameters required.		

This method must be called from any form method that inherits the toolbar form. *Fnd_Tlbr_FormMethod* expects the On Load, On Activate, On Deactivate, and On Resize form events to be enabled.

```
Fnd_Tlbr_FormMethod
```

This method may be called in addition to the *Fnd_Gen_FormMethod*, *Fnd_IO_Info*, or *Fnd_IO_OutputFormMethod* routines. The order in which these calls are made is not important.

Fnd_Tlbr_Info

Fnd_Tlbr_Info (info requested) ➔ Text

Parameter	Type	Description
info requested	Text	Info desired
Function Result	Text	Response

Returns the requested information about the component.

```
$version_t:=Fnd_Tlbr_Info ("version")
```

The *Fnd_Tlbr_Info* method will respond to these requests:

Request	Response	Example
name	The component's full name	Foundation Toolbar
version	The component's version number	4.0.5 beta 2

This routine can also be called using the *Fnd_Gen_ComponentInfo* method without first testing to see if the component is installed:

```
$version_t:=Fnd_Gen_ComponentInfo ("Fnd_Tlbr";"version")
```

See the *Fnd_Gen_ComponentInfo* method for more information.

Fnd_Tlbr_Redraw

Fnd_Tlbr_Redraw

Parameter	Type	Description
No parameters required.		

This command updates the toolbar. It should be called after making any changes to the toolbar.

```
Fnd_Tlbr_Button_Add ("UnlockButton";"Unlock";"Fnd_Bttn_Unlock";"UnlockMethod")
Fnd_Tlbr_Redraw
```

The Toolbar component does not update the toolbar immediately when buttons are added or modified. Instead, it waits until this command is called so that multiple objects can be added or modified before the form update.

This command does not need to be called in the form's On Load event. It only needs to be called when updating the toolbar after it has been displayed. It is also not necessary to call this command after using the *Fnd_Tlbr_StatusMessage* or *Fnd_Tlbr_Button_Enabled* routines.

Fnd_Tlbr_StatusMessage

Fnd_Tlbr_StatusMessage ({new text}) ➔ Text

Parameter	Type	Description
new text	Text	Text to set for the info line (optional)
Function result	Text	Text displayed in the info line

Fnd_Tlbr_StatusMessage allows for the getting and setting of the information text line displayed in the standard toolbar output form. Pass this routine the message to display.

```
Fnd_Tlbr_StatusMessage ("There are "+String(Records in selection([People]))+" People
records.")
```

If this command is not called during the form's On Load phase, or if it is passed a blank string, then no status bar will be visible.

The status message can be updated at any time. It is not necessary to call the *Fnd_Tlbr_Redraw* command after calling this routine to update the status message. However, if the status bar is hidden and text has been passed to this routine to display it, or to hide a visible status bar, *Fnd_Tlbr_Redraw* should be called after calling *Fnd_Tlbr_StatusMessage*.

This routine can also be called as a function to get the current content of the status bar:

```
$displayedMessage_t:=Fnd_Tlbr_StatusMessage
```

Fnd_Tlbr_Platform

Fnd_Tlbr_Platform ({platform name}) ➔ Text

Parameter	Type	Description
platform name	Text	"Mac," "Win" or "Auto" (optional)
Function result	Text	Selected platform name

This routine sets and gets the toolbar platform name for the current process. This affects the general toolbar appearance. The following platform names can be used:

Platform Name
Auto
Mac
Win

Pass this routine one of the above platform names to set the toolbar appearance:

```
Fnd_Tlbr_Platform ("Mac")
```

This can be done in the form's On Load event, or anytime while the form is displayed. If you call this method after the On Load form event, you will need to follow it with a call to the *Fnd_Tlbr_Info* command.

Passing "Mac" will cause the toolbar to be drawn similar to the toolbar in other Mac OS X applications. Passing "Win" will cause the toolbar to be drawn similarly to toolbars in Windows applications. The actual platform in use is not related to the affect of this command — the "Mac" style can be used on Windows and the "Win" style can be used on the Macintosh.

Pass "Auto" to this routine to allow it to automatically select the appropriate setting for the current platform.

The actual look of the toolbar will depend not only on the current toolbar platform setting, but also on the current style setting. See the *Fnd_Tlbr_Style* command for more information.

This routine can also be called as a function to get the current toolbar platform setting.

```
$platform_t:=Fnd_Tlbr_Platform
```

This routine will return only "Mac" or "Win." It will not return "Auto."

Fnd_Tlbr_Style

Fnd_Tlbr_Style ({style name}) ➔ Text

Parameter	Type	Description
style name	Text	A pre-programmed style name (optional)
Function result	Text	Selected style name

Fnd_Tlbr_Style sets and gets the toolbar style name for the current process. The following style names can be used:

Style Name

Large

Small1

Small2

Pass this routine a style name to set the toolbar style name:

```
Fnd_Tlbr_Style ("Small2")
```

This can be done in the form's On Load event, or anytime while the form is displayed. If you call this method after the On Load form event, you will need to follow it with a call to the *Fnd_Tlbr_Redraw* command.

The actual look of the toolbar will depend not only on the current toolbar style, but also on the current platform setting (see the *Fnd_Tlbr_Platform* method).

Virtual Structure Component

Fnd_VS

The Virtual Structure component lets you present custom field and table names to your end-users that are not equal to the names assigned in the Design environment. All of the Foundation routines that display a field or table name get the name to display using this component, rather than directly calling 4D's Table name and Field name functions.

Language Reference

Here is the list of routines in Foundation’s Virtual Structure component:

- Fnd_VS_CreateNameLists
- Fnd_VS_FieldName
- Fnd_VS_GetFields
- Fnd_VS_GetTables
- Fnd_VS_Info
- Fnd_VS_ReplaceString
- Fnd_VS_SetFieldTitle
- Fnd_VS_SetFieldTitles
- Fnd_VS_SetTableTitle
- Fnd_VS_SetTableTitles
- Fnd_VS_TableName
- Fnd_VS_UseNameLists

Fnd_VS_CreateNameLists

Fnd_VS_CreateNameLists

Parameter	Type	Description
No parameters required.		

This routine will create a set of 4D lists that can be used by the virtual structure routines. This routine is called from Foundation's developer menu.

After calling this method, you will see a confirmation dialog:

Click the Continue button, and a dialog will ask you to enter a two-character language code for the new lists:

Enter the language code of your choice, and then Foundation will generate a 4D list for each table in the database. The lists will be named VS_Table001_XX, VS_Table002_XX, VS_Table003_XX, etc. The "XX" will actually be the code you entered in the dialog. The numeric part of the name represents the table number in the structure. So VS_Table001_XX contains the table and field names for the first table, the list named VS_Table002_XX contains the table and field names for the second table, and so on. By referencing the list names by number rather than by name, you can safely change the name of a 4D table without affecting these lists.

Once the lists are created, you can edit any of the table or field names by editing the lists using the 4D List editor. Then, add a call to the *Fnd_VS_UseNameLists* method in your startup routine. This will cause the modified table and field names to be used by the other routines in this component. The new names will also be passed to 4D's **SET TABLE TITLES** and **SET FIELD TITLES** commands, so the new names will be used in the 4D editors.

When you add new tables or fields to the structure, you can call this command again to update these lists. Your existing changes will not be touched, so it is safe to run this command at any time. The routine will only add missing table or field names.

You can safely delete any of these lists - if they are not found when the *Fnd_VS_UseNameLists* method is called, then the actual 4D list names will be used instead.

Fnd_VS_FieldName

Fnd_VS_FieldName(->field) ➜ Text

Parameter	Type	Description
field	Pointer	Pointer to a field
Function result	Text	Virtual field name

This routine returns the virtual field name of the specified field.

```
ALERT("Please enter a value in the "+Fnd_VS_FieldName (->[Contact]Last Name)+" field.")
```

This function is designed to be a replacement for 4D's **Field name** function.

Fnd_VS_GetFields

Fnd_VS_GetFields (->table;->pointer array{; ->text array})

Parameter	Type	Description
table	Pointer	Pointer to a table
pointer array	Pointer	Pointer to a pointer array to receive the field pointers
text array	Pointer	Pointer to a text array to receive the field names (optional)

This method returns in the passed arrays the pointers to and the field names of all of the visible fields for the specified table.

```
ARRAY POINTER(aFieldPtrs;0)
ARRAY TEXT(aFieldNames;0)
Fnd_VS_GetFields (->[Contacts];->aFieldPtrs;->aFieldNames)
```

Fnd_VS_GetTables

Fnd_VS_GetTables (->pointer array{; ->text array})

Parameter	Type	Description
pointer array	Pointer	Pointer to a pointer array to receive the table pointers
text array	Pointer	Pointer to a text array to receive the table names (optional)

This method returns in the passed arrays the names of and pointers to all of the visible tables.

It returns the arrays already sorted in the order the developer wants them sorted.

```
ARRAY POINTER(aTablePtrsArray;0)
ARRAY TEXT(aTableNamesArray;0)
Fnd_VS_GetTables (->aTablePtrsArray;->aTableNamesArray)
```

Fnd_VS_Info

Fnd_VS_Info (info requested) ➜ Text

Parameter	Type	Description
info requested	Text	Info desired
Function Result	Text	Response

Returns the requested information about the component.

```
$version_t:=Fnd_VS_Info ("version")
```

The *Fnd_VS_Info* method will respond to these requests:

Request	Response	Example
name	The component's full name	Foundation Virtual Structure
version	The component's version number	4.0.5 beta 2

This routine can also be called using the *Fnd_Gen_ComponentInfo* method without first testing to see if the component is installed:

```
$version_t:=Fnd_Gen_ComponentInfo ("Fnd_VS";"version")
```

See the *Fnd_Gen_ComponentInfo* method for more information.

Fnd_VS_ReplaceString

Fnd_VS_ReplaceString (old string; new string)

Parameter	Type	Description
old string	Text	String to replace
new string	Text	String to replace old string

Fnd_VS_ReplaceString does a find and replace in all of the virtual table titles and field titles.

This routine was designed specifically to quickly replace underscores with spaces, but could also be used for other global changes. For example, after this call:

```
Fnd_VS_ReplaceString ("_";" ")
```

A field named "First_Name" would appear to the user as "First Name."

Fnd_VS_SetFieldTitle

Fnd_VS_SetFieldTitle (->field; title)

Parameter	Type	Description
field	Pointer	Field to rename
title	Text	Virtual field title

Fnd_VS_SetFieldTitle allows the developer to set a virtual field title for a single field.

```
Fnd_VS_SetFieldTitle (->[Contact]Addr1;"Address Line 1")
```

Fnd_VS_SetFieldTitles

Fnd_VS_SetFieldTitles (->table; ->field titles; ->field numbers)

Parameter	Type	Description
table	Pointer	Table to use
field titles	Pointer	Pointer to array of field titles
field numbers	Pointer	Pointer to array of field numbers

Fnd_VS_SetFieldTitles allows the developer to set virtual field titles. This method is designed to be called as a replacement to 4D's **SET FIELD TITLES** command. Foundation will pass this information on to 4D, but will also store the information for use by the other Virtual Structure routines.

Fnd_VS_SetTableTitle

Fnd_VS_SetTableTitle (->table; title)

Parameter	Type	Description
table	Pointer	Table to rename
title	Text	Virtual table title

Fnd_VS_SetTableTitle allows the developer to set a virtual table title for a single table.

```
Fnd_VS_SetTableTitle (->[Contact];"Customers")
```

Fnd_VS_SetTableTitles

Fnd_VS_SetTableTitles (->table titles; ->table numbers)

Parameter	Type	Description
table titles	Pointer	Pointer to array of table titles
table numbers	Pointer	Pointer to array of table numbers

Fnd_VS_SetTableTitles allows the developer to set virtual table titles. This routine is designed to be called as a replacement to 4D's **SET TABLE TITLES** command.

Fnd_VS_TableName

Fnd_VS_TableName (->table) ➜ Text

Parameter	Type	Description
table	Pointer	Pointer to a table
Function result	Text	Virtual field name

Fnd_VS_TableName returns the virtual table name of the specified table. It is designed to replace calls to 4D's **Table name** function.

```
$tableName:=Fnd_VS_TableName (->[Contact])
ALERT("Are you sure you want to delete this "+$tableName+" record?")
```

Fnd_VS_UseNameLists

Fnd_VS_UseNameLists

Parameter	Type	Description
No parameters required.		

After creating a set of virtual structure name lists using the *Fnd_VS_CreateNameLists* command, call this command to use a set of virtual structure name lists. This command will use the lists that match Foundation's current language.

To use a language code other than English, install the Foundation Localization component and use the *Fnd_Loc_LanguageCode* method to set another language code.

Windows Component

(Fnd_Wnd)

The Foundation Windows component give you tools for creating and managing windows.

This component is used by many of the other components that display windows. If a component uses the Windows component, then you can usually call any of the Windows component routines to affect the new window before calling the component that displays a window.

For example, to set the window title of the List component's Command Dialog, call the Windows component's *Fnd_Wnd_Title* routine before calling *Fnd_List_CommandDialog* method:

```
Fnd_List_AddToListEditor ("Edit Country Names")
Fnd_Wnd_Title ("Configuration")
Fnd_List_CommandDialog
```

Most of the Windows component routines are limited to the process in which they are called. For example, you cannot use *Fnd_Wnd_Title* to set the title of a window that will be displayed by a new process.

Language Reference

Here is the list of routines in Foundation's Windows component:

Fnd_Wnd_CancelCloseAll
Fnd_Wnd_CloseAllWindows
Fnd_Wnd_CloseBox
Fnd_Wnd_CloseNow
Fnd_Wnd_Info
Fnd_Wnd_MoveOnScreen
Fnd_Wnd_OpenFormWindow
Fnd_Wnd_OpenWindow
Fnd_Wnd_Position
Fnd_Wnd_SavePosition
Fnd_Wnd_SendCloseRequests
Fnd_Wnd_SetCloseBox
Fnd_Wnd_SetPosition
Fnd_Wnd_SetTitle
Fnd_Wnd_SetType
Fnd_Wnd_Title
Fnd_Wnd_Type
Fnd_Wnd_UseSavedPosition

Fnd_Wnd_CancelCloseAll

Fnd_Wnd_CancelCloseAll

Parameter	Type	Description
No parameters required.		

Call this method to end Foundation's attempts to close all windows. For example, if the user selects **Close All Windows** from the **Window** menu, and a dialog asks if they want to save changes, and **Cancel** is selected, call this so no more windows are closed.

```
If (Fnd_Gen_QuitNow)
  Fnd_Dlg_Confirm ("Are you sure you want to quit without saving?")
  If (OK=0)
    Fnd_Wnd_CancelCloseAll
  End if
End if
```

Fnd_Wnd_CloseAllWindows

Fnd_Wnd_CloseAllWindows ({wait?}) ➜ Boolean

Parameter	Type	Description
wait?	Boolean	Wait for all windows to close? (optional)
Function result	Boolean	Did all windows close?

Call this method to tell all of the other processes to close their windows. If the **wait** parameter is passed and is **True**, this process will wait until all of the windows have been closed, and then return **True**. If it takes too long for the windows to close, or the user cancels the closing, then **False** is returned.

Do not pass **True** to this routine from a process that is displaying a window that needs to be closed. It will not work properly, since it will not be possible to close the window of the current process while this method is running.

Fnd_Wnd_CloseBox

Fnd_Wnd_CloseBox ({display close box?}) ➔ Boolean

Parameter	Type	Description
display close box?	Boolean	Display the close box? (optional)
Function result	Boolean	True if the next window will have a close box

Call this routine to add a close box to the next window displayed by the *Fnd_Wnd_OpenWindow* or *Fnd_Wnd_OpenFormWindow* commands.

```
Fnd_Wnd_CloseBox (True)
Fnd_Wnd_OpenFormWindow (->[Fnd_Forms];"ConfigureScanner")
```

This routine will affect only the next window in the current process.

This routine has replaced the *Fnd_Wnd_SetCloseBox* method.

Fnd_Wnd_CloseNow

Fnd_Wnd_CloseNow ➔ Boolean

Parameter	Type	Description
No parameters required.		
Function result	Boolean	True if it is time to close the window

This method returns **True** if Foundation is trying to close all windows. For example, this routine will return **True** if the user Option-clicks (on Macintosh) or Alt-clicks (on Windows) a window’s close box. You can test for this condition in a window's form method using this routine.

```
Case of
  : (Form event=On Load )
    ...
  : (Fnd_Wnd_CloseNow)
    CANCEL
  : (Form event=On Outside Call )
    ...
End case
```

You should call this routine from any non-modal windows that do not already call Foundation’s *Fnd_Gen_FormMethod* method. This function should be tested before the On Outside Call form event test, since it will always be delivered during this form event.

If you do call *Fnd_Gen_FormMethod*, *Fnd_IO_InputFormMethod*, or *Fnd_IO_OutputFormMethod*, you do not need to call this routine.

Fnd_Wnd_Info

Fnd_Wnd_Info (info requested) ➔ Text

Parameter	Type	Description
info requested	Text	Information desired
Function Result	Text	Response

Returns the requested information about the component.

```
$version_t:=Fnd_Wnd_Info ("version")
```

The *Fnd_Wnd_Info* method will respond to these requests:

Request	Response	Example
name	The component's full name	Foundation Windows
version	The component's version number	4.0.5 beta 2

This routine can also be called using the *Fnd_Gen_ComponentInfo* method without first testing to see if the component is installed:

```
$version_t:=Fnd_Gen_ComponentInfo ("Fnd_Wnd";"version")
```

See the *Fnd_Gen_ComponentInfo* method for more information.

Fnd_Wnd_MoveOnScreen

Fnd_Wnd_MoveOnScreen (->left; ->top; ->right; ->bottom; {window type})

Parameter	Type	Description
left	Pointer	Pointer to the left side of the window
top	Pointer	Pointer to the top of the window
right	Pointer	Pointer to the right side of the window
bottom	Pointer	Pointer to the bottom of the window
window type	Longint	Window type (optional)

Pass this routine pointers to a set of window coordinates before opening a new window. This routine modifies the window coordinates so the window will be opened completely on screen.

```
...
Fnd_Wnd_MoveOnScreen (->Left_i;->Top_i;->Right_i;->Bottom_i)
$winRef_i:=Open window(Left_i;Top_i;Right_i;Bottom_i)
```

For example, if you were to recall a saved window position before opening a window, this routine would assure that the window is completely visible, even if the user had changed monitor sizes or resolutions since saving the original window coordinates.

If the **window type** is specified, this routine can make more precise adjustments to the new window position, taking into account the window's title bar and border sizes.

You do not need to call this routine before calling any of the Windows component routines. They all call this method internally.

Fnd_Wnd_OpenFormWindow

Fnd_Wnd_OpenFormWindow (->table; form name) ➜ Number

Parameter	Type	Description
table	Pointer	Pointer to the form's table
form name	Text	Name of the form
Function result	Number	The new window's reference number

Pass this method a pointer to a table and a form name from that table, and it will open a new window using the specified form's width and height. You can use other Windows component routines to set the window's type, title, and close box before calling this routine. Unless the new window uses a previously saved position (see the *Fnd_Wnd_UseSavedPosition* method), the new window will use the position set by the *Fnd_Wnd_Position* method.

```
Fnd_Wnd_Title ("New Invoice for "+[Customers]Name)
Fnd_Wnd_Type (Movable form dialog box )
Fnd_Wnd_Position (Fnd_Wnd_CenterOnWindow )
Fnd_Wnd_OpenFormWindow (->[Invoices];"Invoice")
DIALOG([Invoices];"Invoice")
CLOSE WINDOW
```

If you do not need the new window's reference number you can safely call this method as a command, rather than as a function.

Fnd_Wnd_OpenWindow

Fnd_Wnd_OpenWindow (width; height) ➜ Number

Parameter	Type	Description
width	Longint	The width of the new window
height	Longint	The height of the new window
Function result	Number	The new window's reference number

This routine will open a new window with the specified width and height. You can use other Window component routines to set the window's type, title, and close box before calling this routine. Unless the new window uses a previously saved position (see the *Fnd_Wnd_UseSavedPosition* method), the new window will use the position set by the *Fnd_Wnd_Position* method).

If you do not need the new window's reference number you can safely call this method as a command, rather than as a function.

This method is basically Foundation's version of the typical *CenterWindow* method. If no other Foundation Windows routines are used, a plain window will be opened in the center of the screen.

Fnd_Wnd_Position

Fnd_Wnd_Position ({position setting}) ➜ Number

Parameter	Type	Description
position setting	Longint	Window position (optional)
Function result	Number	The position setting for the next window

Allows the developer to set the window position for the upcoming dialog or window. The following positions are available using these constants:

Value	Foundation Constant	Description
0	Default	Centered on screen
1	Fnd_Wnd_CenterOnScreen	Centered on screen
2	Fnd_Wnd_CenterOnWindow	Centered over front most window
3	Fnd_Wnd_MacOSXSheet	Sheet if possible, or centered over front most window
4	Fnd_Wnd_Stacked	Stacked with other stacked windows
5	Fnd_Wnd_StackOnWindow	Stacked over the front most window

The Fnd_Wnd_MacOSXSheet position will use the sheet type window when using 4D 2004 or later on Mac OS X. It will also be used for alert and confirm dialogs displayed by the Dialog component, if the ToolboxPack plugin is installed. Otherwise, the window will be centered over the front most window.

This command can be called before using many Foundation routines that take advantage of the Windows component. For example:

```
Fnd_Wnd_Position (Fnd_Wnd_CenterOnWindow )
Fnd_Dlg_Alert ("A value in the Company field is required.")
```

Fnd_Wnd_SavePosition

Fnd_Wnd_SavePosition ({name{; window ref}})

Parameter	Type	Description
name	Text	The name of the preference item (optional)
window ref	Longint	The window to save (optional)

Saves the front most window of the current process in the user's preferences file (if the Foundation Preferences component is available). See the *Fnd_Wnd_UseSavedPosition* command for an example.

This is basically the same as calling *Fnd_Pref_SetWindow*, but you do not have to check first to see if the Preferences component is available.

Fnd_Wnd_SendCloseRequests

Fnd_Wnd_SendCloseRequests ({process number})

Parameter	Type	Description
process number	Longint	Process number (optional)

Adds the current process to the list of processes with windows that should be closed if the user Option-closes a window or selects **Close All Windows** from a menu.

```
Fnd_Wnd_SendCloseRequests  ` Send window close requests to this process.
```

The window displayed by the process should then call the *Fnd_Wnd_CloseNow* function whenever an outside call is received.

If no process number is passed, the current process is used.

Both the process name and number are stored, to make sure we are trying to close the same process that was submitted here. So you do not need to do anything to remove a process from the internal list stored by Foundation.

Fnd_Wnd_SetCloseBox

Fnd_Wnd_SetCloseBox ({add close box?})

Parameter	Type	Description
add close box?	Boolean	Add a close box? (optional)

This routine is obsolete. Use the *Fnd_Wnd_CloseBox* method instead.

Fnd_Wnd_SetPosition

Fnd_Wnd_SetPosition ({position setting})

Parameter	Type	Description
position setting	Longint	Window position (optional)

This routine is obsolete. Use the *Fnd_Wnd_Position* method instead.

Fnd_Wnd_SetTitle

Fnd_Wnd_SetTitle ({title})

Parameter	Type	Description
title	Text	Window title (optional)

This routine is obsolete. Use the *Fnd_Wnd_Title* method instead.

Fnd_Wnd_SetType

Fnd_Wnd_SetType ({window type})

Parameter	Type	Description
window type	Longint	Window type (optional)

This routine is obsolete. Use the *Fnd_Wnd_Type* method instead.

Fnd_Wnd_Title

Fnd_Wnd_Title ({title}) ➜ Text

Parameter	Type	Description
title	Text	Window title (optional)
Function result	Text	The title for the next window

Gets and sets the window title for the upcoming window. Call this routine before calling *Fnd_Wnd_OpenWindow* or *Fnd_Wnd_OpenFormWindow*.

```
Fnd_Wnd_Title ("New Invoice for "+[Customers]Name)
Fnd_Wnd_OpenFormWindow (->[Invoices];"Invoice")
DIALOG([Invoices];"Invoice")
CLOSE WINDOW
```

This method can also be called before many of the Foundation routines that display windows to override the default window title:

```
Fnd_Wnd_Title ("Find and Print")
Fnd_Find_Display
```

Fnd_Wnd_Type

Fnd_Wnd_Type ({window type}) ➜ Number

Parameter	Type	Description
window type	Longint	Window type (optional)
Function result	Text	The window type for the next window

Call this routine before calling *Fnd_Wnd_OpenWindow* or *Fnd_Wnd_OpenFormWindow* to specify the window type to use. You can pass any standard 4D window type constant.

```
Fnd_Wnd_Type (Modal dialog box )
Fnd_Wnd_OpenFormWindow (500;300)
```

This method can also be called before many of the Foundation routines that display windows to override the default window type:

```
Fnd_Wnd_Type (Plain fixed size window )
Fnd_Find_Display
```

Fnd_Wnd_UseSavedPosition

Fnd_Wnd_UseSavedPosition ({pref name})

Parameter	Type	Description
pref name	Text	Name of the saved position (optional)

This method tells the window routines we want to try to use a saved position. Pass a window name you have previously added to the user's preferences by calling the *Fnd_Wnd_SavePosition* or *Fnd_Pref_SetWindow* methods. If the preference is available, the saved position will be used rather than the position specified by a call to the *Fnd_Wnd_Position* method.

Your method will look something like this:

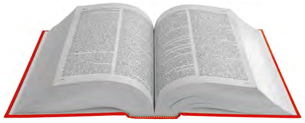
```
Fnd_Wnd_UseSavedPosition ("My Saved Window Position")
Fnd_Wnd_Position (Fnd_Wnd_CenterOnScreen )  ` In case no preference exists.
Fnd_Wnd_OpenFormWindow (->[Invoices];"Invoice")
DIALOG([Invoices];"Invoice")
Fnd_Wnd_SavePosition ("My Saved Window Position")  ` Save the position.
CLOSE WINDOW
```

If no preference name is specified, the current process name is used as the preference name to retrieve the saved window position.

Additional Credits

Thanks to Jack Delay of Island Reflections for updating the *Fnd_Wnd_MoveOnScreen* method so that it can properly handle multiple monitors.

Constants



Foundation has created a few internal constants that are used by different methods.

Fnd_Gen_CurrentFormType

Here are the values that this routine might return:

Value	Constant	Description
0	Fnd_Gen_NoForm	No window
1	Fnd_Gen_UnknownForm	Unknown form type
2	Fnd_Gen_OutputForm	An output form
3	Fnd_Gen_InputForm	An input form
4	Fnd_Gen_PreferencesForm	The Preferences window
5	Fnd_Gen_ListEditorForm	The List Editor window
6	Fnd_Gen_AboutForm	The About box
7	Fnd_Gen_SqNoEditorForm	The Sequence Number Editor window

Fnd_Dlg_SetIcon

This routine uses any of the following constants as its parameter:

Value	Constant	Description
0		Default icon (Note)
1	Fnd_Dlg_NoteIcon	Note Icon
2	Fnd_Dlg_WarnIcon	Warn icon
3	Fnd_Dlg_StopIcon	Stop Icon

Fnd_Wnd_Position

Pass one of the following values or constants to the position setting parameter:

Value	Constant	Description
1	Fnd_Wnd_CenterOnScreen	Centered on screen
2	Fnd_Wnd_CenterOnWindow	Centered over frontmost window
3	Fnd_Wnd_MacOSXSheet	Sheet if possible, or centered over frontmost window
4	Fnd_Wnd_Stacked	Stacked with other stacked windows
5	Fnd_Wnd_StackedOnWindow	Stacked over the frontmost window

Some additional Foundation constants you may find useful:

Value	Constant	Description
1	Fnd_Dlg_CenterOnScreen	Centered on screen
2	Fnd_Dlg_CenterOnWindow	Centered over frontmost window
3	Fnd_Dlg_MacOSXSheet	Sheet if possible, or centered over frontmost window
131072	Fnd_Gen_DefaultStackSize	Default Stack Size for new process