Tecplot®

Add-On Developer's Kit User's Manual

Version 10, Release 3

Tecplot, Inc.
Bellevue, Washington
February, 2004

Table Of Contents

CHAPTER 1 About Add-ons

Tecplot add-ons are executable modules that extend Tecplot's basic functionality in a well-defined, systematic way. Add-ons are implemented as compiled function libraries, called variously shared objects, shared libraries, or dynamic-link libraries (DLLs). Using the Tecplot Application Programming Interface described in this manual and its companion, the *Tecplot Add-on Developer's Kit Online Reference,* you can create add-ons to generate plots, manipulate or analyze data, or perform a broad variety of specialized tasks involving Tecplot. Because the add-ons are shared runtime objects, however, you do not need to link them into Tecplot. This means that you are not limited to using the compilers Amtec uses, and you do not have to compile (or recompile) large libraries of Tecplot function calls.

Different operating systems have different ways of creating and using shared objects. The Tecplot Add-on Developer's Kit provides utilities that mask most of these differences for related platforms (that is, all UNIX systems will behave approximately the same and all Windows systems will behave approximately the same—the ADK tools will resolve the differences).

ADK documentation is occasionally updated. The latest release may be downloaded from **www.tecplot.com/support/tecplot_documentation.htm**.

CHAPTER 2 *Creating Add-ons under UNIX*

2.1. Setting Up to Build Add-ons

To create add-ons in Tecplot you must set up a working directory where source code can be created and edited. This directory will hereafter be called the Add-on Development Root Directory. You may create any number of add-ons in the Add-on Development Root Directory.

To set up for building add-ons do the following:

1. Install Tecplot if you have not done so already. Make sure the Add-on Development Tools option was selected during the installation process.
2. Create the Add-on Development Root Directory if you have not done so already. This can be anywhere you choose.
3. Be sure that you have the **TEC100HOME** environment variable defined and assigned to the directory where Tecplot was installed.
4. Be sure your **PATH** environment variable includes the following:

   ```
   $TEC100HOME/bin:$TEC100HOME/adk/bin
   ```

5. Create a new file called **tecdev.add** in the directory created in step 2 (i.e. your Add-on Development Root Directory). Edit the file and add the following line:

   ```
   #!MC 100
   ```

6. (Optional) If you plan on using the Tecplot GUI builder, then add the following line to the **tecdev.add** file in your Add-on Development Root Directory:

   ```
   $!LoadAddon "|TECHOME|/lib/libguibld"
   ```

7. Set the environment variable **TECADDONDEVDIR** to the path of the directory created in step 2.
8. Set the environment variable **TECADDONDEVPLATFORM** to one of the valid platform names. A list of valid platforms can be obtained by running **tecplot -platlist**

From this point on, when you want to test the add-ons you are developing, use the **-develop** flag when running Tecplot. Later when you want to make your add-on accessible to all who run Tecplot, just

copy the shared object library to the **lib** subdirectory below the Tecplot Home Directory and include the command:

```
$!LoadAddOn "|TECHOME|/lib/libMyAddOnName"
```

in the **tecplot.add** file in the Tecplot Home Directory.

2.2. Creating a New Add-on

1. Go to the Add-on Development Root Directory (i.e., the directory created in step 2 of Section 2.1, "Setting Up to Build Add-ons.").
2. Type:

   ```
   CreateNewAddOn
   ```

   This will ask you a few questions about the add-on to be built, including whether or not you intend to use the Tecplot GUI Builder. When this is finished, you will have a new sub-directory named MyAddOnName, where MyAddOnName is the name that you supplied in step 2 while running CreateNewAddOn. This subdirectory contains a set of file. These files can be compiled to create a minimal add-on.
3. Edit the **tecdev.add** file located in the Add-on Development Root directory and add the following line:

   ```
   $!LoadAddOn "|$TECADDONDIR|/libMyAddOnName"
   ```

   where *MyAddOnName* is the name you supplied in step 2 while running **CreateNewAddOn**.

For your add-on to communicate with Tecplot it must do the following:

4. Make public an "initialization" function named **InitTecAddOn**. When you run **CreateNewAddOn** this function is created automatically for you and is located in the file **main.c** (or **main.cpp**). When Tecplot starts up it scans the **tecdev.add** file, loads named shared object libraries and makes a call to the **InitTecAddOn** function.

   The initialization function typically includes a call to add a converter, add a loader, register a curve fit, or add an item to the Tools menu, so the add-on can be accessed from the Tecplot interface.
5. Make calls to the TecUtil functions available from the **libtec** shared object library. These functions allow you to do a wider range of tasks than can be done through the Tecplot interface itself.
6. If your add-on does not require a custom built GUI, you will, at this point, have a source file named **main.c**, and perhaps a source file named **engine.c**. The latter file contains callback functions for data loaders, data convertors, or curve fits.

2.3. Creating the Graphical User Interface for Your Add-on

The Tecplot Add-on Developers Kit includes a simple GUI builder called Tecplot GUI Builder (TGB). You are not restricted to this GUI builder. You may use a commercial GUI builder like *Builder Xcessory* or *X-Designer*. Chapter 8 of this document outlines how to use the Tecplot GUI Builder. It is provided on the Tecplot CD. When you run **`CreateNewAddOn`** and choose to use the TGB, a starter set of TGB files is created for you.

2.4. Compiling the Add-on

2.4.1. Using Runmake

If you used **`CreateNewAddOn`**, compiling the add-on is straightforward. Go to the subdirectory where your add-on source code is located and type:

```
Runmake
```

You will be prompted for the platform type and what type of executable to create.

If you know the platform name and the build option ahead of time then you can run **`Runmake`** without the questions. For example, to compile on an SGI machine under IRIX 6.5 and create a debug version use:

```
Runmake sgix.65 -debug
```

To make a release version use:

```
Runmake sgix.65 -release
```

If all goes well with the compile, you will end up with a shared object library located in **`../lib/`***platform/buildtype*. Running Tecplot with the **`-develop`** flag automatically directs it to look for your library in this directory.

Note: If the Tecplot Home Directory and your Add-on Development Directory are located in directories that can be remotely mounted by other UNIX computers, then you can log on to those computers and use **`Runmake`** as described earlier. The resulting shared library will be stored in the appropriate subdirectory for the computer platform.

2.4.2. Editing the CustomMake File

The **Runmake** command used to build your add-on actually invokes the UNIX **make** program with a large list of flags that customize the make process for your platform. Just prior to calling **make**, the **Runmake** shell script checks to see if a local file called **CustomMake** exists and is executable. If so, it runs the **CustomMake** shell script in place and then runs **make**. This process allows you to add to or completely replace any assignments made by **Runmake**.

For example, suppose you want to add an additional flag called **-xg** to the **cc** compile command. You could do so by editing the local **CustomMake** shell script in the sub-directory of your add-on and adding:

```
CFLAGS="$CFLAGS -xg"
```

This replaces **CFLAGS** (i.e. the flags used with the **cc** command) with its old contents plus the **-xg** flag.

The default **CustomMake** file created in your add-on directory when you run **CreateNewAddOn** contains edit instructions including an explanation of the flags available for you to change.

CHAPTER 3 *Creating Add-ons under Windows*

3.1. Licensing of Microsoft-Supplied Dynamic-Link Libraries

The Tecplot ADK is supplied with dynamic-link libraries created by Microsoft. This is in compliance with the Visual Studio license agreement. The license agreement, however, also states that licenses cannot be transferred a second time unless the party distributing the libraries also has a Visual Studio license agreement. In other words, if you develop a Tecplot add-on and you plan on distributing it outside of your organization, then you must also have the right to distribute the Microsoft dynamic link libraries yourself (if you own Microsoft Visual Studio then you have this right).

3.2. Setting Up to Build Add-ons

The Tecplot Add-On Developers Kit contains the necessary include files (**`.h`**) as well as a **`tecplot.lib`** file with which to link add-on source code. Tecplot makes its functions available by exporting them from "tecplot.exe".

To setup your system for building add-ons, install Tecplot Version 10 if you have not done so already. Make sure the Add-on Developers Kit option was selected during the installation process. The **`SETUP`** program will automatically set your **`TEC100HOME`** environment variable and include the **`bin`** sub-directory, below the Tecplot Home directory, in your path.

3.3. Creating an Add-on with Visual C++

This section assumes that you are using Visual C++ 5.0 or later and that you are familiar with its use and concepts such as DLLs and callback functions.

Creating an add-on for Tecplot requires creating a DLL. If you are not familiar with this process, please refer to your Visual C++ documentation and online help. It would be a good idea to go through several examples of creating DLLs before attempting to create an add-on.

3.3.1. Using the Tecplot Visual C++ Add-on Wizard

If you are using Developer Studio 5 or 6 to build your add-on, you can use the Tecplot 10 Add-on Wizard to create a starter set of C++ source files.

To integrate the Tecplot Add-on Wizard with Developer Studio:

1. Be sure you are using Developer Studio Version 5 or 6.
2. Run Developer Studio and select "New..." from the File menu. Click on the "Projects" tab and you should see "Tecplot 10 Addon Wizard" as one of the project types. If not, copy the files **`TGBAddOn.awx`** and **`TGBAddOn.hlp`** from the **`|TEC100HOME|\bin\ide`** directory into the **`%MSDEVDIR%\Bin\IDE`** directory and run Developer Studio again. If you are unsure where to copy these files, search for where awx files are located on your computer (i.e. search for **`TGBAddOn.awx`**) and copy **`TGBAddOn.awx`** and **`TGBAddOn.hlp`** to this location.
3. Select "OK" and follow the prompts.
4. From the Developer Studio "Build" menu, select "Build MyProject".

The Tecplot add-on wizard also gives instructions for running Tecplot directly from your DLL project:

1. Select Project/Settings.
2. Click on the Debug tab.
3. Select the “General” category.
4. Set the “Executable for debug session” file to be **`tecplot.exe`** (include the full path).
5. Set the working directory to be “Debug.”
6. Set the program arguments to be -loadaddon *project_name* where *project_name* is the base name of your DLL (that is, without the .dll extension).

You can now set a breakpoint anywhere in your code to debug your add-on.

3.3.2. Creating an Add-on by Hand Using Visual Studio

The steps described below are done automatically by the Tecplot add-on wizard. We recommend using the Tecplot add-on wizard instead of the procedure below.

1. To create an add-on for Tecplot using Visual C++, your project workspace must be of type “MFC AppWizard (dll)” or “Dynamic-Link Library”. (In other words, your project must create a DLL.) You can select this when starting a new project workspace. Note: You must select “Regular DLL using shared MFC DLL” when prompted for the type of DLL.

2. In order to have access to Tecplot's functions and data, you will need to make sure that your project can find the Tecplot include files. These files are located in the include subdirectory of the Tecplot Home Directory. You can add this directory to your project in the Project Settings dialog on the C/C++ tab in the Preprocessor page with the "Additional include directories" field. These files declare Tecplot's functions and data types so that they will be available to your add-on. The functions (with names starting with TecUtil) allow you to do any tasks Tecplot can do, and other tasks that meet your specialized needs.
3. You will also need to make the **tecplot.lib** file available to your project to resolve the Tecplot functions at link time. There are two ways to do this. One is to actually insert the **tecplot.lib** file into your project with the "Project/Add to Project/Files" menu. The other is to add the **tecplot.lib** file to the "Object/library modules" field on the Link tab of the Project Settings dialog. The **tecplot.lib** file is located in the **bin** subdirectory of the Tecplot Home Directory.
4. In order for your add-on to communicate with Tecplot, it must export a **STDCALL** initialization function called InitTecAddOn. In C/C++ the function should look like:

```
EXPORTFROMADDON void STDCALL InitTecAddOn (void)
{
  .
  .
  .
}
```

5. **Note:** If you are using MFC, the following line must be added to your initialization function and any other functions which Tecplot will call:

```
MANAGESTATE
```

6. When you are ready to test your add-on with Tecplot, you need to create your DLL file, which you do by building your project. Make sure you know where your DLL file is located.
7. The easiest way to test your add-on with Tecplot is to run Tecplot from your DLL project. To do this:

   1. Select Project/Settings.
   2. Click on the Debug tab.
   3. Select the "General" category.
   4. Set the "Executable for debug session" file to be **tecplot.exe** (include the full path if necessary).
   5. Set the working directory to be "Debug."

6. Set the program arguments to be -loadaddon *project_name* where *project_name* is the base name of your DLL (that is, without the .dll extension).

You can now set a breakpoint anywhere in your code to debug your add-on.

3.3.3. What If My Add-on Is Not Working?

There are several things to look at if your add-on is not working. The first clue should come from Tecplot, which attempts to let you know where the problem with your add-on occurred. (For example, it couldn't load it or couldn't find its initialization function).

Tecplot expects that all of the functions that are passed to it from an add-on are **`STDCALL`**. Please note that this is not the default calling convention in Visual C++. **`STDCALL`** is automatically included if you use the **`EXPORTFROMADDON`** keyword.

You may want to check your DLL to see what functions are actually being exported. You can do this with the **`DUMPBIN`** utility with the **`/EXPORTS`** flag. (For example, "**`DUMPBIN/EXPORTS myaddon.dll`**".) If **`InitTecAddOn`** is not listed, Tecplot will not be able to access it.

If you are using an MFC DLL make sure you are using MFC in a shared library. Also check your preprocessor definitions: **`_AFXDLL`** must be defined and **`_USRDLL`** must not be defined.

3.3.4. Getting Started—A Simple Example of an MFC DLL

Following is an example of how an MFC DLL can be created (note that using the Tecplot add-on Wizard will accomplish many of the following steps automatically - see section 3.3.1, "Using the Tecplot Visual C++ Add-on Wizard."):

1. To begin creating an add-on for Tecplot using Visual C++, start with a new project workspace. Select a project type of "MFC AppWizard (dll)". Name the project **SimpMFC**. When prompted, select "Regular DLL using shared MFC DLL." (This example will assume that your project is located in **`c:\projects\simpmfc`** and that the Tecplot Home Directory is **`c:\tec100`**. Please substitute the names of your own project directories in the example below.)
2. To add a dialog to your add-on, choose the Insert menu and then the Resource menu. Select "Dialog" and press OK. Add a "Static Text" to your dialog and change its Caption to read "**`This is an MFC add-on`**." Double click on the dialog and change the dialog ID from "**`IDD_DIALOG1`**" to "**`IDD_ADDONDLG`**" and the Caption to "**`Simple MFC Add-On`**."

3. In order to use the dialog in the add-on, you need to create a class for it. Bring up the Class Wizard, and choose to create a new class. Type in a Name of "**CSimpDlg**." Select a Base Class of "**CDialog**" and the Dialog ID "**IDD_ADDONDLG.**" Press Create to create the class, and then you can close the Class Wizard.
4. Edit the file **SimpMFC.cpp**. Near the top of the file, just after the line "**#include SimpMFC.h**", add the following lines:

```
#include "TECADDON.h"
#include "SimpDlg.h"
```

At the bottom of the file, after the line "**CSimpMFCApp theApp**", add the following lines:

```
static void STDCALL LaunchSimpleDialog(void)
{
 AFX_MANAGE_STATE(AfxGetStaticModuleState( ));
  CSimpDlg modal;
  modal.DoModal();
}

EXPORTFROMADDON void STDCALL InitTecAddOn (void)
{
  AFX_MANAGE_STATE(AfxGetStaticModuleState( ));

  TecUtilLockOn();
  AddOnId=TecUtilAddOnRegisterInfo(100,"Simple MFC Test",
                    "1.0",
                    "My Company");
  TecUtilMenuAddOption("Tools",
                       "Simple MFC Addon",
                       'S',
                       LaunchSimpleDialog);
  TecUtilLockOff();

}
```

5. In the Project Settings dialog, make the following changes:
   - Select the Debug tab and change to the General page.
   - Set the "Executable for debug sessions" field to be:

     "c:\tec100\bin\tecplot.exe".

   - Set the "Working directory" to be "Debug".
   - Set the "Program arguments" to be:

     "-loadaddon simpmfc"

- Select the C/C++ tab and change to the Preprocessor page. Add to the "Additional include directories" field:

  "c:\tec100\include"

- Select the Link tab and change to the General page. In the Object/library modules field, add:

  "c:\tec100\bin\tecplot.lib"

6. Build the debug version of your project and run it. When Tecplot is launched, go to the Tools menu and select "Simple MFC Addon". Your dialog will be launched.

3.3.5. Getting Started - A Simple Example of a Non-MFC DLL

Following is an example of how a non-MFC DLL can be created:

1. To begin creating an add-on for Tecplot using Visual C++, start with a new project workspace. Select a project type of "Win32 Dynamic-Link Library." Name the project "**Simple.**" This example will assume that your project is located in **c:\projects\simple** and that the Tecplot Home Directory is **c:\tec100**. Substitute the names of your own project directories in the example below.
2. Start up a new text file, and type in the following lines. Then, save the file as "**SIMPLE.C.**"

```
#include "TECADDON.h"

static void STDCALL LaunchSimpleDialog(void)
{
  TecUtilLockStart(AddOnID);
  TecUtilDialogMessageBox("This is the Simple dialog!",
                          MessageBox_Information);
  TecUtilLockFinish(AddOnID);
}

EXPORTFROMADDON void STDCALL InitTecAddOn(void)
{

  TecUtilLockOn();
  AddOnID=TecUtilAddOnRegisterInfo(100,"Simple non-MFC Test",
                           "1.0",
                           "My Company");
  TecUtilMenuAddOption("Tools",
                       "Simple non-MFC Test",
```

```
                        'S',
                        LaunchSimpleDialog);

    TecUtilLockOff();
}
```

3. In the Project Settings dialog, make the following changes: Select the Debug tab and change to the General page. Set the “Executable for debug sessions” field to be **`c:\tec100\bin\tecplot.exe`**. Set the “Working directory” to be "Debug". Set the “Program arguments” to be **`-loadaddon simple`**.

   Select the C/C++ tab and change to the Preprocessor page. Add **`c:\tec100\include`** to the “Additional include directories” field.

   Select the Link tab and change to the General page. In the Object/library modules field, add **`c:\tec100\bin\tecplot.lib`** to the list of files.

4. Add the **`SIMPLE.C`** file to your project.

   Build the debug version of your project and then run it. When Tecplot comes up, go to the Tools menu and select “**`Simple non-MFC Test`**.” Your dialog is launched with the message, “This is the Simple dialog!”

3.4. Creating an Add-on with Digital (Compaq) Visual Fortran

This section assumes that you are using Digital (now Compaq) Visual Fortran 5.0 or later and that you are familiar with its use and concepts such as DLLs.

Creating an add-on for Tecplot requires creating a DLL. If you are not familiar with this process, please refer to your Visual Fortran documentation and online help. It would be a good idea to go through several examples of creating DLLs before attempting to create an add-on.

3.4.1. Using the Tecplot GUI Builder Add-on Wizard

If you are using Developer Studio 5 or later to build your add-on, you can use the Tecplot GUI Builder add-on wizard to create a starter set of Fortran source files. Note: This wizard gives you the option of using the Tecplot GUI Builder to construct a user interface for your add-on. If you intend to build the interface some other way, you may still use this wizard—simply do not select the "Launch a dialog from the menu" option (for the General Purpose add-on wizard option).

To integrate the Tecplot add-on wizard with Developer Studio:

1. Be sure you are using Developer Studio Version 5 or 6.

2. Run Developer Studio and select "New..." from the File menu. Click on the "Projects" tab and you should see "Tecplot 10 Addon Wizard" as one of the project types. If not, copy the files **`TGBAddOn.awx`** and **`TGBAddOn.hlp`** from the **`|TEC100HOME|\bin\ide`** directory into the **`%MSDEVDIR%\Bin\IDE`** directory and run Developer Studio again. If you are unsure where to copy these files, search for where awx files are located on your computer (i.e. search for **`TGBAddOn.awx`**) and copy **`TGBAddOn.awx`** and **`TGBAddOn.hlp`** to this location.
3. Select "OK" and follow the prompts. Ensure you select the Fortran language.
4. From the Developer Studio "Build" menu, select "Build MyProject".

The Tecplot GUI Builder add-on wizard also gives instructions for running Tecplot directly from your DLL project:

1. Select Project/Settings.
2. Click on the Debug tab.
3. Select the “General” category.
4. Set the “Executable for debug session” file to be **`tecplot.exe`** (include the full path if necessary).
5. Set the working directory to be “Debug.”
6. Set the program arguments to be -loadaddon *project_name* where *project_name* is the base name of your DLL (that is, without the .dll extension).

You can now set a breakpoint anywhere in your code to debug your add-on.

3.4.2. Creating an Add-on by Hand Using Visual Studio

The steps described below are done automatically by the Tecplot GUI Builder add-on wizard. If you intend to use the Tecplot GUI Builder to create your add-on’s user interface, it is recommended that you use the wizard instead of the procedure below.

1. To create an add-on for Tecplot using Visual Fortran, your project workspace must be of type “Dynamic-Link Library”. (In other words, your project must create a DLL.) You can select this when starting a new project workspace.
2. In order to have access to Tecplot's functions and data, you will need to make sure that your project can find the Tecplot include files. These files are located in the include subdirectory of the Tecplot Home Directory. You can add this directory to your project in the Project Settings dialog on the Fortran tab in the Preprocessor page with the “Custom INCLUDE and USE Paths” field. These files declare Tecplot's functions and data types so that they will be available to your add-on. The functions (with names starting with TECUTIL) allow you to do any tasks Tecplot can do, and other tasks that meet your specialized needs.

3. You will also need to make the **tecplot.lib** and **fglue.lib** files available to your project to resolve the Tecplot functions at link time. If you will be using the Tecplot GUI Builder to create your add-on's user interface, file **WinGUI.lib** is also required. There are two ways to do this. One is to actually insert the files into your project with the "Project/Add to Project/Files" menu. The other is to add the files to the "Object/library modules" field on the Link tab of the Project Settings dialog. The files are located in the **bin** subdirectory of the Tecplot Home Directory.
4. In order for your add-on to communicate with Tecplot, it must export an initialization subroutine called InitTecAddOn (case is not important). This subroutine should look like:

```
subroutine InitTecAddOn ()
!DEC$attributes DLLEXPORT::InitTecAddOn
 .
 .
 .
end
```

5. When you are ready to try out your add-on with Tecplot, you need to create your DLL file, which you do by building your project. Make sure you know where your DLL file is located. When you build the Debug version of your project, you may get a warning message recommending you use the "/NODEFAULTLIB" link option. You may generally ignore this warning, but if you wish to eliminate it, you may check the "Ignore all default libraries" link option, and explicitly add the additional Fortran and C libraries required (refer to your Fortran documentation for a list of these).
6. The easiest way to run your add-on with Tecplot is to run Tecplot from your DLL project. To do this:

   1. Select Project/Settings.
   2. Click on the Debug tab.
   3. Select the "General" category.
   4. Set the "Executable for debug session" file to be **tecplot.exe** (include the full path if necessary).
   5. Set the working directory to be "Debug."
   6. Set the program arguments to be -loadaddon *project_name* where *project_name* is the base name of your DLL (that is, without the .dll extension).

You can now set a breakpoint anywhere in your code to debug your add-on.

3.4.3. What If My Add-on Is Not Working?

There are several things to look at if your add-on is not working. The first clue should come from Tecplot, which attempts to let you know where the problem with your add-on occurred. (For example, it couldn't load it or couldn't find its initialization function).

Tecplot expects that all of the functions and subroutines that are passed to it from an add-on are **STDCALL**. This is the default calling convention in Visual Fortran version 5 (you should *not* explicitly set the STDCALL attribute for these subroutines).

You may want to check your DLL to see what functions are actually being exported. You can do this with the **DUMPBIN** function with the **/EXPORTS** flag. (For example, "**DUMPBIN/EXPORTS myaddon.dll**".) If **INITTECADDON** is not listed, Tecplot will not be able to access it. Note that the name of your initialization function will be decorated as follows: "**_INITTECADDON@0**". If it is not, make sure you are compiling using the default External Procedures options for Fortran. You can do this by displaying the Project Settings dialog, selecting the Fortran tab, then selecting the External Procedures option, and choosing the options marked with an asterisk. Also make sure you have set the **DLLEXPORT** attribute as shown in the **InitTecAddOn** example above.

3.4.4. Getting Started - A Simple Example of a DLL

Following is an example of how a DLL can be created:

1. To begin creating an add-on for Tecplot using Visual Fortran, start with a new project workspace. Select a project type of "Win32 Dynamic-Link Library." Name the project "**Simpfor**." This example will assume that your project is located in **c:\projects\simpfor** and that the Tecplot Home Directory is **c:\tec100**. Substitute the names of your own project directories in the example below.
2. Start up a new text file, and type in the following lines. Then, save the file as "**SIMPFOR.F.**"

```
      subroutine LaunchSimpforDialog()
      include "FGLUE.INC"
      integer i
      call TecUtilLockStart(AddOnID)
      i = TecUtilDialogMessageBox(
     &       "This is the Simpfor dialog!"//char(0),
     &       MessageBox_Information)
      call TecUtilLockFinish(AddOnID)
      return
```

```
end


subroutine InitTecAddOn()
!DEC$ attributes DLLEXPORT::InitTecAddOn
include "FGLUE.INC"
integer i
external LaunchSimpforDialog

call TecUtilLockOn
call TecUtilAddOnRegister(100,
&      "Simple Fortran Test"//char(0),
&      "1.0"//char(0),
&      "My Company"//char(0)AddOnID)
i = TecUtilMenuAddOption(
&      "Tools"//char(0),
&      "Simple Fortran Test"//char(0),
&      'S'//char(0),
&       LaunchSimpforDialog)

call TecUtilLockOff
return
end
```

3. In the Project Settings dialog, make the following changes: Select the Debug tab and change to the General page. Set the "Executable for debug sessions" field to be **`c:\tec100\bin\tecplot.exe.`** Set the "Working directory" to be "Debug". Set the "Program arguments" to be **`-loadaddon simpfor`**.

   Select the Fortran tab and change to the Preprocessor page. Add **`c:\tec100\include`** to the "Custom INCLUDE and USE Paths" field (separated from any other listed paths by a semi-colon).

   Select the Link tab and change to the General page. In the Object/library modules field, add **`c:\tec100\bin\tecplot.lib`** and **`c:\tec100\bin\fglue.lib`** to the list of files.
4. If you did not add the **`SIMPFOR.F`** file to your project when you created it, add it to your project now.
5. Build the debug version of your project and then run it. When Tecplot comes up, go to the Tools menu and select "**`Simple Fortran Test`**." Your dialog is launched with the message, "This is the Simpfor dialog!"

CHAPTER 4 *Porting Add-ons between Windows and UNIX*

Ideally, the process of transferring an add-on between operating systems begins when you write the first version of the add-on. The cross-platform strategy revolves around creating the original add-on with cross-platform-compatible ingredients. For many users, this means writing the add-on in C, since many UNIX machines have C compilers more readily available.

If your add-on has a graphical user interface, we recommend that you use the Tecplot GUI Builder (TGB), or some other code library which is portable between platforms. If your add-on uses MFC, then you must isolate the MFC code as much as possible and rewrite the user interface on the Motif side.

4.1. Porting Add-ons from Windows to UNIX

Here is the general procedure for porting add-ons from Windows to UNIX:

1. If your code is in C++ files, create a C file to compile on UNIX by doing the following:
   a. For each **`.cpp`** file, create a **`.c`** file with the same name.
   b. Move all of the code from the **`.cpp`** file into the **`.c`** file.
   c. Delete the code from the **`.cpp`** file and add the line: **`#include "`***xxx***`.c"`**.
   d. If this was an MFC project, add the line **`#include "stdafx.h"`** to the top of the file.
2. In UNIX, run the CreateNewAddOn script to start a new add-on. Be sure to select the TGB option if your add-on used the TGB option under Windows. See Chapter 2 for more information on how to use CreateNewAddOn.
3. Move all of the **`*.c`** and **`*.h`** files from your Windows project into the UNIX project directory.
4. Edit the UNIX Makefile to include the new **`*.c`** and **`*.h`** files.
5. Compile your add-on under UNIX.

4.2. Porting Add-ons from UNIX to Windows

Here is the general procedure for porting add-ons from UNIX to Windows:

1. Follow the directions in Chapter 3 for creating a non-MFC DLL in Visual Studio. It is important that you select "Win32 Dynamic Link Library" as the project type. Alternatively, you can use the new TGB Add-On Wizard to create this project for you (if your add-on used the TGB in UNIX; access the wizard via the New option on the File menu).
2. Move all of the **`*.c`**, and **`*.h`** files from your UNIX project directory to the new project directory in Windows.
3. In Visual Studio, select Project/Add to Project/Files... and add all of the **`*.c`** and **`*.h`** files. If you used the TGB in UNIX, you must tell Developer Studio not to compile the file **`guibld.c`**. Select Project/Settings..., choose settings for All Configurations, select the file **`guibld.c`**, set the toggle “Exclude file from Build,” and click OK.
4. If you did *not* use the TGB Add-On Wizard, be sure that you link with **`wingui.lib`**.
5. Select Project/Build to build your add-on.

CHAPTER 5 *Migrating Add-ons*

5.1. Migrating Add-ons from Version 9 to Version 10

5.1.1. Updating Fortran Add-ons

In most cases, you will only need to rename GUIF_ functions to TecGUI. There are a few exceptions to this rule, however. They are:

`GUIF_DeallocItemList` has been removed. You must replace this with *`TecUtilArrayDealloc`*.

Any TecGUI function which returns a string must pass the length after the result. This is different from the GUIF_ functions, which passed the length before the result. For example, **`GUIF_TextFieldGetString(TextField,MaxChars,Result)`** becomes **`TecGUITextFieldGetString(TextField,Result,MaxChars)`**. See the syntax for TecGUITextGetString and TecGUITextFieldGetString in the ADK Online Reference for complete information.

5.1.2. Binary Compatibility

Most add-ons written for Tecplot version 9 will run without modification with Tecplot version 10. However, in order to take advantage of all of the new features in version 10, some sections of add-on source code will have to be revised and the add-on will need to be recompiled.

Although most changes are optional, some changes are required for add-ons which manage Tecplot data. Add-ons which manage Tecplot data must be aware that it may encounter cell-centered or shared data. If the add-on is not aware of these new data features, it will not work properly when the user loads or creates this type of data in Tecplot. *Additonally, we strongly recommend that add-on developers not use raw data pointers.*

In particular, if an add-on obtains a variable reference for zone A and intends on writing to the reference and assumes that writing to that reference only effects zone A, then it must first call **`TecUtilDataValueBranchShared()`** before obtaining the reference.

For more complete information about the TecUtil functions mentioned below, see the ADK Online Reference.

5.1.3. Source Code Compatibility

Although no TecUtil functions have been removed from Version 10 ADK, some TecUtil functions and state changes have been superseded by new equivalent functions and state changes (see Section 5.7 for a list of deprecated functions). Therefore, while your add-on source code should compile without changes, use the new functions wherever possible to help ensure maximum future compatibility. Using the new functions and or state changes will, however, prevent your add-on from running with earlier versions of Tecplot.

If your add-on is written in FORTRAN, the new Version 10 ADK includes many new glue functions, enabling you to more fully utilize the new features of Version10. The new Windows Add-on Wizard can also generate FORTRAN code directly.

5.1.4. Access to Tecplot Data

NOTE: This section only applies if your add-on queries and/or manipulates Tecplot data that the user has loaded or created.

The most important consideration when converting add-ons from an earlier version of Tecplot is that in Tecplot 10 your add-on may encounter cell centered or shared data. If the addpon makes an assumption that the data is not cell-centered or shared, then it may not work properly when encountering such data. More importantly, in the case of shared data the user may not even be aware that there is a problem. For example, if an add-on obtains avariable reference for zone A and intends on writing to the reference and assumes that writing to that reference only effects zone A, then it must first call **`TecUtilDataValueBranchShared`** before obtaining the reference.

5.1.4.1. Data Sharing

If you are running an add-on built prior to V10 which attempts to modify shared data, a warning will be generated. To avoid this, you must use the new "share" family of functions when working with shared data.

5.1.5. Nodal and Cell-Centered Data

Add-ons written prior to Version 10 are assumed to have no knowledge of cell centered data. Thus, any add-on which does not register itself as a Version 10 add-on (using the locking/ registration functions described below) will generate an error if any of the following TecUtil functions are used when a cell centered variable is involved:

```
TecUtilDataValueGetRef
TecUtilDataValueGetByZoneVar
TecUtilDataValueSetByZoneVar
TecUtilDataValueGetRawPtr
```

If your add-on uses any of the above functions in conjunction with cell centered data, you will need to change your registration so that it uses the new `TecTuilAddOnRegister` function to identify itself as a Version 10 add-on. See section 5.5 for more information.

Note that an add-on can query a variable's value location (nodal or cell centered) with the following function:

```
ValueLocation_e TecUtilDataValueGetLocation(EntIndex_t Zone,
                    EntIndex_t Var)
```

5.1.6. Locking/Add-on Registration

Tecplot Version 10 contains the following new locking and add-on registration functions. Note that in order to use the new locking functions, you must also use the new registration functions. One reason to use the new locking functions is that they provide Tecplot with more information about which add-ons are loaded and what they are doing, which can help with add-on debugging problems.

```
Addon_pa TecUtilAddOnRegister(int TecplotAPIVersionNumber,
                    const char *name
                    const char *addon_version_string
                    const char *author
```

Returns a handle to use with TecUtilLockStart and TecUtilLockFinish. The first parameter is the TecUtilAPIVersion, 100 for version 10. This number informs Tecplot of the level of expertise of your Add-on. Add-ons registered with 100 will be assumed to know how to handle new features and thus are allowed to access some functions without errors or warning.

- To Lock Tecplot for add-on AddonID.

```
void TecUtilLockStart(AddOn_pa AddonID)
```

- Unlocks Tecplot

```
void TecUtilLockFinish(AddOn_pa Addon)
```

- Returns the name of the object currently locking Tecplot.

```
char *TecUtilLockGetCurrentOwnerName(void)
```

Note that there remains one instance where you must use the older **TecUtilLockOn** and **TecUtilLockOff** functions and that is for the start and end of **InitTecAddOn** itself. This is because prior to this point, **TecUtilAddOnRegister** has not been called and thus there is no valid **AddonID** handle.

5.1.7. State Changes

5.1.7.1. New State Changes in V10: The new state changes in V10 allow an add-on to know when auxiliary data has been added, changed or removed.

5.1.8. New State Change Registration Function

State Change Value	Example	Supplemental Information	When This Occurs
StateChange_AuxDataAdded	Auxiliary data has been added to the zone, dataset or frame.	An AuxDataLocation_e type which indicates what kind of auxiliary data was added. The zone number is also provided if the auxiliary data type is AuxDataLocation_Zone	See family of TecUtilAuxData functions
StateChange_AuxDataDeleted	Auxiliary data has been deleted from the zone, dataset or frame.	An AuxDataLocation_e type which indicates what kind of auxiliary data was deleted. The zone number is also provided if the auxiliary data type is AuxDataLocation_Zone	See family of TecUtilAuxData functions
StateChange_AuxDataAltered	Auxiliary data has been altered from the zone, dataset or frame.	An AuxDataLocation_e type which indicates what kind of auxiliary data was altered. The zone number is also provided if the auxiliary data type is AuxDataLocation_Zone	See family of TecUtilAuxData functions

Figure 5-1.

New add-ons can use the TecUtilStateChangeAddCallbackX function to register state changes (see Chapter 12 "Handling Tecplot State Changes From an Add-on"). The callback registered with this function is more flexible and can provide more information and is used in conjunction with the other new **`StateChange`** functions **`TecUtilStateChangeGetIndex`**, **`TecUtilStateChangeGetArbEnum`**, **`TecUtilStateChangeGetZone`**, **`TecUtilStateChangeGetZoneSet`** and **`TecUtilStateChangeGetStyleParam`**.

5.1.9. Deprecated TecUtil Functions

Note that the old functions are still available for backward compatibility.

Deprecated V9 Function	Equivalent V10 Function
TecUtilAddOnRegisterInfo	TecUtilAddOnRegister
TecUtilAnimateXYMapsX	TecUtilAnimatedLineMapsX

Figure 5-2.

Deprecated V9 Function	Equivalent V10 Function
TecUtilContourLabelAdd	TecUtilContourLabelX
TecUtilContourLabelDeleteAll	TecUtilContourLabelX
TecUtilContourLevelAdd	TecUtilContourLabelX
TecUtilContourLevelDeleteRange	TecUtilContourLabelX
TecUtilContourLevelDelNearest	TecUtilContourLabelX
TecUtilContourLevelNew	TecUtilContourLabelX
TecUtilContourLevelReset	TecUtilContourLabelX
TecUtilContourSetVariable	TecUtilContourLabelX
TecUtilCreateSimpleXYZone	TecUtilCreateSimpleZone
TecUtilFrameGetLinking	TecUtilLinkingGetValue
TecUtilFrameGetMode	TecUtilFrameGetPlotType
TecUtilFrameSetLinking	TecUtilLinkingSetValue
TecUtilFrameSetMode	TecUtilFrameGetPlotType
TecUtilGeomGetXYZAnchorPos	TecUtilGeomGetAnchorPos
TecUtilGeomSetXYZAnchorPos	TecUtil GeomSetAnchorPos
TecUtilLockOff	TecUtilLockFinish
TecUtilLockOn	TecUtilLockStart
TecUtilPickAddXYMaps	TecUtilPickAddLineMaps
TecUtilPickListGetXYMapIndex	TecUtilPickListGetLineMapIndex
TecUtilPickListGetXYMapNumber	TecUtilPickListGetLineMapNumber
TecUtilPolarToRectangular	TecUtilTransformCoordinates
TecUtilProbeXYGetDepValue	TecUtilProbeLinePlotGetDepValue
TecUtilProbeXYGetIndValue	TecUtilProbeLinePlotGetIndValue
TecUtilProbeXYGetSourceMap	TecUtilProbeLinePlotGetSourceMap
TecUtilStateChangeAddCallback	TecUtilStateChangeAddCallbackX
TecUtilStyleSetLowLevel	TecUtilStyleSetLowLevelX
TecUtilTextGetXYPos	TecUtilTextGetAnchorPos
TecUtilTextSetXYPos	TecUtilTextSetAnchorPos
TecUtilXYMapCopy	TecUtilLineMapCopy
TecUtilXYMapCreate	TecUtilLineMapCreate
TecUtilXYMapDelete	TecUtilLineMapDelete
TecUtilXYMapGetActive	TecUtilLineMapGetActive

Figure 5-2.

Deprecated V9 Function	Equivalent V10 Function
TecUtilXYMapGetAssignment	TecUtilLineMapGetAssignment
TecUtilXYMapGetCount	TecUtilLineMapGetCount
TecUtilXYMapGetName	TecUtilLineMapGetName
TecUtilXYMapIsActive	TecUtilLineMapIsActive
TecUtilXYMapGetAssignment	TecUtilLineMapSetActive
TecUtilXYMapGetCount	TecUtilLineMapGetCount
TecUtilXYMapGetName	TecUtilLineMapGetName
TecUtilXYMapIsActive	TecUtilLineMapIsActive
TecUtilXYMapSetActive	TecUtilLineMapSetActive
TecUtilXYMapSetAssignment	TecUtilLineMapSetAssignment
TecUtilXYMapSetBarChart	TecUtilLineMapSetBarChart
TecUtilXYMapSetCurve	TecUtilLineMapMapSetCurve
TecUtilXYMapSetErrorBar	TecUtilLineMapSetErrorBar
TecUtilXYMapSetName	TecUtilLineMapSetName
TecUtilXYMapSetSymbol	TecUtilLineMapSetSymbol
TecUtilXYMapSetSymbolShape	TecUtilLineMapSetSymbolShape
TecUtilXYMapShiftToBottom	TecUtilLineMapShiftToBottom
TecUtilXYMapShiftToTop	TecUtilLineMapShiftToTop
TecUtilXYMapStyleGetArbValue	TecUtilLineMapStyleGetArbValue
TecUtilXYMapStyleGetDouble Value	TecUtilLineMapStyleGetDoubleValue
TecUtilXYSetLayer	TecUtilLinePlotSetLayer
TecUtilZoneSetIJKMode	TecUtilZoneSetVolumeMode

Figure 5-2.

5.1.10. Deprecated Macro Subcommands in Version 10

TecUtilStyleSetLowLevel and TecUtilStyleSetLowLevelX operate using the equivalent macro commands and sub-commands required to set style attributesin Techplot. If you are calling TecUtilStyleSetLowLevelX with any older sub commands it will still work, however you can not take advantage of any of the new style settings in Tecplot.

5.1.10.1. Deprecated Version 9 Style Subcommands:

Old Constant	New Constant(s)
SV_VALUEBLANKINGCELLMODE	SV_VALUEBLANKCELLMODE
SV_CUTBELOW	SV_CONSTRAINT,SV_RELOP
SV_FRAMEMODE	SV_PLOTTYPE
SV_USERELATIVEPATHSINLAYOUTS	SV_USERRELATIVEPATHS
SV_FNAMEEXTENSION	SV_FNAMEFILTER
SV_TIMEOUT	NONE - NOW IGNORED BY TECPLOT
SV_MAXTRACELINES	NONE - NOW IGNORED BY TECPLOT
SV_THREEDVIEWCHANGEDRAWLEVEL	SV_USEAPPROXIMATEPLOTS, SV_PLOTAPPROXIMATIONMODE
SV_NONCURRENTFRAMEREDRAWLEVEL	SV_USEAPPROXIMATEPLOTS, SV_PLOTAPPROXIMATIONMODE
SV_FORCEGOURAUDFOR3DCONFLOOD	SV_GLOBALTHREED,SV_LIGHTSOURCE,SV_FORCEGOURAUDFOR3DCONTFLOOD
SV_MOUSEBUTTON2MODE	SV_MOUSEACTIONS
SV_MOUSEBUTTON3MODE	SV_MOUSEACTIONS
SV_MIDDLEMOUSEBUTTONMODE	SV_MOUSEACTIONS
SV_RIGHTMOUSEBUTTONMODE	SV_MOUSEACTIONS
SV_COLORMAPSHADERATIO	SV_PRINTSETUP,MS_NUMLIGHTSOURCESHADES
SV_AUTOREDRAW	SV_AUTOREDRAWISACTIVE
SV_SOFTWARE3DRENDERING	SV_USESSOFTWARERENDERING
SV_TIMEDREDRAWTIMEOUT	NONE - NOW IGNORED BY TECPLOT
SV_INITIALFRAMEMODE	SV_INITIALPLOTTYPE

Figure 5-3.

5.2. Migrating Add-ons From Tecplot 10 Release 1 to Release 3

Add-ons written using the Tecplot GUI Builder (TGB) should be updated if you plan on doing further development of these add-ons with Tecplot 10 Release 3. All other add-ons do not need any modifications.

For the most part the change involves renaming all functions starting with "**GUI_**" to "**TecGUI**", plus a couple of other minor changes.

Steps to Update

1. Make a backup copy of the source code for our add-on.
2. Go to the source code directory for your add-on.
3. Run: **`UpdateAddOn`**

   This script is available under UNIX only. Under Windows, you must edit all source code that contains calls to **`GUI_`** functions and replace "**`GUI_`**" with "**`TecGUI`**".

   If add-on is FORTRAN then rename "`GUIF_`" to be "`TecGUI`".
4. If your add-on uses the Tecplot GUI builder then bring up Tecplot with the new **GUI** builder and choose **Build**.
5. If you make use of **`adkutil.c`** functions then obtain the latest copies of **`adkutil.c`** and **`ADKUTIL.h`** from any sample add-on in the Tecplot distribution that uses this module.
6. (UNIX only) Edit **`Makefile`** and remove the reference to **`$(LIBGUI)`** from the link instruction.
7. (Windows only) In Developer Studio:
   - Select Project/Settings
   - Click in the **Link** tab.
   - Remove **`$(TEC100HOME)/bin/wingui.lib`** from the list of library modules.

CHAPTER 6 Running Tecplot with Add-ons (UNIX and Windows)

When Tecplot is started, it goes through various initialization phases, including the processing of the **tecplot.cfg** file, the loading of the Tecplot stroke font file (**tecplot.fnt**), and the initialization of the graphics. After all of this has been completed, Tecplot looks for add-ons.

6.1. Specifying Which Add-ons to Load

You can customize lists of add-ons to be loaded by different Tecplot users on your network, or by a single user by starting Tecplot with different commands.

6.1.1. Add-ons Loaded by All Users

In a normal installation of Tecplot, the add-ons you want loaded by all users of Tecplot are named in an add-on load file called **tecplot.add**, located in the Tecplot Home Directory. The only command allowed in a **tecplot.add** file is the **$!LoadAddOn** command. The following is an example of a typical **tecplot.add** file:

```
#!MC 100
$!LoadAddOn "myaddon"
```

6.1.2. Specifying a Secondary Add-on Load File

You may also instruct Tecplot to load a different list of add-ons by naming a second add-on load file using one of the following methods:

- Include **-addonfile** *addonfilename* on the command line, or
- Set the environment variable **TECADDONFILE**.

Both of these methods tell Tecplot the name of another add-on load file to process.

6.1.3. Specifying Add-ons on the Command Line

You can also instruct Tecplot to load a particular add-on via the command line. Simply include the file name (include path and extension) of the add-on on the command line. You may specify as many add-ons on the command line as you want. After add-ons are loaded, Tecplot reprocesses all command line arguments not processed earlier (for graphics and add-on initialization). This ordering allows for a data reader add-on (discussed later) to be used to load data specified on the command line.

6.2. Using the $!LoadAddOn Command

The **tecplot.add** file is a special macro file that is executed at startup time and contains one or more **$!LoadAddOn** commands to load add-ons into Tecplot. **$!LoadAddOn** is, in fact, the only macro command allowed in a **tecplot.add** file. The syntax for the **$!LoadAddOn** command is:

$!LoadAddOn "*libname*"

where

libname .....The name of the shared object library file or DLL (see below). This must be in quotes.

Special rules govern how *libname* name is specified. In all cases the filename extension is omitted.

If you assign *libname* to just the basename of the shared object library then Tecplot will do the following:

UNIX:

The shared library to load will come from the file specified by:

Tecplot-Home-Directory/**lib/lib**+*basename*+*platform-specific-extension*

Where *platform-specific-extension* is **.sl** for HP platforms and **.so** for all others.

WINDOWS:

The add-on *basename***.dll** will be searched for in the following directories (in this order):

1. The directory where the Tecplot executable resides.
2. The Windows system directories.

3. The directories in your **PATH** environment variable.

If an absolute pathname is used in *libname*, then in Windows, .**dll** is appended, and in UNIX .**so** or .**sl** is appended.

6.3. Specifying Add-ons under Development

Bugs in an add-on can cause Tecplot, and all other add-ons loaded into it, to crash. While you are developing an add-on, we recommend that you keep it isolated so that other users of Tecplot in your network are not forced to use something that is potentially unstable. The way this is accomplished is somewhat different under UNIX and Windows.

6.3.1. Developing Add-ons in UNIX

We recommend that each add-on developer set up a separate "Add-On Development Root Directory." Below this directory you will create a separate sub-directory for each add-on. You then create a file called **tecdev.add** and put it in the Add-On Development Root Directory. Entries in this **tecdev.add** file must look like:

```
$!LoadAddOn "|$TECADDONDIR|/lib<myaddon>"
```

Where *myaddon* is the name of the add-on you are developing. To launch Tecplot so that it will load your add-on that is under development, you use:

```
tecplot -develop
```

This launches Tecplot in a manner such that the **tecdev.add** file in your Add-On Development Root Directory is processed and also sets up the environment variable **TECADDONDIR** so the specific add-on for your platform can be found. If you don't want to load the standard add-ons listed in the main **tecplot.add** file (located in the Tecplot Home Directory), then include the **-nostdadd-ons** flag on the command line.

See Chapter 2, "Creating Add-ons under UNIX," for more details about developing add-ons.

6.3.2. Developing Add-ons in Windows

If you're using Developer Studio to develop your add-ons in C or C++, we recommend the following setup. The directions below assume that the add-on **MyAddon** is being developed in **c:\dev\MyAddon**, and that the **TEC100HOME** environment variable has been set.

1. In Visual C++, select **Settings**... from the Project menu.
2. Click the **Debug** tab.
   (a) Set "Executable for Debug Sessions" to be the full path of **`tecplot.exe`**.
   (b) Set "Program Arguments" to be **`-loadaddon MyAddon.dll`**
3. Click the "**Custom Build**" tab, and select "**All configurations**".
   (a) Set "**Build Commands**" to **`xcopy /d /q $(TargetPath) $(TEC100HOME)\bin`**
   (b) Set "**Output files**" to **`$(TEC100HOME)\$(TargetPath)`**

This will copy your DLL from **`c:\dev\MyAddon\Debug (or \Release)`** to the **`TEC100HOME\bin`** directory after each build, so that when you select **Build\Go**, Tecplot will load and run your add-on. See Chapter 3, "Creating Add-ons under Windows," for more details about developing add-ons.

CHAPTER 7 *Add-on Initialization and Cleanup*

7.1. Add-on Initialization

When Tecplot loads an add-on, it makes a call to initialize the add-on. This function must be named **InitTecAddOn**. The function **TecUtilAddOnRegister** is the only function that is required to be called from your initialization function. The following example shows **TecUtilAddOnRegisterAddOn_pa AddOn_id** being called from the initialization function.

```
AddOn_pa AddOnId;
void InitTecAddOn(void)
{
   TecUtilLockOn();
   AddOnId = TecUtilAddOnRegister(100,
                    "CFD Add-On",
                    "V1.0-09/10/98",
                    "Amtec Engineering Inc.");
   TecUtilLockOff():
 }
```

Note this is the only instance where you must use the older TecUtilLockOn and TecUtilLockOff functions. **TecUtilAddOnRegister** registers information which is then accessible to the user via the "Help/About Add-Ons" menu option. It also informs Tecplot of the version of Tecplot (100) that your add-on was built for.

In addition to **TecUtilAddOnRegister**, one or more of the following function calls are almost always found in the add-on initialization function:

```
TecUtilImportAddConverter()
TecUtilImportAddLoader()
TecUtilMenuAddOption()
TecUtilCurveRegisterExtCrvFit()
```

Each of the above function calls, in one way or another, adds a means by which the user can access the add-on via the Tecplot interface. **TecUtilMenuAddOption** will add a menu option (currently confined to the Tools menu). `TecUtilImportAddConverter` and `TecUtilImportAddLoader` register the add-on as a special type of add-on that is used to load non-

Tecplot format data into Tecplot. The user gains access to these options from a scrolled list of loaders and converters that is launched when the File/Import menu is selected. `TecUtilCurveRegisterExtCrvFit` registers the add-on as a special type that is used to extend Tecplot's XY-plot curve fit capability. The user gains access to these curve fits from the Curve Type option on the Curve Attributes dialog. At initialization time, an add-on may also elect to launch one or more dialogs immediately in addition to, or in place of, adding menu options to the Tecplot interface.

7.2. Add-on Cleanup

When the user elects to quit Tecplot, the following happens:

1. Tecplot queries all add-ons (which have previously made a call to **`TecUtilQuitAddQueryCallback)`** to determine if their status permits terminating the Tecplot session. Tecplot shutdown only occurs if all add-ons agree that it is okay to do so. If an add-on does not want to terminate then it brings up an error-message dialog informing the user of the problem (and returns FALSE to the call).
2. If step one is successful, Tecplot will call all of the registered state change functions in add-ons with StateChange_QuitTecplot. An add-on can register a state change function by calling **`TecUtilStateChangeAddCallback`**. When an add-on receives a state change callback with **`StateChange_QuitTecplot`**, it must assume that the add-on has already given permission to terminate and should only do things like free previously allocated memory, close open files, etc.
3. Tecplot cleans itself up and terminates.

CHAPTER 8 *Tecplot GUI Builder*

The Tecplot GUI Builder (TGB) is a tool for building a platform-independent graphical user interface for a Tecplot add-on. It is not necessary to use TGB—you can use other commercial graphical layout tools. However, using TGB will allow you to quickly generate platform-independent user-interface code in C, C++ or FORTRAN.

The remainder of this document presumes you have already run **`CreateNewAddOn`** under UNIX or Mac OS X, or the Tecplot Add-On Wizard under Windows, to get your add-on development started. (See chapters 2 and 3 for more details on using these utilities.)

8.1. New in Tecplot GUI Builder 4.0

Updated TGB options include:

- **TGB API is now part of Tecplot.**

The TGB API has been fully merged with the TecUtil API. It is no longer necessary to link with a separate TGB library. All of the TGB GUI_* functions have been renamed TecGUI*, and are provided in Tecplot itself. The old API functions are still provided for backward compatibility, but you must use the new API to use the new TGB features described below.

- **Redesigned GUI Builder Add-on.**

The TGB Gui Builder add-on has been enhanced to use bitmap buttons and menus, and has a more streamlined design. Also included in the new GUI builder add-on is a “Preview Dialog” function, and a set of Widget alignment options, such as “space evenly across”, and “space evenly down”. When designing an add-on with the TGB, you may also “anchor” the dialog controls to the paper. This allows you to resize a dialog while keeping all of the controls in a fixed location.

- **Sidebars**

Add-ons may now create sidebars that can be swapped with Tecplot’s sidebar. Sidebars are designed using the TGB in a way very similar to using the TGB to design dialogs.

- **Bitmap Buttons**

Create bitmap buttons directly in the TGB. Bitmap buttons can be created using “bmp, png, or jpeg” bitmaps of any color depth The bitmap data is written directly in the guibld.c file as a static byte array, so button image files are not needed at runtime.

- **Bitmap toggles:**

Bitmap toggles are identical to bitmap buttons, however they stay in the “pushed” state when selected. Your add-on receives a value changed callback exactly as it would with a non-bitmap toggle.

- **Tooltips:**

A Tooltip provides pop-up help when the cursor hovers over the control. Any TGB control may now have a tool tip associated with it. Tooltips are particularly useful for bitmap buttons and toggles. Tooltips may be globally enabled/disabled at any time with the **`$!INTERFACE SHOWTOOLTIPS=<Boolean>`** macro command or by changing the “Show Tool Tips” toggle in Tecplot's “Performance” dialog.

- **Menu status line:**

TGB Menu items may have descriptive text which is displayed in the Tecplot status line whenever the menu selection is highlighted. See the TecGUIMenu* API functions for more information.

- **Optional action area buttons:**

The “Okay/Close”, “Help”, “Cancel” and “Apply” action area buttons are no longer required. You may select any set of action area buttons when designing a TGB dialog.

`Tabtest` and `Sidebartest`, sample TGB add-ons, demonstrate how to use TGB’s new features.

8.2. Migrating Add-ons From Tecplot 10 Release 1 to Release 3

Add-ons written using the Tecplot GUI Builder (TGB) should be updated if you plan on doing further development of these add-ons with Tecplot 10 Release 3. All other add-ons do not need any modifications.

For the most part the change involves renaming all functions starting with "`GUI_`" to "`TecGUI`", plus a couple of other minor changes.

Steps to Update

1. Make a backup copy of the source code for our add-on.
2. Go to the source code directory for your add-on.
3. Run: `UpdateAddOn`

   This script is available under UNIX only. Under Windows, you must edit all source code that contains calls to `GUI_` functions and replace "`GUI_`" with "`tecGUI`".

   If add-on is FORTRAN then rename "`GUIF_`" to be "`TecGUI`".
4. If your add-on uses the Tecplot GUI builder then bring up Tecplot with the new GUI builder and choose **Build**.
5. If you make use of `adkutil.c` functions then obtain the latest copies of `adkutil.c` and `ADKUTIL.h` from any sample add-on in the Tecplot distribution that uses this module.
6. (UNIX only) Edit `Makefile` and remove the reference to `$(GUILIB)` from the link instruction.
7. (Windows only) Remove reference to `libgui.lib` in the link instructions.
8. Note that in place of `GUI_ListDeallocItemList`, you should use `TecGUIArrayDealloc`.

8.3. Using Tecplot GUI Builder

TGB is an add-on which generates the C, C++ or FORTRAN source code used to create dialogs and controls (push buttons, text fields, scales, and so on).

Dialogs are laid out by creating frames in Tecplot and adding text and geometries to the frames. TGB distinguishes among different controls based on the style of the text and on keywords that appear in the text. TGB's controls allow you to place new controls in a dialog easily, without requiring you to remember the particular text style for each type of control.

If you have enabled TGB from your **`tecplot.add`** file, it will be accessible via the Tools menu in Tecplot.

8.3.1. How TGB Works

Figure Figure 8-1 shows the main steps in building a graphical user interface for your add-on using TGB. These steps are:

1. Using Tecplot and TGB (accessed via Tecplot's Tools menu), create or open an existing layout file that stores all information needed to define dialogs and the controls that go into the dialogs.

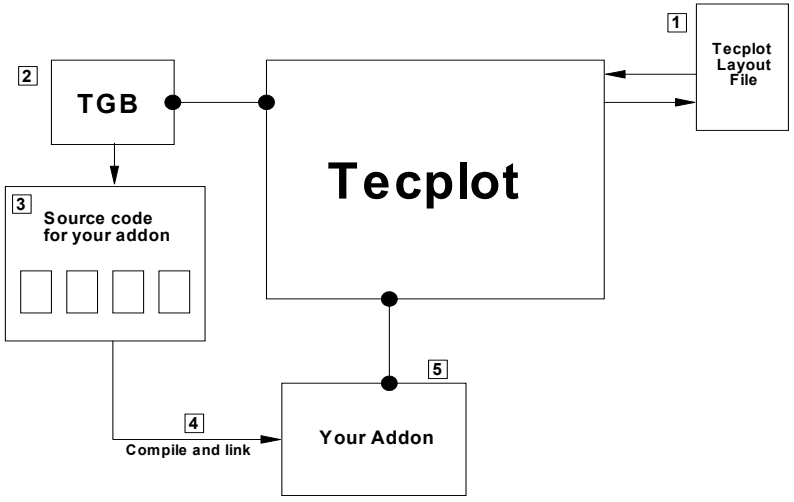


Figure 8-1. Building a graphical user interface using TGB.

2. On the TGB dialog, select a language and the control buttons you need, then click the **Go Build** button. TGB generates source code (FORTRAN, C++, or C) to operates the controls on your GUI.
3. Modify the source code files as desired.
4. Compile and link the source code to create a shared library add-on.
5. Inform Tecplot of your new add-on.
6. Restart Tecplot—your add-on is attached.

You can repeat steps 1 through 6 as needed to make modifications to the graphical user interface for your add-on. The rest of this chapter describes each step above in detail.

8.4. Step 1: Building and Maintaining the GUI

If you used the Tecplot Add-on Wizard or **`CreateNewAddOn`** shell script to create your add-on, there will already be a default **`gui.lay`** file in your add-on directory, along with a number

of default source code files, unless a Simple Callback has been chosen in CreateNewAddOn. You are now ready to modify and/or extend the default GUI.

The following sections describe how to create and add controls to the dialogs. Before you add dialogs or controls to your GUI you must first start Tecplot and open the layout file that defines your GUI.

8.4.1. Adding Dialogs

This section assumes you are running Tecplot, have loaded the layout file defining your GUI, and have the Tecplot GUI Builder dialog up on the screen.

Dialogs are added by choosing either the Modeless Dialog or Modal Dialog buttons from the TGB dialog. This will create a new frame in Tecplot. The frame will have a default size and position. The frame name contains three important pieces of information that determine characteristics of the dialog to be generated. Different keywords are used in the frame name. These keywords are listed below along with a description.

Keyword	Default	Description
ID=*n*	Dynamic (TGB automatically generates a unique number)	The dialog ID is assigned to be n. Do not change this once you start generating source code. Each dialog must have a unique dialog ID.
TITLE=*string*	“Untitled”	The title at the top of the dialogue.
MODE=*mode*	Dynamic (the mode is set based on the type of dialog selected from the GUI Builder	MODE can be either "MODAL" or "MODELESS".
OKCLOSEBUTTON=*boolean*	TRUE	Includes an okay/close button in the dialog.
CANCELBUTTON=*boolean*	TRUE	Includes a cancel button in the dialog (applies to modal dialoges only).
HELPBUTTON=*boolean*	TRUE	Includes a help button in the dialog.
APPLYBUTTON=*boolean*	FALSE	Includes an apply button in the dialogue.

In addition, the frame title may also begin with a comment enclosed in square brackets. The text inside the brackets is ignored by TGB. This is useful if you have a lot of dialogs and want to quickly identify and pop them using Tecplot's frame ordering function.

For example, the frame title could be:

```
[This is the main dialog] ID=1 TITLE="Main Dialog" MODE=MODAL
```

The comment shows up first in the frame ordering dialog, so you can quickly see which dialog this frame represents.

You can edit the frame name by double clicking on the frame's edge in Tecplot and editing the frame name text field. (Or choose the Edit Current Frame option from the Frame menu.)

8.4.2. Adding Sidebar Dialogs

Note: the ***sidebartest*** example add-on illustrates the TGB sidebar API described below.

8.4.2.1. About Sidebars. A *sidebar dialog* is a dialog which replaces the Tecplot sidebar. An add-on may create any number of sidebar dialogs, but only one sidebar dialog may be displayed at a time, including Tecplot's own sidebar. Sidebar dialogs are limited to the width and height of the Tecplot sidebar, and will be clipped if they are too wide or too tall. They may contain any controls available in the TGB, including Text Fields, Bitmap Buttons, etc. All controls in a sidebar receive the same callbacks that they would receive if they were in a modal or modeless dialog. They may not contain menus. An add-on may create several sidebar dialogs and switch between them using **`TecGUISidebarActivate()`**. The Tecplot sidebar may also be restored by calling **`TecGUISidebarActivate()`** with the special id value of **`TECGUITECPLOTSIDEBAR`**.

Add-ons may create sidebars which are wider or narrower than the Tecplot sidebar. The handling of different sized sidebars may be changed with the following macro command:

```
$!Interface SidebarSizing = {Dynamic | MaxOfAll}
```

If SidebarSizing is set to **`MaxOfAll`** (which is the default), the sidebar size will be set the maximum width of all registered sidebars, including Tecplot. If **SidebarSizing** is set to Dynamic, the sidebar size will be adjusted each time a new sidebar is displayed.

8.4.2.2. Creating Sidebars. Designing a sidebar in the TGB is no different than designing a dialog. To create a sidebar in TGB, click on the **Sidebar** button in the TGB dialog. A dialog will be created in the TGB which is the approximate width and height of the Tecplot sidebar. Resizing the width of this dialog is not recommended, since it represents all of the space available to create a sidebar. Tecplot will clip any sidebars that are too large before displaying them. After you have created the sidebar in the TGB, you may add any controls just as you would a modal or modeless dialog. TGB will generate the code to create the sidebar and add controls to it in a function **`BuildSidebarNN()`**, which is placed in **`guibld.c`**. This function call must be called before activating a sidebar.

8.4.2.3. Activating Sidebars. At most, one sidebar is active at any given time. Activating a sidebar deactivates the previous sidebar. The active sidebar is shown in Tecplot's main window, unless you are in print preview mode or the workspace is maximized. The user can activate sidebars through the workspace sidebar menu.

After calling **`BuildSidebarNN()`**, you can activate it by calling **`TecGUISidebarActivate(SidebarNN).`** You can activate the Tecplot sidebar by calling **`TecGUISidebarActivate(TECGUITECPLOTSIDEBAR);`**

8.4.2.4. Deactivating Sidebars. You "deactivate" a sidebar by replacing it with another one (which could be the Tecplot sidebar). To remove all sidebars including Tecplot's, call **`TecGUISidebarDeactivateAll()`**. The user can deactivate all sidebars by using the **None** option in the workspace sidebar menu.

8.4.2.5. TGB Sidebar Keywords. Sidebar dialogs are identified as sidebars in the TGB with the keyword **`SIDEBAR=TRUE`** in frame title. Normally you do not have to add this keyword. The TGB will add it for you when you click the **Sidebar** button to create a new sidebar.

8.4.2.6. Adding Controls to Dialogs. This section assumes you are running Tecplot, have loaded the layout file defining your GUI, and have the Tecplot GUI Builder dialog up on the screen.

Controls are added to dialogs by choosing any one of the controls buttons in TGB. The control is immediately added to the current frame in Tecplot. You can reposition the control and edit the text of the control.

8.4.2.7. Variable Names. Controls which generate events must have a unique variable name so that TGB can output correct source code. This variable name is specified in the "Macro Command" text field using the **`VARNAME=`** `keyword`. To change the "Macro command" text field, click **Options** on the Text Details dialog for the control.

For example, to assign the variable name "banana" to a toggle, in the "Macro Command" field, type:

```
varname=banana
```

Note that since variable names may not contain spaces, double quotes are *not* used.

8.4.2.8. ToolTips. You can add a tool tip for any control by using the new ToolTip keyword in the "Macro Command" field. For example, to add a tooltip to the "banana" variable, type the following:

```
varname=banana tooltip="This is the banana toggle"
```

Note that since a tool tip is a string, double quotes are *required*. You may add a tool tip to any type of TGB control.

IMPORTANT: In previous versions of TGB, you could specify a variable name simply by typing it into the "Macro Command" text field. This syntax is still supported, however, if you want to use a tool tip then you *must* use the **VARNAME=** `syntax` so that TGB can differentiate the variable name from the tool tip text.

For example, if the Macro Command text in an existing add-on is "`banana`", and you want to add a tool tip, you must edit the line as follows:

```
Varname=banana tooltip="This is the tooltip"
```

If a name is not assigned as the Macro Function Command, TGB assigns a name by looking at the text used for the control. Some controls must begin with a keyword identifying the control type. The following table lists the controls, the keyword, and text style that TGB uses to identify the control type:

Control (X Motif)	Control (Windows)	Keyword	Tecplot text style
Label.	Static text.	None.	Plain text (no text box).
Form.	Form.	FM:Group=NN	Filled text box.
Multi-line text field.	Multi-line edit.	T:	Filled text box (multi-line).

Figure 8-2.

Control (X Motif)	Control (Windows)	Keyword	Tecplot text style
Multi-selection list.	Multi-selection list box.	MLST:	Filled text box.
Option menu.	Combo box (drop-down menu).	OPT:	Filled text box.
Push button.	Push button.	None.	Filled text box.
Radio box.	Radio box.	7	Hollow text box.
Read only multi-line text field.	Multi-line read only.	TRO:	Filled text box.
Scale.	Slider.	SC:	Filled text box.
Set of tabs.	Property sheet.	TAB:Group=NN	Filled text box.
Single-selection list.	Single-selection list.	SLIST:	Filled text box.
Text field.	Edit.	TF:	Filled text box.
Text field with spin.	Text field with spin.	TFS:	Filled text box.
Toggle.	Toggle.	7	Plain text.
Bitmap Button	Bitmap Button	None	Image geometry
Bitmap Toggle	Bitmap Toggle	None	Image geometry

Figure 8-2.

For all controls except labels and push buttons, TGB can use the text after the keyword to determine a name for the control. This is only used if the Macro Function Command field, mentioned earlier, is not used. In order for TGB to identify the control type correctly, the keywords above must be used at the beginning of the actual text used for the control. TGB does not look for these keywords in the Macro Function Control field.

For example, if you have a toggle with the text **7 Include Banana**, TGB will name the control "`Include Banana`." Note that **7** signifies toggles and the radio box because it resembles a check box when it is displayed on the screen. For labels, TGB just uses the label text. By default, variable names are limited to 30 characters. To remove this limitation, select Layout/Options in the GUI Builder dialog and uncheck the "Limit variable name length to 30 characters" toggle. Note that if you uncheck this option on an existing add-on project, you may have to edit your `guicb.c` file before compiling your add-on, since the variable names may change with the longer length limit.

8.4.2.9. Adding Group Boxes and Separators. Group boxes are rectangles that surround groups of controls in a dialog. A group box can be added by simply adding a rectangle geome-

try to a frame in Tecplot. In addition, you can add a label to the group box. This is done by adding the text for the label into the Macro Function field for the rectangle geometry.

Horizontal and Vertical separators can also be added by creating a simple two point line segment geometry that is either horizontal or vertical. Note that you can press the “H” or “V” keys on the keyboard while drawing the line segment and Tecplot will force it to be horizontal or vertical, respectively.

8.4.3. Bitmap Buttons

About Bitmap Buttons A "Bitmap button" is a button which contain a bitmap rather than text. Bitmap buttons have two sizes:

1. The dimensions of the button itself.

2. The pixel dimensions of the bitmap which is to be placed in the button

When you run an add-on, Tecplot will create a bitmap button as follows:

1. A button will be created using the button dimensions.

2. A bitmap will be created using the pixel bitmap dimensions and centered in the button.

Note that the bitmap is never resized to fit onto the button. If the bitmap is too large, then it will be clipped to the size of the button. If it is too small, then there will be additional empty space on the button.

When you add a bitmap button in TGB, TGB will create an image geometry of the appropriate size for the bitmap dimensions. Resizing the button is not recommend. You may resize this button, but note that the original bitmap size does not change and will be centered on the button, or clipped if it is too large.

8.4.4. Creating Bitmap Buttons in TGB

You create bitmap buttons in TGB by clicking **bitmap button**. A bitmap button in TGB is represented by an image geometry.

TGB will launch another dialog, where you can select the following:

Bitmap file name: Enter the file name of the bitmap. Note that this file is not needed when you run your add-on. It is only needed when TGB generates source code for the add-on. TGB will generate a static array of bytes in the guicb.c file which represent the bitmap. The bitmap is generated in true color (24bit RGB) format.

Tool tip text: It is recommended that bitmap buttons have Tooltip text associated with them, since it is not always clear to users what a bitmap button does. The text you enter in the "`tool tip`" field will be added to "`Macro Command`" text field of the image geometry in the form:

```
Tooltip = "Tool Tip Text"
```

TGB adds this for you only as a convenience. You may edit the tool tip at any time after creating the bitmap button by editing the "Macro Command" callback of the image geom..

Button Type: Select a bitmap button or bitmap toggle.

Use transparent color: Most bitmaps have a transparent color. Check this toggle to enable the bitmap to have a transparent color. The transparent color will be replaced by the button background color when you run your add-on.

Transparent color: Enter the transparent color in as RGB in 6 digit hex format (similar to HTML).

Examples: Red: FF0000 Black: 000000, White: FFFFFF, Blue: 0000FF

8.4.5. Bitmap Toggles

Bitmap toggles are identical to bitmap buttons, except that they will stay "pushed" when selected, with the pushed state representing "toggle on". To create a bitmap toggle, press the "bitmap toggle" button in TGB.

8.4.6. Adding Form Controls

A form control is a rectangular region of a dialog which can show and hide different sets of controls at different times.

A *parent* form control is a rectangular region of a dialog. A form page is a set of dialog controls which can be shown or hidden inside the parent form control. Form pages are shown as

separate dialogs in TGB. You can add controls to them just as you would a regular TGB dialog. The difference is that the size of a form page dialog is always exactly the same as the size as its parent form control. Child form dialogs in TGB have a cyan background in order to distinguish them from normal dialogs.

To create a new form in a TGB dialog, click **Form**. This will create a new control on the dialog with the type **`FM:Group=NN`**, where **`NN`** is an automatically generated link group number. *Do not edit this text or the group number*, since it is needed by TGB to identify the control as a form and to identify the form pages associated with this control. When you add a new form to a dialog it is initially empty. In order to create sets of controls, you must add form pages.

To add a form page, click **Form Page**. *The button is active only if you have selected a form control*. This will create a new dialog in TGB which is actually a form page. This new dialog (a Tecplot frame) will have the same link group number as the **`Group=NN`** text in its parent form control. It will also be exactly the same size as the form control it is linked to. *Do not edit or remove the group linking from this frame*, or it will not be recognized by TGB as a form page. In the frame name will be an additional keyword, **`FORMPAGE=T`**, which identifies this dialog as a form page. See Section 1.3.1.2 for a complete list of keywords for form and tab pages.

You may add any number of form pages to a form control. Controls are added to form pages exactly like dialogs and they are built by TGB exactly like dialogs.

In your source code, TGB will generate a variable representing each form page using the parent form. This variable is similar to the **`Dialog`*`N`*`Manager`** variable that TGB creates for each dialog.

To set the controls for a form control to this set, call:

TecGUIFormSetCurrentPage(Form*N*_*G*Manager)

Where **`G`** is the group number associated with the form and **`N`** is the form page number.

For a sample TGB Add-on which uses forms, tabs and spin controls, see the source code for **`Tabtest`**.

Note: An add-on can have no more than twenty form or tab controls. This is because forms and tabs make use of frame-linking, which is limited to twenty distinct linking groups. Each form or tab may have an unlimited # of pages, however.

8.4.7. Adding Tab Controls

Tabs are identical to forms except that tabs have an additional set of controls above the parent form area. The form is automatically changed for you when the user clicks a tab. Everything documented about using forms also applies to tab controls. Individual tab pages are added using the Tab, and Tab Page buttons on the TGB dialog.

Tab pages have the additional keyword, **`POS=NN`**, in the frame name. This specifies the position from left to right. See Section 1.3.1.7 for a complete list of keywords for form and tab pages.

For a sample TGB Add-on which uses forms, tabs and spin controls, see the source code for **`Tabtest`**.

8.4.7.1. The Resize Button. If you change the size of a parent frame or tab control, you must click **Resize** on the TGB dialog. This resizes all of the linked forms or tab pages to reflect the new size of the parent form or tab. If you do not resize the tab or form pages to match the new size of the parent control they will not be sized correctly in the final GUI.

8.4.7.2. New Frame Name Keywords. Generally, you never have to specify the **`FORMPAGE`**, **`TABPAGE`**, and **`POS`** keywords yourself. They are automatically generated when the appropriate TGB buttons are clicked. You should *not* edit these keywords manually. However, if you wish to reorder a set of tab pages, you may edit the **`POS=`*`n`*** value. Note that the ordinal *n's* do not have to be consecutive, since tab pages are sorted by TGB in ascending order before generating source code. For example:

POS=1, POS=8, POS=10

is equivalent to

POS=1,POS=2,POS=3

You will want to change the **`TITLE`** association with the tab. This is done using the Edit Current Frame dialog.

Each form and tab page frame must also have a linked group number, allowing TGB to associate a set of pages with a parent control. This is done automatically when you use Add Form, Add Tab, Add Form Page, or Add Tab Page buttons.

Keyword	Default	Description
FORM-PAGE=boolean	FALSE	**TRUE** if this dialog is a form page.
TABPAGE=boolean	FALSE	TRUE if this dialog is a tab page.
POS=n	None.	If this dialog is a form page, the NN is its position, with NN=1 at the left.
TITLE="string"	Page n.	Title of the tab page.

8.4.7.3. Text Field Spin Controls. In TGB a text field spin control is a text field with two small arrow buttons anchored at the right end of the text field control. Spin controls are interchangeable with text field controls, and may be passed to any Tec**GUI** function requiring a text field control.

In addition to the text changed callback, spin controls also receive a callback when users click **up** or **down** arrows. It is up to the add-on to manage the text inside the control. This typically involves incrementing or decrementing a numeric value in the text control then re-displaying it. However, this is not a requirement. Spin controls may contain any text which may be changed in any way when up or down arrows are clicked.

8.4.8. New options in the TGB

8.4.8.1. Selecting a language. To select a language, go to Layout/Options…

You may select C, C++, or FORTRAN

The only difference between C and C++ is that C++ will generate files with the .cpp extension. Internally, the files are identical with the C files. In the remainder of this document, reference to .c/.cpp files are interchangeable.

8.4.8.2. Anchor Controls To Paper. When you select this toggle, all of the controls will be anchored to the paper. This allows you to resize a dialog without changing the position of any

controls. Note that you should check this toggle only when you are resizing the dialog. Normally it should be unchecked.

8.4.8.3. Alignment Options. To align controls using the alignment options, select 1 or more controls and press the appropriate alignment button. Note that, when appropriate, the first control selected controls the alignment. For example, clicking the **Align Left** button will align all of the controls based on the left margin of the first selected control.

8.4.8.4. Previewing Dialogs. You may preview a dialog by selecting it and clicking the **Preview** button on the TGB dialog. Previewing is useful for checking the layout of a dialog before generating the source code. Only one dialog may be previewed at a time, and the only buttons which have any effect on a preview dialog are the action area buttons "Ok", "Cancel", etc.

8.4.8.5. Note on Previewing Dialogs and System Resources. Each time you preview a dialog, that dialog's resources are created from scratch. Since dialog resources are not released until you exit Tecplot, excessive previewing during the same Tecplot session will gradually reduce the available resources. Normally this is not an issue unless you preview a dialog an excessive number of times. However, when designing a dialog in TGB, we recommend that you periodically close and restart Tecplot if you are frequently previewing dialogs.

8.5. Step 2: Building the Source Code

To build source code,, click **Go Build** at the bottom of the dialog. TGB will generate the source code for your GUI and at the same time update the Tecplot layout file so it reflects the changes you have made. You can now exit Tecplot, merge the generated GUI code with the previous GUI code, and compile your add-on.

TGB-Generated Text Files When you finish laying out one or more dialogs, save them as a Tecplot layout file, and click **Go Build** at the bottom of the TGB dialog, TGB creates the following files:

C language:

- **`guicb.tmp:`** Template for the callback module.
- **`guibld.c:`** Interface builder module.
- **`GUIDEFS.h:`** Include file naming all of the controls plus some other stuff.
- **`guidefs.c:`** Contains definitions of global variables.

FORTRAN language:

- **guicb.tmp:** Template for the callback module.
- **guibld.F:** Interface builder module.
- **GUICB.INC:** Include file naming all of the callback functions.
- **GUIDEFS.INC:** Include file naming all of the controls plus some other items.

C++ language:

- **guicb.tmp**
- **guibld.cpp**
- **GUIDEFS.h**
- **guidefs.cpp**

The file **guicb.tmp** is the template for the **guicb.c** (or **guicb.f**) module you will be editing to customize all of the callbacks generated by your interface. A callback is a function that is called when the user interacts with one of the controls in your dialogs. The first time you run TGB, just rename **guicb.tmp** to **guicb.c** (or **guicb.f**). Section 8.6.1, "Adding or Removing Controls," goes into detail on what the **guicb.c** module is and how to modify it.

The files **guibld.c** and **guibld.f** are the C and FORTRAN interface build modules. You should never edit these as they simply reflect any changes made to dialogs or controls in your interface.

Note: You should never edit the file **guidefs.c** or the include files, **GUIDEFS.h** (C language), **GUIDEFS.INC** and **GUICB.INC** (FORTRAN language). If you ever modify any of these files, be aware that they will be overwritten the next time you run Tecplot GUI Builder.

8.6. Step 3: Modifying Your Source Code

The file **guicb.c** (or **guicb.F** for FORTRAN) contains the functions called whenever a control (that is, a button or a text field) in your interface is operated by the end user. For example, suppose you have a push button that is labeled "Eject." TGB will then create code for a function called **Eject_BTN_CB_D1** that is called when the button is pressed. TGB names the functions according to some base string that you provide ("Eject" in this case, see Section 8.4.2, "Adding Sidebar Dialogs," above) plus some other decoration to uniquely identify the function. Here **BTN_CB** means this is a push button callback and **D1** means the button resides in dialog number 1.

8.6.1. Adding or Removing Controls

If you later decide to make changes to the interface, and the changes involve more than just the placement of controls or shape of the dialog, you must make changes to the **guicb.c** or **guicb.F** file.

For example, if you add a new push button to a dialog you would perform the following steps:

- Look at the **guicb.tmp** template file that is generated. It contains a new callback function for the new button.
- Cut and paste this new function from **guicb.tmp** to the existing **guicb.c** or **guicb.F** file. You can then add code to carry out the button press action.

If you remove a control from a dialog, it is not necessary to edit **guicb.c** or **guicb.F**. However, if you do not, you will end up with a callback function that is never called.

If you rename a control, you should look at **guicb.tmp** and see how TGB has now named things, then edit **guicb.c** or **guicb.F.** Change the name of the callback function to match.

Special Coding For Option Menus Option menus require special callback coding.

8.6.2. Dynamic Option Menus:

In addition to specifying a static string for the options, you may also call the new dynamic option menu functions. These dynamically add and remove strings from the option menu at run time.

The new dynamic option menu APIs are similar to the **TecGUIList** APIs. For example, **TecGUIOptionMenuDeleteAllItems()** removes all items in an option menu. (See the API reference for further information.)

8.6.3. Static Option Menus

Using C/C++:

When generating interface code in C, TGB creates a static string in the **guicb.tmp** file that is used to store the options for the option menu. For example, if you have an option menu control with the name “Fruit” then **guicb.tmp** will contain the declaration:

```
static char *Fruit_OPT_D1_List = "Option 1,Option 2,Option 3";
```

After transferring this to **guicb.c** you can edit the string and put the items you want to appear in the option menu in the static string. Separate items with a comma. For example, the resulting declaration in **guicb.c** may appear as:

```
static char *Fruit_OPT_D1_List = "apple,banana,orange";
```

Using FORTRAN: Option menu coding in FORTRAN is different than C, because TGB does not give you any hints as to what to add. The procedure is the following:

- Find the spelling of the character string created to hold the option menu items. It will be located in the file **GUIDEFS.INC**. For example, if an option menu to assign colors is named **coloropt** and is in the first dialog, then you will find the variable **coloropt_OPT_D1_List** in **GUIDEFS.INC**. It is of type **character*100** by default.
- Put all assignments for the character strings that define the option menus into the **InitTecAddon** function. It is critical that this assignment is made at the very beginning. If the color choices are “red,” “blue,” and “green,” then the statement to add to the initialization function is as follows:

```
Subroutine IntTecAddOn()
    .
    .
    .
Call TecUtilLockOn()
coloropt_OPT_D1_List = "Red,Blue,Green"
```

8.6.4. Adding a Menu Bar to a Dialog

Menu Bars currently must be added by hand. A menu bar is constructed as follows:

- Call **TecGUIMenuBarAdd**.
- For each menu option to add to the menu bar call **TecGUIMenuAdd** using the ID of the menu bar as the parent.
- For each menu item added to a menu option call **TecGUIMenuAddItem**.

Other **TecGUIMenu** functions are available to add things such as toggled menu items and to modify the menu structure once it is in place. Also note that **TecGUIMenuAdd** can be used to create walking menus by using another menu as the parent instead of the menu bar.

The menu creation code must only be executed once and should be done so immediately after the creation of the dialog. The best place to put the code is just after the call to **`BuildDialog()`** for the dialog. The example below demonstrates how to do this in a way that guarantees the menu bar code will only be executed once.

Create a menu bar that has the following menu structure:

```
Main Menu Bar
 +-> File
     +-> New Project ..
     +-> Open Project ...
     +-> Save Project ...
 +-> Setup
     +-> Solver Setup ...
     +-> Reference Values ...
     +-> Define Output
         +-> Print ...
         +-> Integeration ...
         +-> History Plot ...
         +-> Solution Plot ...

... in the callback to launch the dialog....

if (Dialog1Manager == BADDIALOGID)
  {
    BuildDialog1(MAINDIALOGID);

    MenuBar = TecGUIMenuBarAdd(Dialog1Manager);

    FileMenu = TecGUIMenuAdd(MenuBar,"File");
    NewProject_item = TecGUIMenuAddItem(FileMenu,"New Project...",
                                        NewProject_MN1_D1_CB);
    OpenProject_item = TecGUIMenuAddItem(FileMenu,"Open Project...",
                           OpenProject_MN1_D1_CB);
    SaveProject_item = TecGUIMenuAddItem(FileMenu,"Save Project...",
                                         SaveProject_MN1_D1_CB)

    SetupMenu = TecGUIMenuAdd(MenuBar,"Setup");

    SolverSetup_item = TecGUIMenuAddItem(SetupMenu,"Solver Setup...",
                                         SolverSetup_MN2_D1_CB);

    ReferenceVal_item = TecGUIMenuAddItem(SetupMenu,"Reference Values...",
                                          ReferenceVal_MN2_D1_CB);

    DefineOutput_menu = TecGUIMenuAdd(SetupMenu,"Define Output");
    PrintOutput_item = TecGUIMenuAddItem(DefineOutput_menu,"Print...",
                                         PrintOutput_MN2_D1_CB);
    IntegrationO_item = TecGUIMenuAddItem(DefineOutput_menu,"Integration...",
                                          IntegrationO_MN2_D1_CB);
    HistoryPlotO_item = TecGUIMenuAddItem(DefineOutput_menu,"History Plot...",
                                          HistoryPlotO_MN2_D1_CB);
    SolutionPlot_item = TecGUIMenuAddItem(DefineOutput_menu,"Solution Plot...",
                                          SolutionPlot_MN2_D1_CB);
  }
...
```

8.7. Step 4: Compiling Your Add-On

UNIX and Mac OS X: Compiling the add-on consists of running the **Runmake** shell script provided in the distribution. You can run **Runmake** with no parameters and you will be prompted for the options, or you can put the options on the command line. For example, if your platform is sgix.62, use:

Runmake sgix.62 -debug

Note: Always use the **-debug** flag when developing add-ons. Only when you are ready to make a release version use the **-release** flag. Using **-debug** puts the resulting shared library in the appropriate location so that Tecplot will know where to get it when using the **-develop** flag.

Windows: In Developer Studio, click **Build**.

8.8. Step 5: Informing Tecplot of Your New Add-On

This step is only required if you are developing add-ons under UNIX or Mac OS X.

If you have just created this TGB add-on, then you must inform Tecplot of its existence by editing the **tecdev.add** file in the add-on development root directory and adding the entry

$!LoadAddon "|TECADDONDIR|/lib*myaddon***"**

Where *myaddon* is the base name of your add-on.

8.9. Step 6: Running Your New Add-On

8.9.0.1. UNIX or Mac OS X: To run the debug version of your new add-on you must set the environment variables:

TECADDONDEVDIR=*myaddondevdir*
TECADDONPLATFORM=*myplatform*

where *myaddondevdir* is the path to the directory above your add-on projects. This is the directory from which you run **CreatNewAddOn**, to create your add-on in the first place. We recommend that you add the above environment variable settings to your **.cshrc** or **.profile** files.

myplatform is the same platform name you used with **Runmake**.

After setting up these environment variables, run Tecplot using:

```
tecplot -develop
```

Windows: In Developer Studio, click Go or press F5**.**

CHAPTER 9 *Building Data Set Reader Add-ons*

A data set reader add-on allows you to load non-Tecplot format data into Tecplot. Once registered with Tecplot, the data reader can then be accessed with the Import dialog and referenced with the **$!ReadDataSet** macro command. This then enables layout files generated by Tecplot to reference your data set reader add-on.

Data set readers are divided into two different types: “data set converters” and “data set loaders.”

9.1. Data Set Converters

A data set converter is the easier of the two to write. The main part of a data set converter is a function that knows how to read a non-Tecplot data format and turn around and write out a binary Tecplot data file. Functions are provided that make it easy to write out a Tecplot binary data file once you have read in your own data. A data set converter does not have to worry about any interface dialogs, etc. The standard Tecplot file dialogs are used and Tecplot takes care of reading in multiple files, doing partial reads, and so on. Data set converters are registered with Tecplot by making the following call:

```
TecUtilImportAddConverter(MyConverterFunction,
                          "MyConverterName",
                          "FNameExtension");
```

where

MyConverterFunction is the function that converts the data. It looks like:

```
Boolean_t  MyConverterFunction(
           const char  *DataFName,
           const char  *TempBinFName,
           char  **MessageString);
```

MyConverterFunction reads from the file *DataFName*, writes to the file *TempBinFName*, and if and only if there are any errors, places the error messages in the string *MessageString*. *MessageString* must be allocated inside *MyConverterFunction* using

TecUtilStringAlloc as follows:

```
*MessageString = TecUtilStringAlloc(Size,
                                    "Error message string");
```

You can use **strcpy** to assign an error string. For example:

```
strcpy(*MessageString, "My Error");
return FALSE;
```

MyConverterName is a unique name assigned to your data set converter. It must be less than 32 characters long. This is also used as the text in the Import dialog in the Tecplot interface.

FNameExtension is the extension you want as the default for the file dialog when it is displayed by Tecplot.

A data set converter should use *only* the following functions to write out the binary Tecplot data file:

TecUtilTecIni
TecUtilTecZne
TecUtilTecZneX
TecUtilTecDat
TecUtilTecNod
TecUtilTecEnd
TecUtilTecLab
TecUtilTecFil

These functions duplicate the capabilities of the **TECIO** functions as described in the *Tecplot Reference Manual*.

9.1.1. Example Data Set Converter

Included in the Add-On Developer's Kit is a simple data set converter which reads a comma delimited spreadsheet file. This converter is located in **TEC100HOME/adk/samples/cnvss**.

9.2. Data Set Loaders

Data set loaders are a bit more involved than data set converters. Data set loaders must supply their own user interface and must read the data and place the results into Tecplot directly. At first this seems complicated, but it is in fact a fairly straightforward process.

Typical coding for the initialization function in your data set loader will look like:

```
void STDCALL LaunchMyLoaderInterface()
{
  TecUtilLockStart(AddOnID);
  /* Collect loading parameters from the user,
   *  build a string list and call MyLoaderEngine() below.
   */
  TecUtilLockFinish(AddOnID);
}

Boolean_t STDCALL MyLoaderEngine(StringList_pa sl)
{
  Boolean_t IsOk = TRUE;
  TecUtilLockStart(AddOnID);
  /* Read the instructions from sl and load my data */
  TecUtilLockFinish(AddOnID);
  return IsOk;
}

Boolean_t STDCALL MyLoaderInstructionOverride(StringList_pa sl)
{
  Boolean_t IsOk = TRUE;
  TecUtilLockStart(AddOnID);
  /* Code here to override data source instructions */
  TecUtilLockFinish(AddOnID);
  return IsOk;
}

void STDCALL InitTecAddOn(void)
{
  TecUtilLockOn();
  AddOnId=TecUtilAddOnRegister("My 100 Loader","V1.00",
                               "My Company, Inc.");
  TecUtilImportAddLoader(MyLoaderEngine,"My Loader",
         LaunchMyLoaderInterface,MyLoaderInstructionOverride);

  /*
```

```
     * Other initialization you may want should be added here
     */

    TecUtilLockOff();
}
```

The data set loader is registered with Tecplot by calling **TecUtilImportAddLoader** in the **InitTecAddOn** function, as shown above. This identifies your data set loader by name and associates a set of callback functions for the core of Tecplot to reference.

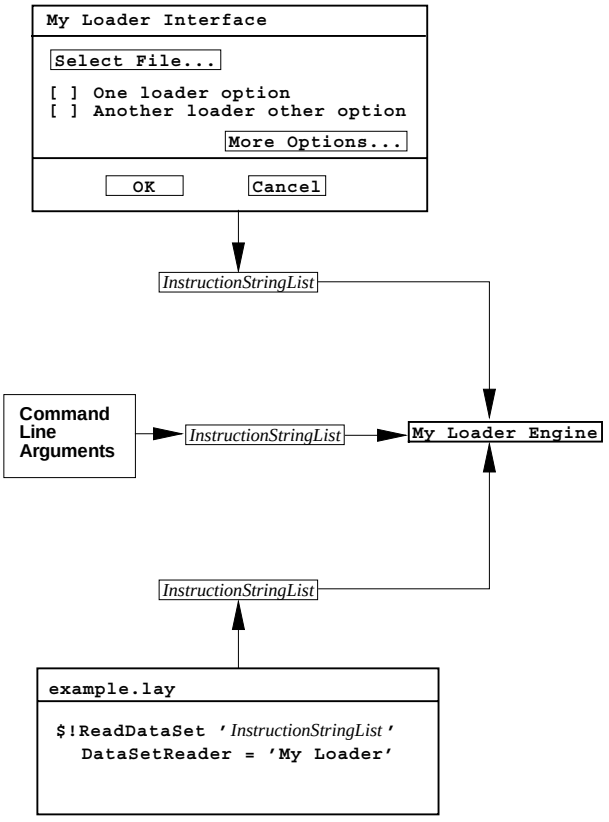
In the example above, **LaunchMyLoaderInterface** is the name of the function called by Tecplot when the user (chooses from the Tecplot interface) to load data using your data set loader. A discussion of the requirements for the interface used with a data set loader is discussed in Section 9.2.1, "The Data Set Loader User Interface."

MyLoaderEngine is the name of the function that does the actual loading of the data. It must be exported so instructions in layout files and other macros can be used to load data using your loader. This function is discussed in more detail in Section 9.2.2, "Coding the Data Set Loader Engine."

MyLoaderInstructionOverride is the function that Tecplot will call if the user is requesting to open a layout file that contains instructions to load data with your data set loader, and the user wants to override the instructions used to load the data. Use of this function is more advanced and is discussed in Section 9.2.4, "Overriding Data Set Loader Instructions." You can pass **NULL** for this parameter if you do not wish to support this capability.

Note: If your add-on uses the standard instruction syntax, then the **MyLoaderInstructionOverride** may not be necessary. See Section 8.2.5 for more detail.

Figure 9-1. How the Loader Engine gets Instructions



9.2.1. The Data Set Loader User Interface

You must provide a set (one or more) of dialogs to prompt the user for information on how and what to load into Tecplot. When the user has completed the process, create a string list (**`StringList_pa`**) variable, add the loading instructions, and pass it to the data set loader engine.

The loading instruction can be in just about any format you want, however if you follow a few simple guidelines, it will make your job a lot easier. Keep in mind that these instructions will appear in layout and macro files and so they should be somewhat readable. Athough not necessary, it is highly recommended that you follow the "Standard Instruction Syntax" because

doing so allows you to take advantage of built in services in Tecplot (see section 9.2.5.)See the sample loader mentioned in Section 9.2.7, “Data Set Loader Example,” for an example instruction string format.

The instructions as they appear in a layout file will be contained within a single string. Internally in Tecplot this string is broken down into a series of substrings where substrings are separated from one another with double quotes.

Example Instructions:

Instructions in layout	Internal representation
' "My" "Instruction" '	My
	Instructions
' "Load/X" "C:\My Documents\abc" '	Load/X
	C:\My Documents\abc
' "Fetch" "Experiment" "J 9" "Table 7" '	Fetch
	Experiment
	J 9
	Table 7

Each instruction is added to the string list that is passed to your loader function.

Data set loader instructions may not contain single quotes.

9.2.2. Coding the Data Set Loader Engine

The data set loader engine function (**`MyLoaderEngine`** in the prior example code) takes a single parameter, which is a string list that contains all the instructions necessary to identify the data to load and how to load it. The actual function prototype looks like:

```
Boolean_t MyLoaderEngine(StringList_pa sl)
```

You must use the **`TecUtilStringList`***xxx* functions to read each individual instruction. For example, if your loader uses the following set of flags:

```
-banana
-apple
-sourcefile <sourcefilename>
```

An instruction string sent to your data set loader engine could look like:

```
'"-banana""-apple""-sourcefile myfile.dat"'
```

After decoding the instructions, your loader should do the following:

1. (Optional) Call **TecUtilDialogLaunchPercentDone** to show a percent done dialog on the screen.
2. Call **TecUtilDataSetCreate** to create a new data set.
3. Call **TecUtilDataSetAddZone** for each zone created.
4. (Optional) Call **TecUtilDialogCheckPercentDone** every once in a while to update the percent done dialog.
5. Call **TecUtilDataValueGetRef** to obtain handles to the field data and then:

   Call **TecUtilDataValueSetByRef** to stuff the values (faster)
   or

   Call **TecUtilDataValueSetByZoneVar** (slower).
6. (Optional) Call **TecUtilDialogDropPercentDone**.
7. Call **TecUtilFrameSetMode** to set the frame mode.
8. Call **TecUtilRedraw** to show the first image.

See the sample loader mentioned in Section 9.2.7, “Data Set Loader Example.”

9.2.3. Appending Data with a Data Loader

A data loader can provide the capability to append data. Data loaders can also append data to an existing dataset. If your loader wishes to do this you must do the following:

1. Prior to appending it is a good idea to query the existing dataset in Tecplot to make sure it is compatible with the data you wish to append.

2. Prior to calling any TecUtil functions that modify data you must suspend dataset marking in tecplot by calling: TecUtilDataSetSuspendMarking(TRUE);

3. Do not call TecUtilDataSetCreate(). Just call TecUtilDataSetAddVar and/or TecUtilDataSetAddZone.

4. After appending data call TecUtilStateChanged (or TecUtilStateChangedX) with StateChanged_ZonesAdded and/or StateChanged_VarsAltered - depending on what you did.

5. Turn off the suspension of dataset marking:
   TecUtilDataSetSuspendMarking(FALSE);

6. As with normal loading, call TecUtilImportSetLoaderInstr to register the instructions needed for the append.

Tecplot will journal this new command along with any other existing data creation/loading commands. The suspension of dataset marking prevents the data journal in tecplot from being invalidated with the activity of adding variables or zones..

9.2.4. Overriding Data Set Loader Instructions

When opening a layout in Tecplot, you are given the opportunity to override the data source instructions for the layout file. This allows you to apply a given layout to different data. If you choose this option then Tecplot will do the following:

1. Scan the layout file and determine the instructions needed to load the data for each referenced data set.
2. Determine the data set reader type required to load each data set.
3. Allow the user to select the Reader/Instructions to override.

If you need the capability in your loader to only allow overide of the filenames then a much simpler approach is to use the standard instruction syntax (see section 8.2.5).

At this point Tecplot determines if in fact there is even a way to override the data source instructions. If your data set loader has registered a data set loader override function, then it will be called, and it is up to the function to show the user what the instructions were and allow the user to change them.

Adding this capability to your data set loader is optional. To omit this capability just supply **NULL** for the data set loader override function parameter in the **TecUtilImportAddLoader** function:

```
TecUtilImportAddLoader(MyLoaderEngine,
                       "MYLOADER",
                       LaunchMyLoaderInterface,
                         NULL);
```

If you do supply the function, then create it as follows:

```
Boolean_t MyLoaderInstructionOverride(StringList_pa sl)
{
  /*
   * First step is to decode the instructions and display the current
   * settings. Use the TecUtilStringListxxx functions to read each
   * instruction and to create a new set of instructions.
   */

   /* return TRUE if all goes well, otherwise FALSE. */
}
```

9.2.5. Using Standard Instruction Syntax

Another way to enable data set overriding with your loader which does not require writing an override function is to use a standard instruction syntax. When the standard loader instruction syntax is used, Tecplot will parse the instructions, identify file names and directory names within those instructions, and prompt the user for replacements as necessary. Once the user has selected the new names, Tecplot will insert them into the loader instructions and call the loader engine with the revised filenames and/or directory names.

An advantage of loader standard syntax is that it allows Tecplot to save loader instructions with relative or absolute path names. If a user saves a layout and elects to use relative path names, Tecplot will scan through the loader instructions for all file and directory names identified via the above mechanism, and replace them with paths relative to the layout file path, just as it does with Tecplot data file names.

Standard loader instructions are specified as tag/value string pairs in the instruction string list. That is, the "tag" string and "value" strings must be consecutive strings in the string list. Except for the standard syntax identifier tag (which must be the first string in the string list), tag/value pairs may appear at any ordinal location in the string list and may be interspersed with you own custom instructions which are ignored by Tecplot. Note that each custom instruciton must be in the Tag/Value form.

The standard syntax tags are summarized as follows:

Tag Name	Value	Required?	Notes
STANDARDSYNTAX	1.0	YES	To use loader standard instruction syntax, the loader must have the this tag/value as the first two strings in the string list.
DIRNAME [directory_name_identifier]	[Directory Name]	NO	If the loader uses a directory name as part of it's instruction string, then use this tag. Normally this tag is not used.

FILENAME_[file_name_identifier]	[File Name]	NO	If your loader uses a filename as part of it's instructions, use this tag.
FILELIST_[file_list_identifier]	[N] [FileName1, FileName2,..FileNameN]	NO	If your loader uses a list of file names, use this tag. See the notes below for more information

Notes: Any text may follow the underscore in the DIRNAME_, FILENAME_, and FILELIST_ tags.

Examples:

```
"DIRNAME_MyDirectory"
"FILENAME_FilesToLoad",etc.
```

Using the FILELIST tag

The FILELIST tag is followed by a string which contains an integer number indicating the number of file names in the list.

Example:

```
"STANDARDSYNTAX"
"1.0"
"DIRNAME_MyDirectory"
"c:\Users\Joe"
"FILENAME_MyConfigFile"
"c:\Users\Joe\MyConfigFile.cfg"
"FILELIST_FilesToLoad"
"3"
"c:\Users\Joe?le1.fil"
"c:\Users\Joe?le2.fil"
"c:\Users\Joe?le3.fil"
```

A limitation of data overriding via this mechanism is that Tecplot will prompt the user only once for all files in a particular loader instruction-the user must select all files and directories at once. For example, the above instructions contain one directory and four file names, a total of five items. A user who opens a layout containing the above instructions and elects to override the layout data would have to select 5 items in the resulting file selection dialog. If the user selected only 3 items, then only the first 3 standard items in the loader instructions would be replaced.

9.2.6. Setting Styles with Data Loaders

Setting of style is generally frowned upon in a data loader. The reason for this preference is that it is cleaner to keep the data loading focused on data loading and make the assignment of style secondary.

For example, it probably is not a good idea to turn on the contour layer and assign contour levels in a data loader, which confuses layout files. Suppose you read in data with your data loader and then set up the style for a plot. Then you save a layout file. In the layout file Tecplot writes:

- Read the data using the data loader.
- Assign style.

The assign style section assumes that the plot up to this point has a factory default style and some style settings are omitted based on this assumption.

9.2.7. Data Set Loader Example

Included in the Add-On Developer's Kit is a sample data set loader. This loader is located in **`TEC100HOME/adk/samples/loadss`**.

9.3. Accessing Non-Tecplot Format Data Sets via the Command Line

To access non-Tecplot format data via the Tecplot command line use:

```
tecplot [tecplot options] -datasetreader <readername> [reader options]
```

No Tecplot-specific options may follow the **`-datasetreader`** flag. What follows must be the name of the data set reader followed by the options for the reader itself.

Each option is placed into a separate string in the string list argument sent to the loader.

CHAPTER 10 *Building Extended Curve Fit Add-ons*

An extended curve fit add-on allows you to extend Tecplot's XY-plot curve fitting capability. Once registered with Tecplot, the extended curve fit can be activated from the Curve Attributes dialog (Curves page of the Mapping Style dialog) by selecting the maps to which it will apply, clicking Curve Type, choosing Extended from the drop-down menu, and choosing the curve fit from the list in the Choose Extended Curve Fit dialog. If the extended curve fit has settings which may be modified (optional), the dialog for modifying the settings may be accessed by clicking Curve Settings with the appropriate map selected.

The extended curve fit consists of an initialization routine and one or more callback functions, which are called by Tecplot when appropriate. These callback functions will in turn call functions to compute the curve fit from the raw data. They may also call functions to launch dialogs for entry of user configurable settings and output of the curve fit coefficients.

10.1. Registering the External Curve Fit

Extended curve fit add-ons are registered with Tecplot by calling the following function:

```
Boolean_t TecUtilCurveRegisterExtCrvFit(
              const char                              *CurveFitName,
              GetXYDataPointsCallback_pf               XYDataPointsCallback,
              GetProbeValueCallback_pf                 ProbeValueCallback,
              GetCurveInfoStringCallback_pf            CurveInfoStringCallback,
              GetCurveSettingsCallback_pf              CurveSettingsCallback,
              GetAbbreviatedSettingsStringCallback_pf
AbbreviatedSettingsStringCallback);
```

which returns TRUE if the extended curve fit was added successfully.

CurveFitName is a unique name given to the extended curve fit. This name is used in the list of extended curve fits in the Choose Extended Curve Fit dialog, launched from Extended option on the Curves page of the Mapping Style dialog.

XYDataPointsCallback is the name of the function that will calculate the curve fit. This is the only function that needs to be defined to create an extended curve fit add-on.

ProbeValueCallback is the name of the function that will return the dependent value when the extended curve fit is probed at a given independent value. If *ProbeValueCallback* is set to NULL, Tecplot will perform a linear interpolation based on the values returned by the *XYDataPointsCallback* function.

CurveInfoStringCallback is the name of the function that will create a string to be presented in the Data/XY-Plot Curve Info dialog. *CurveInfoStringCallback* may be set to NULL if you do not wish to present a string to the XY-Plot Curve Info dialog.

CurveSettingsCallback is the name of the function that is called when the Curve Settings button on the Curves page of the Mapping Style dialog is pressed while the extended curve fit is set as the Curve Type. *CurveSettingsCallback* may be set to NULL if there are no configurable settings for the extended curve fit. If settings are changed, it is the responsibility of the add-on writer to inform Tecplot of the change by calling the function *TecUtilCurveSetExtendedSettings*. This function is usually called when the OK button is pressed on the add-on dialog.

AbbreviatedSettingsStringCallback is the name of the function that will return a short version of your curve settings string. This string will be presented in the Curve Settings text field on the Curves page of the Mapping Style dialog. *AbbrevieatedSettingsStringCallback* may be set to NULL if you do not wish to assign anything to this string. Even if you do not assign anything to the *CurveSettings* string, you may define this function and return any string you wish. The Curve Settings option on the Curves of the Mapping Style dialog is roughly 50 characters wide, so strings longer than roughly 50 characters will be truncated.

Since extended curve fit add-ons aren't compatible with Tecplot versions earlier than 10, it is important to check the Tecplot version number before registering the curve fit. The version number may be obtained using the *TecUtilGetTecplotVersion* function. If this function returns a value less that 1000000, display an error message and don't call TecUtilCurveRegisterExtCrvFit.

It you are creating your add-on using the Tecplot GUI builder, and have created templates by running the Add-on Wizard (Windows) or CreateNewAddOn script (Unix), the add-on registration code (including the Tecplot version test) will be found in the module main.c, the above callback functions will be found in the module engine.c, and the curve settings dialog callback functions will be found in guicb.c (if configurable settings were requested). Typically it won't be necessary to modify main.c.

10.2. Calculating the Curve Fit

Curve fitting is modeling the data with an analytical function containing adjustable parameters (curve fit coefficients). In many cases, the values of these coefficients are computed such that the curve is "best" in some statistical sense (the Least Squares method, for example). In other cases, the values of the coefficients are computed so that the curve passes through the raw data points and represents one possible interpolation between the points (splines, for example). Occasionally, a curve fit add-on may be used to compute and display a variable derived from the raw data points (for example, see the Running Average add-on provided as a sample with Tecplot).

Tecplot represents the curve by a number of points connected by line segments. The number of points used is specified in the Curve Points field of the Curve-Fit Attributes dialog. *XYDataPointsCallback* is the name of the callback function where the curve points of the curve fit are calculated. This is the only callback function that needs to be defined to create an extended curve fit add-on. It looks like:

```
Boolean_t STDCALL XYDataPointsCallback(
              FieldData_pa RawIndV,
              FieldData_pa RawDepV,
              CoordScale_e IndVCoordScale,
              CoordScale_e DepVCoordScale,
              LgIndex_t    NumRawPts,
              LgIndex_t    NumCurvePts,
              EntIndex_t   XYMapNum,
              char         *CurveSettings,
              double       *IndCurveValues,
              double       *DepCurveValues);
```

Where:

XYDataPointsCallback is the name of your functions that calculates the curve fit.

RawIndV is the handle to the raw field data of the independent variable.

RawDepV is the handle to the raw field data of the dependent variable.

IndVCoordScale is an enumerated variable specifying whether the independent variable axis has a linear or log scale.

DepVCoordScale is an enumerated variable specifying whether the dependent variable axis has a linear or log scale.

NumRawPts is the number of raw field data values.

NumCurvePts is the number of points that will construct the curve fit.

XYMapNum is the map number that is currently being operated on.

CurveSettings is the curve settings string for the current xy-map.

IndCurveValues is a pre-allocated array of size *NumCurvePts* which the add-on will populate with the independent values for the curve fit.

DepCurveValues is a pre-allocated array of size *NumCurvePts* which the add-on will populate with the dependent values for the curve fit.

The arrays, *IndCurveValues* and *DepCurveValues* are the main result of this function call.

Generally speaking, the *XYDataPointCallback* consists of two parts: the first computes the curve fit coefficients and the second populates the curve values arrays. The process for computing the curve fit coefficients is beyond the scope of this manual. For common techniques such as linear least squares and splines, there are several good books you can refer to for theory (see Numerical Recipes, for example) and various libraries are available on the internet and elsewhere. For less common techniques, or simpler fits, you may write your own functions to compute the curve fit coefficients.

Regardless of what method you use to compute the curve fit parameters, you will need to extract data from the raw data arrays. This may be done with the TecUtilDataValueGetByRef utility. For example, with:

```
int I;
DepVar = TecUtilDataValueGetByRef(RawDepV, I)
```

DepVar will contain the Ith element of the raw dependent variable array.

Populating the dependent variable arrays is relatively easy to do. Determine the spacing of the independent curve variable with the following code

```
double IndVarMin, IndVarMax;
TecUtilDataValuesGetMinMaxByRef(RawIndV,
                                &IndVarMin,
```

```
                                    &IndVarMax);
Delta = (IndVarMax-IndVarMin)/(NumCurvePts-1);
```

Then set the independent and dependent curve variables in the following loop

```
for (ii = 0; ii < NumCurvePts; ii++)
{
  IndCurveValues[ii] = ii*Delta + IndVarMin;
  DepCurveValues[ii] = CurveFunction(IndCurveValues[ii],
   Parameters);
}
```

If you use the Add-on Wizard (Windows) or the CreateNewAddOn script (Unix), most of the code for populating the curve variable arrays is created for you. You will only need to modify the line that computes DepCurveValues[ii] (the default code sets this to the mean value of the raw dependent variable).

One final note, there are three items in the *XYDataPointCallback* parameter list that are not needed every time. The *CurveSettings* string is only needed if the curve fit has user configurable curve settings. The other two are *IndVCoordScale* and *DepVCoordScale,* which are only needed if the nature of the curve fit will depend upon the axis scale used (log or linear). Generally this is not the case. Tecplot's standard curve fits, for example, are not dependent upon the axis scale.

10.3. Improving the Probe Value

Unless something special is done, a probe of an xy-map with an extended curve fit will perform a linear interpolation based upon the *IndCurveValues* and *DepCurveValues* arrays (discussed in previous section). This will be correct at the curve points and off by some error at points between the curve points. The magnitude of this error will depend upon the curve function and the data, and it will reduce as the number of curve points increases. If this error is acceptable, set *ProbeValueCallback* to NULL in the TecUtilCurveRegisterExtCrvFit call. If this error is unacceptable, provide a *ProbeValueCallback* function, which returns the dependent variable for a given independent variable. The syntax for this function is:

```
Boolean_t STDCALL ProbeValueCallback (
             FieldData_pa RawIndV,
             FieldData_pa RawDepV,
             CoordScale_eIndVCoordScale,
             CoordScale_eDepVCoordScale,
             LgIndex_t    NumRawPts,
             LgIndex_t    NumCurvePts,
```

```
EntIndex_t   XYMapNum,
char         *CurveSettings,
doubleProbeIndValue,
double*ProbeDepValue);
```

Where:

ProbeValueCallback is the name of your ProbeValueCallback function.

RawIndV is the handle to the raw field data of the independent variable.

RawDepV is the handle to the raw field data of the dependent variable.

IndVCoordScale is an enumerated variable specifying whether the independent variable axis has a linear or log scale.

DepVCoordScale is an enumerated variable specifying whether the dependent variable axis has a linear or log scale.

NumRawPts is the number of raw field data values.

NumCurvePts is the number of points that will construct the curve fit.

XYMapNum is the map number that is currently being operated on.

CurveSettings is the curve settings string for the current xy-map.

ProbeIndValue is the value of the independent variable at the location of the probe.

ProbeDepValue is the calculated value of the dependent variable at the location of the probe, based on the value of *ProbeIndValue*.

Much of the discussion in the previous section, “Calculating the Curve Fit,” applies here as well. The main difference is that you are computing a scalar variable, *ProbeDepValue*, instead of an array. It will be necessary to recompute the curve fit parameters and call the same curve function as discussed previously.

```
*ProbeDepValue = CurveFunction(ProbeIndValue, Parameters);
```

10.4. Providing Curve Fit Information

Once a user has utilized a curve fit, they will often want to view and/or save the Parameters computed by the curve fit for their data. This information is displayed in the XY-Plot Curve Info dialog, which is launched from the Data menu. *CurveInfoStringCallback* is the name of the function that will create a string to be displayed in the Data/XY-Plot Curve Info dialog. This callback may be set to NULL if you do not wish to present a string to the XY-Plot Curve Info dialog. The syntax for this function is:

```
Boolean_t STDCALL CurveInfoStringCallback (
                 FieldData_pa       RawIndV,
                 FieldData_pa       RawDepV,
                 CoordScale_eIndVCoordScale,
                 CoordScale_e       DepVCoordScale,
                 LgIndex_t          NumRawPts,
                 EntIndex_t         XYMapNum,
                 char*CurveSettings,
                 char**CurveInfoString);
```

Where:

CurveInfoStringCallback is the name of your *CurveInfoString* Callback function,

RawIndV is the handle to the raw field data of the independent variable.

RawDepV is the handle to the raw field data of the dependent variable.

IndVCoordScale is an enumerated variable specifying whether the independent variable axis has a linear or log scale.

DepVCoordScale is an enumerated variable specifying whether the dependent variable axis has a linear or log scale.

NumRawPts is the number of raw field data values.

XYMapNum is the map number that is currently being operated on.

CurveSettings is the curve settings string for the current XY-map.

CurveInfoString is the string that is to be presented in the Data/XY-Plot Curve Info dialog.

The *CurveInfoString* must be allocated inside *CurveInfoStringCallback* using **`TecUtilStringAlloc`** as follows:

```
*CurveInfoString = TecUtilStringAlloc(Size, "CurveInfoString");
```

The string may then be written to using sprintf. In general, you should provide enough information in the *CurveInfoString* that the user can independently create the curve fit. Curve fit coefficients should be provided, for example, along with the ranges of applicability (if they aren't obvious). You may also include statistical information about the curve fit or the data, as seen with the General curve fit, which is distributed with Tecplot.

10.5. Curve Fit Settings

Extended curve fit add-ons may have user configurable settings. If so, the settings for each XY-map are saved in a character string maintained by Tecplot. Your add-on must initialize this string, update it when settings are changed, and inform Tecplot by calling TecUtilCurveSetExtendedSettings or TecUtilXYMapSetCurve (with the first parameter being "EXTENDEDSETTINGS"). Your add-on must also parse this string to extract the configurable curve fit parameters.

The curve settings string can be in just about any format you like. However, if you follow a few simple guidelines, it will make your job a lot easier. Keep in mind that these instructions will appear in layout and macro files, so they should be somewhat readable. Also, the string must not contain single quotes since the entire curve settings string is surrounded by single quotes in Tecplot layout and macro files.

CurveSettingsCallback is the name of the function that is called when the Curve Settings button on the Curves page of the Mapping Style dialog is pressed while the extended curve fit is set as the Curve Type. If *CurveSettingsCallback* is set to NULL in the call to *TecUtilCurveRegisterExterCrvFit*, there are no configurable settings for the extended curve fit. The syntax for this function is:

```
void STDCALL CurveSettingsCallback(
        Set_pa          XYMapSet
        StringList_pa SelectedXYMapSettings);
```

Where:

CurveSettingsCallback is the name of your function that launches your extended curve settings dialog.

XYMapSet is the set of XY-Maps that are selected in the Plot-Attributes dialog.

SelectedXYMapSettings is a string list of the curve settings for the XY-maps that are selected on the Mapping Style dialog.

CurveSettingsCallback usually just sets any global variables and launches the curve settings dialog.

You must provide one or more dialogs to prompt the user for the curve fit settings. It is highly recommended that this be a modal dialog to minimize the complexity of monitoring for changes while the dialog is up. When the OK button is pushed on the Curve Settings dialog, update the curve settings string and inform Tecplot by calling **`TecUtilCurveSetExtendedSettings`** or **`TecUtilXYMapSetCurve`** (with the first parameter being "EXTENDEDSETTINGS"). Remember that the settings must be modified for all XY-maps that were selected when the dialog was launched. These map numbers are provided by Tecplot in *XYMapSet*. If **`TecUtilCurveSetExtendedSettings`** is used, you must loop through the maps in *XYMapSet* and set each one. In contrast, **`TecUtilXYMapCurve`** only needs to be called once, with *XYMapSet* as an argument.

It you are creating your add-on using Tecplot GUI Builder, and have created templates by running the Add-on Wizard (Windows) or **`CreateNewAddOn`** script (UNIX) (with configurable settings requested), the **`CurveSettingsCallback`** function will be found in the module **`engine.c`**. This function, which saves **`XYMapSet`** and **`XYMapSettings`** in global variables and launches the Curve Settings dialog, will probably not need to be modified. The curve settings dialog callback functions will be found in **`guicb.c`**. See the *Tecplot GUI Builder Manual* for more information on modifying this dialog.

10.6. Creating the Curve Settings Text Field

Below the Curve Settings button in the Curve Attributes dialog is a text field providing a brief description of the curve settings for each map. This text field is set in the *AbbreviatedSettingsStringCallback* function. If you do not want to provide an *AbbreviatedSettings* string, set *AbbreviatedSettingsStringCallback* to NULL in the call to TecUtilCurveRegisterExtCrvFit.

Note: Even if you do not have configurable settings, you may define this function and return any string you wish. The Curve Settings option on the Curves page of the Mapping Style dialog is roughly 50 characters wide, so strings longer than roughly 50 characters will be truncated. The syntax for this function is:

```
void STDCALL       GetAbbreviatedSettingsStringCallback(
```

```
    EntIndex_t   XYMapNum,
    const char   *CurveSettings,
    char         **AbbreviatedSettings);
```

Where:

AbbreviatedSettingsStringCallback is the name of your function that will return the Abbreviated Settings string.

XYMapNum is the map number that is currently being operated on.

CurveSettings is the string that Tecplot maintains which contains the extended curve fit settings for the current xy-map.

AbbreviatedSettings is the short form of the *CurveSettings* that are passed into your function by Tecplot.

The *AbbreviatedSettings* string must be allocated inside the *AbbreviatedSettingsStringCallback* function using TecUtilStringAlloc as follows:

```
*AbbreviatedSettings = TecUtilStringAlloc(Size,
"AbbreviatedSetting");
```

The string may then be written to using sprintf.

CHAPTER 11 *Locking and Unlocking Tecplot*

Locking is the method by which add-ons can determine if it is safe to call TecUtil functions. All add-ons must use locking, but only add-ons with callbacks from timers or other asynchronous operations need to monitor locking. In a timer callback, when Tecplot is locked, it is not safe to call TecUtil functions (except for the TecUtilLock*xxx* functions, described below). When Tecplot is unlocked, it is safe to call TecUtil functions. Tecplot is locked if add-ons are performing operations or if the user is interacting with the Tecplot interface.

11.1. Locking Functions

The following table lists the seven add-on functions that control or monitor locking and unlocking in Tecplot:

`void TecUtilLockStart(AddonId_pa AddonID)`	Locks Tecplot. You may call **`TecUtilLockStart`** any number of times, as long as each call is matched with a call to **`TecUtilLockFinish()`**.
`void TecUtilLockFinish(AddonID_pa AddonID)`	Unlocks Tecplot. You must have exactly one call to **`TecUtilLockFinish()`** for each call to **`TecUtilLockStart()`**.
`Boolean_t TecUtilLockIsOn(void)`	Returns **`TRUE`** if Tecplot is currently locked.
`int TecUtilLockGetCount(void)`	Returns the number of levels of locking that are currently active in Tecplot. In other words, the number of calls to **`TecUtilLockStart()`** without matching calls to **`TecUtilLockFinish()`**.
`char * TecUtilLockGetCurrentOwnerName`	Returns the name of the entity currently locking Tecplot (or NULL if unknown). You must use TecUtilStringDealloc to free this string when you are finished using it.

`void TecUtilLockOn()`	Locks Tecplot anonymously. Must be used in InitTecAddOn. Should use TecUtilLockStart elsewhere.
`void TecUtilLockOff()`	Unlocks Tecplot anonymously. Must be used in InitTecAddOn. Should use TecUtilLock-Finish elsewhere

11.2. Using the Locking Functions

1. Most TecUtil functions require that Tecplot be locked. The only exceptions to this are for the four lock functions listed above and, in general for most query type functions like TecUtilFrameGetMode etc. If you are not sure you should call **`TecUtilLockStart`** at the beginning of any function that calls TecUtil functions and call **`TecUtilLockFinish`** at the end. The only exception to using the Start/End forms of the lock functions is for the initialization of your add-on (i.e. the function InitTecAddOn). Here you must use **`TecUtilLockOn`** and **`TecUtilLockOff`** because the AddOnID handle has not been set.

   Example:

```
void STDCALL InitTecAddOn()
{
   /* Since we will call a TecUtilxxx function, Tecplot must
     be locked. */
   AddonID = TecUtilLockOn();
   TecUtilAddOnRegister(100, "My Addon","1.0","My Company");
   TecUtilMenuAddOption("Tools","Simple Add-on",'S',
     MenuCallback);
   TecUtilLockOff();
}
```

2. Calls to launch modal dialogs must call **`TecUtilLockStart`** prior to the launch, and should call **`TecUtilLockFinish`** in the close/cancel button callback. If you are using the Tecplot GUI Builder (TGB) to build your interface, the TGB will generate the appropriate code; otherwise you must insert this code by hand.

   Here is a Visual C++ example:

```
void MenuCallback()
{
   /* Assumes we are using Visual C++/MFC. */
```

```
   CMyDialog dlg;
   TecUtilLockStart(AddonID);
   /* Be sure to call
     TecUtilStateChanged(StateChange_ModalDialogLaunch,NULL);
     in your override of OnInitDialog(). */
   dlg.DoModal(); /* This function will not return until the
     dialog is closed. */
   TecUtilLockFinish(AddonID);
}
```

3. Callbacks from timers or other asynchronous means that need to modify data in real time- should be coded as follows:

```
void MyTimerCallback
{
   if (!TecUtilLockIsOn())
   {
      /* No other add-on has Tecplot locked, so it is safe
        to modify internal Tecplot data. */
      TecUtilLockStart(AddonID);
      /* Modify some data here with TecUtilxxx functions. */
      TecUtilLockOff(AddonID);
   }
   else
   {
      /* Some other add-on has Tecplot locked, so do not
        do anything. */
   }
}
```

CHAPTER 12 *Modal and Modeless Dialogs in Windows*

As far as Tecplot is concerned, there are two types of dialogs — “modal” and “modeless.” *Modal dialogs* lock out the rest of Tecplot from being used. *Modeless dialogs* do not. Examples of modal dialogs are file dialogs, error messages, and the Print dialog. All of these dialogs require you to OK or Cancel them before doing anything else inside Tecplot. Examples of modeless dialogs are the Quick Edit dialog, the Rotate dialog, and the Zone Style/Mapping Style dialogs. You may have these dialogs up and still interact with the rest of Tecplot.

This modal/modeless dialog paradigm does not exactly match that used by Windows, so add-ons must inform Tecplot when a modal dialog is launched and dismissed so that Tecplot (and other add-ons) can disable its interface. Add-ons with modeless dialogs must keep track of when to enable and disable their modeless dialogs. This is easily done by using Tecplot's state change mechanism. Modal dialogs must inform Tecplot (and other add-ons) of their launch/dismissal with **`TecUtilStateChanged`**. Add-ons with modeless dialogs must monitor Tecplot's state with **`TecUtilStateChangeAddCallback`** and disable the modeless dialogs at the appropriate times.

Note: This is done for you automatically if you are using TGB for your interface.

12.1. Modal Dialogs

Whenever a modal dialog is launched, the add-on must inform Tecplot of the launch. Do this by calling

```
TecUtilStateChanged(StateChange_ModalDialogLaunch, NULL);
```

when processing **`WM_INITDIALOG`** or (in MFC) in the dialog's **`OnInitDialog`**.

Whenever a modal dialog is dismissed (closed), the add-on must inform Tecplot of dismissal. Do this by calling:

```
TecUtilStateChanged(StateChange_ModalDialogDismiss, NULL);
```

when processing **WM_NCDESTROY** or (in MFC) in the dialog's **PostNcDestroy**.

That is all that is needed for modal dialogs. If you do not call these functions, Tecplot may appear to work, but other add-ons may fail to work with your add-on.

Note: Failure to call these functions in balanced pairs (i.e., calling one but not the other) is a serious error.

Dialogs created by Tecplot itself (error messages, file dialogs, and so on; basically any function starting with "**TecUtilDialog**") already process the state change. You do not need to add state change calls when using those functions.

12.2. Modeless Dialogs

Note: The following section applies only if you are NOT using TGB for your interface.

If your add-on uses modeless dialogs like Tecplot does, you have a little more work to do. First, you must monitor Tecplot's state changes. You do this by creating a state change callback function (see chapter 12). Within the state change callback, you must put the following:

```
{
 /* When using MFC, the following line is always required on callbacks. */
 /* Do not include this line if you are not using MFC. */
  AFX_MANAGE_STATE(AfxGetStaticModuleState( ));

  static int nDisabledCount = 0;

  if ( StateChange == StateChange_ModalDialogLaunch )
    {
      nDisabledCount++;
      if ( nDisabledCount > 0 )
        {
          /* Disable all modeless dialogs, e.g., in Win32 SDK. */
          if ( hwndModelessDialog )
            EnableWindow(hwndModelessDialog, FALSE);
          /* Or, under MFC. */
          if ( modeless_dlg )
            modeless_dlg->EnableWindow(FALSE);
        }
    }
  else if ( StateChange == StateChange_ModalDialogDismiss )
```

```
    {
      nDisabledCount--;
      if ( nDisabledCount <= 0 )
        {
          DisabledCount = 0;
          /* Enable all modeless dialogs, e.g. in Win32 SDK. */
          if ( hwndModelessDialog )
            EnableWindow(hwndModelessDialog, TRUE);
          /* Or, under MFC. */
          if ( modeless_dlg )
            modeless_dlg->EnableWindow(TRUE);
        }
    }
}
```

Each modeless dialog must be individually enabled or disabled, so if there are many such dialogs, a function to enable or disable all modeless dialogs would be a good idea.

If you do not monitor Tecplot's state, your add-on's modeless dialogs will not disable themselves at the appropriate times, and the user may be able to access your add-on at an unexpected time.

Note: This could cause any number of serious problems, including a crash of Tecplot.

On the plus side, if you use both modal and modeless dialogs in your add-on, monitoring state changes for the modeless dialogs, and informing Tecplot of the launch/dismiss of the modal dialogs, will automatically make your add-on's modeless dialogs disable themselves when the add-on itself brings up a modal dialog.

Note: If you use the Tecplot GUI Builder for your interface, then the code above is done for you.

12.3. PreTranslateMessage Function for Modeless Dialogs

In order to get keyboard navigation of your modeless dialog working under Windows, you must add a PreTranslateMessage function and inform Tecplot of this function. If you do not do this, the tab key, the escape key, the return key, and other keys will not work in your dialog.

Under MFC, the PreTranslateMessage function almost always looks like this:

```
Boolean_t STDCALL PreTranslateMessage(MSG *pMsg)
```

```
{
  AFX_MANAGE_STATE(AfxGetStaticModuleState());
  return AfxGetApp()->PreTranslateMessage(pMsg);
}
```

This will handle any number of modeless dialogs in your add-on.

Under Win32 SDK, the function should look something like this:

```
HWND hwndModelessDialog1; /* a modeless dialog */
HWND hwndModelessDialog2; /* another modeless dialog */

Boolean_t STDCALL PreTranslateMessage(MSG *pMsg)
{
  Boolean_t Result = FALSE;
  if ( hwndModelessDialog1 != NULL )
    Result = IsDialogMessage(hwndModelessDialog1);
  if ( !Result && hwndModelessDialog2 != NULL )
    Result = IsDialogMessage(hwndModelessDialog2);
  return Result;
}
```

If you have more than one modeless dialog in SDK, you need to call **IsDialogMessage** on each modeless dialog as long as **IsDialogMessage** does not return **TRUE**.

The TecUtil function to inform Tecplot of your **PreTranslateMessage** function is **TecUtilInterfaceWinAddPreMsgFn**. You will usually call this function in your **InitTecAddon** function:

```
EXPORTFROMADDON void STDCALL InitTecAddOn(void)
{
  /* When using MFC, the following line is always required
      on callbacks. */
  AFX_MANAGE_STATE(AfxGetStaticModuleState( ));

  AddonID = TecUtilLockOn();

  TecUtilAddOnRegister(100,
                       "MyAddon",
                       "1.0",
                       "Amtec Engineering, Inc.");

  /* When using modeless dialogs, you must include a
      StateChange callback. */
```

```
  TecUtilStateChangeAddCallback(StateChangeCallback);

  /* And a PreTranslateMessage function. */
  TecUtilInterfaceWinAddPreMsgFn(PreTranslateMessage);

  /* Now, perform the rest of initialization. */

  TecUtilLockOff();
}
```

CHAPTER 13 *Accessing Field Data*

There are three ways to query and set field data. Each method has advantages and drawbacks with respect to speed and ease-of-use. Complete examples of methods 2 and 3 can be found in the **`showdata`** sample.

13.1. Indexing into the Data

All data access via the TecUtil layer is "1 based." For example, the first value in the data set is at offset 1.

In addition, all functions used to access data only take a single LgIndex_t value to index into the data. This section describes how to calculate this index for various zone types and variable locations.

13.1.1. Value Location

Starting with version 10 of Tecplot, add-ons must be aware of the data value location. In tecplot 10 data values can be stored at the nodes or at the cell centers. Future versions of tecplot will likely add other locations. When accessing data you must know ahead of time where the values are stored because the offset into the data will mean one thing for nodal values and another for cell centered values.

You determine the value location by calling TecUtilDataValueGetLocation.

Example:

```
ValueLoction_e ValueLocation;
ValueLocation = TecUtilDataValueGetLocation(Zone,Var);
switch (ValueLocation)
  {
    case ValueLocation_Nodal :
      {
        ....data is stored at the nodes.
        ....Access the data accordingly.
      }
    case ValueLocation_CellCentered :
      {
```

```
            ....data is stored at the cell centers.
            ....Access the data accordingly.
        }
    }
```

13.1.2. Indexing Nodal Ordered Data

For Ordered data, the access is done by treating the n-dimensional array of values in tecplot as a one dimensional array. For example, suppose you have an IJK-Ordered zone dimensioned by 10x20x30. To access the value at I=3, J=4, K=5 (one based) you would use:

IMax = 10
JMax = 20
KMax = 30
I = 3
J = 4
K = 5

Index = I + (J-1)*IMax + (K-1)*IMax*JMax
or
Index = I + IMax*((J-1) + (K-1)*JMax)

13.1.3. Indexing Nodal Finite Element Data

For finite element data there is a one-to-one correspondence between the nodal values supplied in the data file and the index you use to access these values. Thus to access the 5th nodal value for the 5th data point use an index of 5.

13.1.4. Indexing Cell Centered Ordered Data

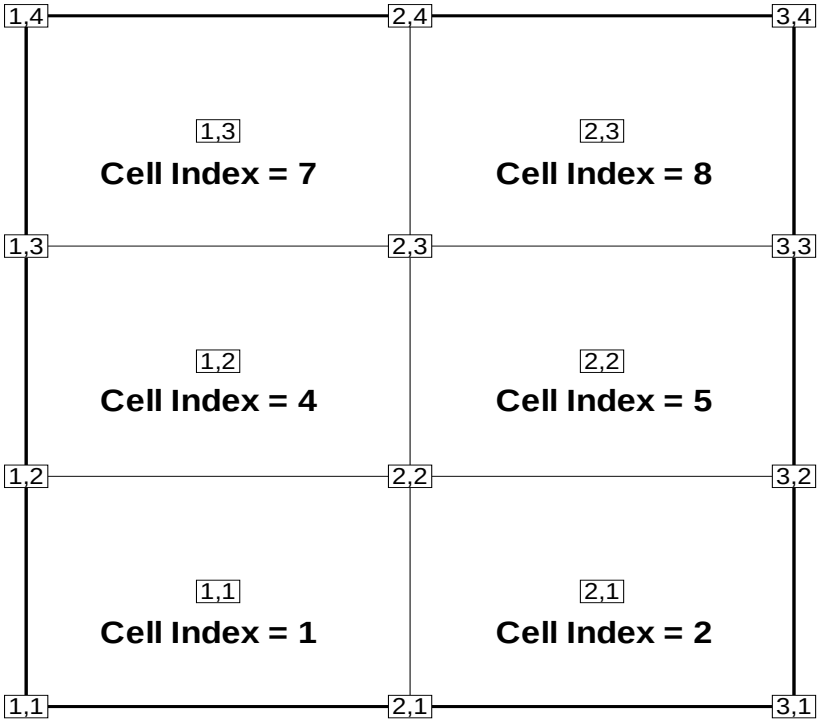
For ordered data, the index that represents the cell center is the same as the nodal index that represents the lowest indexed corner of the cell.

For example, the figure in this section shows an IJ-Ordered zone dimensioned 3x4. To access a cell centered value for the cell in the upper right had corner use the following:

IMax = 3
JMax = 4
KMax = 1
I = 2
J = 3
K = 1

Index = I + (J-1)*IMax + (K-1)*IMax*JMax
or
Index = I + IMax*((J-1) + (K-1)*JMax)

You'll notice that the equations are exactly the same as with nodal data. As a result there are gaps of unused values at IMax, JMax, and KMax that *must* be left unassigned.



13.1.5. Indexing Cell Centered Finite Element Data

For finite element data there is a one-to-one correspondence between the cell centered values supplied in the data file and index you use to access these values. For example, to access the cell centered value for the 5th cell in the connectivity list use an index of 5.

13.2. Accessing Data Using TecUtilDataValueSetByZoneVar and TecUtilDataValueGetByZoneVar

This method is the easiest to use, but it is also the slowest. if your add-on does not need to access large amounts of field data, you may find these functions to be the most convenient as there is no setup required to use these functions.

To set the first data point of the second variable of zone 5 to 3.14, you would call:

```
TecUtilDataValueSetByZoneVar(5,2,1,3.14);
```

To query this value:

```
double Value = TecUtilDataValueGetByZoneVar(5,2,1);
```

All indices are one-based. Also, you must be sure that the variable, index, and zone number parameters are valid. If not, this function will issue an error.

Note also that these functions use the **double** data type, regardless of the field data type of the zone. If the field data type is not **double**, then these functions will perform an appropriate conversion for you. If needed, you may find out the type of the field data by calling:

```
FieldDataType_e fd_type = TecUtilDataValueGetRefType(FD);
```

(References like FD in the above call will be discussed in the next section.)

13.3. Accessing Data Using TecUtilDataValueSetByRef and TecUtilDataValueGetByRef

This method requires that you set up a **FieldData_pa** pointer to reference the data, but it is much faster than method 1, while still offering error checking and parameter conversion.

Add one to the first data point of the second variable of zone 5:

```
double value;
FieldData_pa FD = TecUtilDataValueGetRef(5,2); /* Zone 5,
Variable 2 */
/* You can now use FD to get/set the data. */
value = TecUtilDataValueGetByRef(FD,1);

value += 1.0;
TecUtilDataValueSetByRef(FD,1,value);
```

```
/* No need to free FD. */
```

Note also that these functions use the **double** data type, regardless of the field data type of the zone. If the field data type is not **double**, then these functions will perform an appropriate conversion for you. If needed, you may find out the type of the field data by calling:

```
FieldDataType_e fd_type = TecUtilDataValueGetRefType(FD);
```

We strongly recommend using either this method or method 1 to access field data. The error checking that these functions perform will most likely save you development time and will help ensure that your code is bug-free.

13.4. Accessing Data using Raw Data Pointers

This method is the fastest way to get/set field data, but it must be used with extreme care. The ADK API allows you to retrieve the data pointers used internally by Tecplot to display the data. Once you have these pointers, you can get/set data values just as you would using any C or FORTRAN array pointer.

The drawback to this method is that you must be certain that you know both the data type (that is, what type of data does the pointer point to), and the number of data points (that is, the maximum index). The risks of using a raw field data pointer incorrectly include:

- De-referencing an invalid memory location.
- De-referencing one type of memory pointer as a different type of pointer.
- Not handling cell-centered or shared data connectivity.

All errors will most likely result in a crash or random behavior in Tecplot.

We recommend using this method only if maximum speed is a necessity. Usually, however, there is no discernible difference in speed between this method and method 2, thus method 2 is preferred.

Example: Add one to the first data point of the second variable of zone 5. Assume the data type is double.

```
{
  double *FieldDataArray;
  FieldDataType_e fd_type;
  TecUtilDataValueGetRawPtr(5,2,&FieldDataArray,&fd_type);
  if (fd_type == FieldDataType_Double)
```

```
    {
      double Value;
      Set_pa set = TecUtilSetAlloc(FALSE);

      FieldDataArray[0] += 1.0;

      /*
       * Inform Tecplot that we changed a variable.
       */
      TecUtilSetAddMember(set,2,FALSE);   /* Var 2 */
      TecUtilStateChanged(StateChange_VarsAltered,
                          (ArbParam_t)set);
      TecUtilSetDealloc(&set);
    }
  else
    {
      /* Handle error of unexpected type.
    }
}
```

13.5. Working with Shared Data

Tecplot allows for the sharing of data to help save on the use of physical memory. Variables and connectivity information may be shared between zones. Each variable in each zone may be shared with the same variable in any other zone.

Some operations in tecplot will automatically share variables and/or connectivity information. For example, duplicating a zone will share all variables and the connectivity information between the original zone and the newly created zone.

Likewise, some operations in tecplot will automatically force the branching of shared variables and/or connectivity information. Branching simply means that the variable or connectivity is no longer shared and every zone subsequently has its own copy. An example where tecplot automatically branches is if you executed the equation X = X + 1 exclusively on zone 1 where prior to the operation zone 1 and zone 2 shared X.

Add-ons registering themselves as "V10 Aware" (i.e., they used the new TecUtilAddOnRegister function) must be aware and know how to handle shared data.

13.5.1. TecUtil functions.

The following functions are provided to work with shared data:

TecUtilDataConnectBranchShared	Branch connectivity information that was previously shared between two zones.
TecUtilDataConnectShare	Share connectivity information between two zones.
TecUtilDataValueBranchShared	Branch a variable that was previously shared between two zones.
TecUtilDataValueShare	Share a variable between two zones.
TecUtilDataConnectGetShareCount	Get the number of zones sharing connectivity information
TecUtilDataValueGetShareCount	Get the number of zones sharing a variable.

Figure 13-1.

Note that there are essentially two types of functions - ones that deal with variables and ones that deal with connectivity information.

13.5.2. Allowing Data Sharing

The function TecUtilDataSetIsSharingAllowed should be called prior to using any of the TecUtil sharing functions if your add-on has any reason to believe that customers may shut down sharing. This should be an isolated occurrence and most likely limited to cases where an older add-on must be used that cannot handle shared data and the user is forced to tell tecplot to shut down sharing.

To shut down data sharing add the following entry to the tecplot.cfg file:

$!Compatibility AllowDataSharing = No

13.5.3. Querying or Modifying shared data

Variable and connectivity sharing information in Tecplot may be queried or altered using the standard methods described earlier in this chapter. If the variable or connectivity information is shared, any modifications will be realized by all zones that share the information.

State changes can be sent to Tecplot identifying the variable (or connectivity information) that was changed. Only those zones that were altered need to be identified and Tecplot will recognize that other zones may be affected because of sharing.

13.5.4. Sharing

A variable or connectivity information can be shared between zones. When you do this, the memory allocated for the variable in the destination zone is freed and the variable in the destination zone will point to the memory used by the variable in the source zone. The share count is then incremented by 1.

13.5.5. Branching

It may be the case that you wish to modify a variable in a zone and also require that these modifications occur exclusively in that zone. If this is the case, and you have no previous knowledge of sharing of this variable then it is best to branch the variable prior to modification. If previously shared, branching will allocate memory for the variable (or connectivity information) and make a copy of all the values. The share count for the original data is decremented by 1. Branching an already branched variable does nothing.

Example: Do some data operation to variable 3 exclusively in zone 7.

```
if (TecUtilDataValueBranchShared(7,  /* Zone */
                                 3)) /* Var */
  {
    .... do data operation on variable 3 in zone 7.
  }
```

CHAPTER 14 *Handling Tecplot State Changes from an Add-on*

State changes are Tecplot's method for propagating information when an event occurs. The basic sequence of events is:

1. An action is taken,
2. A state change message is sent to Tecplot, and
3. Tecplot transmits the state change message to any add-ons which have registered state change callbacks.

Some examples of actions which cause state changes are loading a data file, changing the color of a mesh plot, creating a new zone, and changing the frame mode.

There are two main issues in handling Tecplot state changes from an add-on. The first is how to listen for Tecplot state change messages from your add-on. The second is how and when to send state change messages to Tecplot from your add-on.

For details on any TecUtil function mentioned in this chapter, please see the *ADK Online Reference*.

In general, your add-on should listen for state changes if it needs to take some action based on the state of Tecplot. For example, if your add-on displays a dialog which deals with 3-D plots, you may want to drop the dialog if the frame mode is changed to XY. Or, you may need to update the dialog if a new data set is loaded.

Most state changes are generated automatically for you by Tecplot. Only under certain circumstances will your add-on be required to explicitly "Send" a state change notification. See 14.3, “Sending State Changes,” for details on when your add-on needs to explicitly send a state change.

14.1. State Change Values

Table 14-1, “State change values,” on page 102 shows the available state change values. Column 1 shows the state change value constants that appear in **`GLOBAL.h`**. New versions of tecplot may add new state changes so you may want to refer to **`GLOBAL.h`** in case this documentation is out of date.

Table 14-1. State change values

State Change Value	Explanation	Example of when this occurs using the user interface	Example of when this occurs using TecUtil functions
`StateChange_VarsAdded`	One or more variables were added.	Adding a new variable in the Data/Alter/Specify Equations dialog.	`TecUtilDataSetAddVar`
`StateChange_ZonesAdded`	One or more zones were created.	Data/Create Zone/Circular dialog.	`TecUtilCreateRectangularZone`
`StateChange_ZonesDeleted`	One or more zones were deleted.	Data/Delete Zones dialog.	`TecUtilZoneDelete`
`StateChange_VarsAltered`	Values of one or more variables were altered.	Alter variable values in the Data/Alter/Specify Equations dialog.	`TecUtilDataAlter`
`StateChange_NodeMapsAltered`	The node map for one or more zones was altered.	Cannot do this from the interface.	`TecUtilDataNodeSetByZone`
`StateChange_DataSetReset`	A new data set has been loaded.	File/Load DataFile dialog.	`TecUtilReadDataSet`
`StateChange_DataSetFileName`	The current data set has been saved to a file.	File/Write DataFile dialog.	`TecUtilWriteDataSet`
`StateChange_DataSetTitle`	The current data set title has been changed.	Change data set title in Data/Data Set Info dialog.	`TecUtilDataSetSetTitle`
`StateChange_NewLayout`	The current layout has been cleared and reset.	File/New Layout menu.	`TecUtilNewLayout`
`StateChange_NewTopFrame`	A new frame has become the current frame.	Frame/Order Frames dialog.	`TecUtilFrameCreateNew`
`StateChange_FrameDeleted`	A frame was deleted.	Frame/Delete Current Frame menu.	`TecUtilFrameDeleteTop`
`StateChange_Style`	The style of the plot has been altered.	Zone Style/Mapping Style dialogs.	`TecUtilZoneSetMesh TecUtilStyleSetLowLevel`
`StateChange_Text`	One or more text elements have changed.	Adding, removing, or modifying text.	`Explicitly by calling TecUtilStateChanged with StateChange_Text.`

Table 14-1. State change values

State Change Value	Explanation	Example of when this occurs using the user interface	Example of when this occurs using TecUtil functions
`StateChange_Geom`	One or more geometry elements have changed.	Adding, removing, or modifying geometries.	`Explicitly by calling TecUtilStateChanged with StateChanged_Geom.`
`StateChange_LineMapAssignment`	An X-Y mapping definition has been altered (includes zone and axis information).	Definitions page of the Mapping Style dialog.	`TecUtilLineMapSetAssignment`
`StateChange_ContourLevels`	The contour levels have been altered.	Field/Contour Levels.	`TecUtilContourLevelAdd`
`StateChange_ZoneName`	The name of a zone has been altered.	Rename a zone in the Data/Data Set Info dialog.	`TecUtilZoneRename`
`StateChange_VarName`	The name of a variable has been altered.	Rename a variable in the Data/Data Set Info dialog.	`TecUtilVarRename`
`StateChange_LineMapName`	The name of an line mapping has been altered.	Rename a line map in an XY Line plot.	`TecUtilLineMapSetName`
`StateChange_LineMapAddDeleteOrReorder`	The set of existing line mappings has been altered.	Mapping Style dialog, Create Map button.	`TecUtilLineMapDelete`
`StateChange_View`	The view of the plot has been altered (usually a translate, scale, or fit action).	View/Translate-Magnify dialog.	`TecUtilViewTranslate`
`StateChange_ColorMap`	The color mapping has been altered.	Workspace/ColorMap dialog.	`TecUtilColorMapResetToFactory`
`StateChange_ContourVar`	The contour variable has been reassigned.	Field/Contour Variable dialog.	`TecUtilContourSetVariable`
`StateChange_Streamtrace`	The set of streamtraces, a termination line, or the streamtrace delta time has been altered.	Field/Streamtrace Placement dialog.	`TecUtilStreamtraceAdd`
`StateChange_NewAxisVariables`	The axis variables have been reassigned.	Plot/Assign XYZ dialog.	`TecUtilStyleSetLowLevel`

Table 14-1. State change values

State Change Value	Explanation	Example of when this occurs using the user interface	Example of when this occurs using TecUtil functions
`StateChange_MouseModeUpdate`	A new mouse mode (tool) has been selected.	Select a new mouse mode (tool) in the sidebar.	`TecUtilPickSetMouseMode`
`StateChange_PickListCleared`	All picked objects are unpicked.[a]	Click on the paper in the workspace.	`TecUtilPickDeselectAll`
`StateChange_PickListGroupSelect`	A group of objects has been added to the pick list.[a.]	Draw a box around a group of objects with the selector or adjuster tool.	`TecUtilPickAddAll`
`StateChange_PickListSingleSelect`	A single object has been added to or removed from the pick list.[a.]	Select an object with the selector or adjuster tool.	`TecUtilPickAtPosition`
`StateChange_PickListStyle`	An action has been performed on all of the objects in the pick list.[a.]	Quick Edit dialog.	`TecUtilPickEdit`
`StateChange_ModalDialogLaunch`	A modal dialog has been launched (see Chapter 12, “Modal and Modeless Dialogs in Windows”).	Workspace/Ruler-Grid dialog.	`TecUtilDialogMessageBox`
`StateChange_ModalDialogDismiss`	A modal dialog has been dismissed (see Chapter 12, “Modal and Modeless Dialogs in Windows”).	Workspace/Ruler-Grid dialog.	`TecUtilDialogMessageBox`
`StateChange_CompleteReset`	Anything could have happened (see Section 14.2.1, “State Change Modes,” below).	File/Open Layout dialog.	`TecUtilOpenLayout`
`StateChange_DrawGraphicsOn`	Graphics have been turned back on.	Execute the macro command `$!DRAWGRAPHICS ON`.	`TecUtilDrawGraphics(TRUE);`
`StateChange_DrawGraphicsOff`	Graphics have been turned back off.	Execute the macro command `$!DRAWGRAPHICS OFF`.	`TecUtilDrawGraphics(FALSE);`

Table 14-1. State change values

State Change Value	Explanation	Example of when this occurs using the user interface	Example of when this occurs using TecUtil functions
`StateChange_DataSetLockOn`	The dataset attached to the active frame in tecplot has been locked.	Cannot be done via user interface.	`TecUtilDataSetLockOn`
`StateChange_DataSetLockOff`	The dataset attached to the active frame in tecplot has been unlocked.	Cannot be done via user interface.	`TecUtilDataSetLockOff`
`StateChange_DrawingInterrupted`	The user has interrupted the drawing.	User clicks with the mouse in the workarea..	`TecUtilInterrupt().`
`StateChange_QuitTecplot`	Tecplot is about to exit.	File/Exit menu.	`TecUtilQuit`
`StateChange_AuxDataAdded`	Auxiliary data was added.	Cannot do this via the user interface	`TecUtilAuxDataSetItem`
`StateChange_AuxDataDeleted`	Auxiliary data was deleted	Cannot do this via the user interface.	`TecUtilAuxDataDeleteItemByName`
`StateChange_AuxDataAltered`	Auxiliary data was altered.	Cannot do this via the user interface.	`TecUtilAuxDataSetItem`

a. See Chapter 18, “Working With Picked Objects,” for more information on picked objects and the pick list.

14.2. Listening for State Changes

In order to listen for Tecplot state changes, your add-on must register a callback function which Tecplot will call whenever a state change occurs.

State change add-on callback functions are of the type **`StateChangeAddOnCallbackV2_pf`** and look like:

```
void StateChangeCallback (StateChange_e StateChange);
```

where *StateChange* is one of the types listed above in Table 14-1, “State change values,” on page 102. When called, your callback can then turnaround and query Tecplot for supplemental information using one or more of the following functions:

TecUtilStateChangeGetArbEnum
TecUtilStateChangeGetIndex
TecUtilStateChangeGetStyleParam
TecUtilStateChangeGetVarSet

TecUtilStateChangeGetZone
TecUtilStateChangeGetZoneSet

The state changes that have supplemental information that can be queried are:

Table 14-2. State changes and supplemental infromation.

State Change	Supplemental Information
`StateChange_VarsAltered`	VarSet, ZoneSet (optional),Index (optional)
`StateChange_VarsAdded`	VarSet
`StateChange_ZoneDeleted`	ZoneSet
`StateChange_NodeMapAltered`	ZoneSet
`StateChange_MouseModeUpdate`	Enum for the new mouse mode (MouseButtonMode_e)
`StateChange_Style`	Style parameters P1, P2, P3, P4, P5, P6 (P2-P6 are optional)
`StateChange_View`	Enum for view action (View_e)
`StateChange_Streamtrace`	Enum for Streamtrace action (Streamtrace_e)
`StateChange_AuxDataAltered`	Enum for Auxiliary Location (AuxDataLocation_e),Zone if location is AuxDataLocation_Zone
`StateChange_AuxDataAdded`	Enum for Auxiliary Location (AuxDataLocation_e),Zone if location is AuxDataLocation_Zone
`StateChange_AuxDataDeleted`	Enum for Auxiliary Location (AuxDataLocation_e),Zone if location is AuxDataLocation_Zone

Note that not all supplemental information may be available all the time. It is up to the supplier of the state change to supply the supplemental information and in some cases (like older add-ons calling TecUtilStateChange) it may not be supplied. If information is not supplied then you must assume the worst case. For example, on a StateChange_VarsAltered, if the ZoneSet is not supplied you must assume that the variables were altered in all zones. For the vars altered state change, you can assum that the VarSet is supplied.

As an example, the following code registers a state change callback called **`StateChangeCallback`** which monitors contour level and contour variable changes and updates a dialog when those changes occur.

```
void StateChangeCallback (StateChange_e StateChange)
{
```

```
  TecUtilLockStart(AddonID);
  if ((StateChange == StateChange_ContourLevels) ||
      (StateChange == StateChange_ContourVar)    ||
      (StateChange == StateChange_CompleteReset))
    UpdateMyContourDialog();
  TecUtilLockFinish(AddonID);
}

void LaunchMyContourDialog (void)

  ArgList_pa ArgList;
  TecUtilLockStart(AddonID);
  ArgList = TecUtilArgListAlloc();
  TecUtilArgListAppendFunction(ArgList,
                               SV_CALLBACKFUNCTION,
                               (Void *)StateChangeCallback);
  TecUtilArgListAppendInt(SV_STATECHANGEMODE,
                          StateChangeMode_v100);
  TecUtilStateChangeAddCalbackX(ArgList);
  TecUtilArgListDealloc(&ArgList);
  TecUtilLockFinish(AddonID);
}

void InitTecAddOn (void)
{
  TecUtilLockOn();
  AddonID = TecUtilAddOnRegisterInfo(
      "Contour Plot Enhancer", "1.0",
      "My Company");
  TecUtilMenuAddOption("Tools", "Contour Dialog",
                       'C', LaunchMyContourDialog);
  TecUtilLockOff();
}
```

Only certain state changes will probably be of interest to your add-on. For example, the above add-on only monitors state changes which involve the contour variable or the contour levels. **StateChange_CompleteReset** must be listened for as well, since anything could have happened to cause it. See Section 14.2.1, "State Change Modes," for an in depth discussion of when to listen for the **StateChange_CompleteReset** state change.).

For an example where supplemental infromation is used, look at the case where the above add-on instead wants to update its dialog when the contour variable value was altered:

```
void StateChangeCallback (StateChange_e StateChange)
{
  TecUtilLockStart(AddonID);
  if (StateChange == StateChange_VarsAltered
    {
      EntIndex_t ContourVarNum =
               TecUtilVarGetNumByAssignment('C');
      Set_pa VarSet = NULL;
      TecUtilStateChangeGetVarSet(&VarSet);
      if (TecUtilSetIsMember(VarSet,ContourVarNum
      UpdateMyContourDialog();
    }
  else if (StateChange == StateChange_CompleteReset)
    UpdateMyContourDialog();
  TecUtilLockFinish(AddonID);

}
```

Do *not* deallocate VarSet after use, as Tecplot just hands off a reference to the internal value.

14.2.1. State Change Modes

The original implementation of the state change mechanism behaved such that when a lot of processing was taking place Tecplot would turn the graphics off and not bother propagating the state changes until the graphics were turned back on again after processing, at which point Tecplot would simply broadcast **`StateChange_CompleteReset`** notification.

Particularly, this happened with loading stylesheets and layouts The advantage to this approach was that add-ons were not bombarded with notifications where a simple reset notification would suffice.

The limitation to this approach occured with the introduction of state changes in add-ons that did more than just inform add-ons of changes intended to keep consistancy with modeless dialogs. For instance, an add-on may want to know all the details of just what has transpired during the execution of a macro so it can later act appropriately.

This being the case, a state change mode is available as an option when registering your callback. In the example code on page 107, the following line:

```
TecUtilArgListAppendInt(SV_STATECHANGEMODE,
StateChangeMode_v100);
```

sets the state change mode to V100. **StateChangeMode_v100** is the default.

Setting the mode to **StateChangeMode_v100** has the following effects:

- All state changes occuring while the graphics are suspended are propagated to your call-back.
- **StateChange_DrawGraphicsOff** is sent to the add-on when graphics are shut off in Tecplot and **StateChange_DrawGraphicsOn** is sent when graphics are turned on.
- **StateChange_CompleteReset** is not sent to your callback.

Setting the mode to **StateChangeMode_v75** has the following effects:

- No state changes are propagated to your callback when the graphics are off.
- You receive a **StateChange_CompleteReset** when the graphics are turned back on.
- You never receive **StateChange_DrawGraphicsOff** or **StateChange_DrawGraphicsOn**.

14.2.2. Coding Rules for State Change Callbacks

When a call is made to your state change callback function it is almost always the case that either Tecplot or some other add-on is in the middle of a sequence of tasks. Thus, it is highly advisable that the code in your callback function itself refrain from actions that themselves generate state changes. All **TecUtil** functions that query state information from Tecplot are acceptable, as those requests do not cause Tecplot's state to change. Executing functions like **TecUtilCreateRectangularZone** is not recommended since they modify Tecplot's state.

If you are unsure if the code in your add-on is generating state change callbacks, you can make use of the *statechg* add-on provided by Amtec. Simply uncomment the entry for this add-on in the main **Tecplot.add** file. The *statechg* add-on monitors all state changes and displays them in a dialog.

If you have a case where you would like to make **TecUtil** function calls that do change the state in Tecplot, then the best approach is to set a local flag for your add-on so that the desired operation can be performed later when other GUI callback events are processed. By registering an on-idle callback, Tecplot will notify the add-on when Tecplot returns to an idle state. It is important for an add-on to set a local flag when it registers the callback (and check that the flag is not set before registering) and clears the flag after the registered callback is called by Tecplot so that only one on-idle callback is registered to handle the pending operations since Tecplot

may issue many "state changed" calls before returning to an idle state. Please refer to the **TecUtilOnIdleQueueAddCallback** definition in the ADK Online Reference for an example of registering such a callback.

An example of an unacceptable reaction to a pick type state change event would be to delete the object. It is more acceptable to have an interface button that when pressed would delete all picked objects (for example, objects of a particular type). You would then use the pick type state change to update the sensitivity of the button instead of immediately deleting the object.

14.3. Sending State Changes

There are only certain circumstances under which your add-on will need to send state changes to Tecplot. Most of the TecUtil functions will transmit the necessary state changes automatically. For example, when an add-on calls **TecUtilZoneDelete**, Tecplot automatically transmits the **StateChange_ZonesDeleted** state change.

Currently, your add-on must send state changes to Tecplot under the following circumstances:

Table 14-3. State Changes Add-Ons are Allowed to Send

Circumstance	Relevant state change value	Supplemental Infromation supplied to Tecplot
Launch and dismissal of modal dialogs (Windows only).	**StateChange_ModalDialogLaunch**, **StateChange_ModalDialogDismiss**	None.
After a variable has been added and subsequently modified.	**StateChange_VarsAdded**	None.
After a variable has been modified.	**StateChange_VarsAltered**	Set of affected variables.* Index of value changed. Set of affected zones.
After the node map has been modified.	**StateChange_NodeMapsAltered**	Set of affected zones.*
After **TecUtilDataSetAddZone** has been called and the field data set has been subsequently modified.	**StateChange_ZonesAdded**	Set of affected zones.*

Table 14-3. State Changes Add-Ons are Allowed to Send

Circumstance	Relevant state change value	Supplemental Infromation supplied to Tec-plot
After adding, removing, or modifying one or more text elements.	`StateChange_Text`	None.
After adding, removing or modifying one or more geometry elements.	`StateChange_Geom`	None.

*This information must be supplied via the calldata parameter if using TecUtilStateChanged and must be supplied as an arglist member if using TecUtilStateChangedX.

To send state changes to Tecplot, you may use the older **`TecUtilStateChanged`** function or the newer **`TecUtilStateChangedX`** function.

14.3.1. Using TecUtilStateChanged

The prototype for TecUtilStateChanged is:

```
void TecUtilStateChanged (StateChange_e StateChange,
                          ArbParam_t    CallData);
```

where *StateChange* is one of the types listed above in table 10-3, and *CallData* is extra information that is sent only when necessary. For state changes which use the *CallData* parameter, the extra information is required. For example, when sending a **`StateChange_VarsAltered`**, you must send the set of altered variables as the *CallData* parameter. See the class of **`TecUtilSetXXX`** functions for creating, manipulating and destroying sets in the *ADK Online Reference*.

14.3.2. Using TecUtilStateChangedX

Since the State Change mechanism in Tecplot V10 has been expanded to allow the transfer of supplemental information, it is necessary that there be a way to supply this information when broadcasting a state change. The older TecUtilStateChanged function is likely sufficient at present for most cases, however future versions of tecplot will allow the transfer of more supplemental information. In V10,the state change StateChange_VarsAltered is the one case where add-ons may supply more than one piece of information.

TecUtilStateChangedX has the following prototype:

```
void TecUtilStateChangedX(ArgList_pa ArgList);
```

Where Arglist is constructed like any other "X" function. The ArgList may contain one or more of the following:

Table 14-4. ArgList values

Argument Name	Type
SV_STATECHANGE	StateChange_e
SV_ZONELIST	Set_pa
SV_VARLIST	Set_pa
SV_INDEX	Integer

When an addon submits a state change with StateChange_VarsAltered you may supply not only the set of variables altered (required) but also the set of zones in which those variables were altered and, if only one value was altered, the index of the point that was altered.

Example:

Your addon alters the 3rd variable in zones 5 and 6 at offset 10. You are through altering data so you must broadcast a state change:

Note: Error checking omitted for clarity.

```
{
 ArgList_pa ArgList;
 Set_pa     ZoneList;
 Set_pa     VarList;

 ArgList  = TecUtilArgListAlloc();
 ZoneList = TecUtilSetAlloc(FALSE);
 VarList  = TecUtilSetAlloc(FALSE);

 TecUtilSetAddMember(ZoneList,5,FALSE);
 TecUtilSetAddMember(ZoneList,6,FALSE);
 TecUtilSetAddMember(VarList,3,FALSE);
 TecUtilArgListAppendInt(ArgList,SV_STATECHANGE,
                         (LgIndex_t)StateChange_VarsAltered);
 TecUtilArgListAppendSet(ArgList,SV_ZONELIST,ZoneList);
 TecUtilArgListAppendSet(ArgList,SV_VARLIST,VarList);
 TecUtilArgListAppendInt(ArgList,SV_INDEX,10);
```

```
    TecUtilStateChangedX(ArgList);
    TecUtilSetDealloc(&ZoneList);
    TecUtilSetDealloc(&VarList);
    TecUtilArgListDealloc(&ArgList);
}
```

As discussed in the previous sections, the state change listeners can choose to use any/all of the supplemental information they desire. Listeners must also know how to handle situations where the supplemental information is not supplied (assume worst case).

To see examples of the use of **`TecUtilStateChanged`** or **`TecUtilStateChangedX`**, please refer to the *ADK Online Reference* and refer to the relevant **`TecUtil`** functions which are listed above. Note that no other state change notifications may explicitly originate from your add-on besides the ones listed in Table 10-3.

CHAPTER 15 *Augmenting Tecplot's Macro Language*

Tecplot has a macro language you can use to automate tasks that are performed repeatedly. Macro functions can be assigned to buttons in the Quick Macro Panel, or macros can be created to load and process data retrieved from a large number of files. Your add-on can also be designed to augment Tecplot's macro language so tasks performed by your add-on can be automated as well.

For details on the TecUtil functions discussed in this chapter, please see the *ADK Online Reference*.

15.1. Processing Custom Macro Commands

You can augment Tecplot's macro language with your own set of commands that will be routed directly to your add-on for processing. The Tecplot macro command **$!ADDONCOMMAND** is the bridge that extends Tecplot's macro language. The **$!ADDONCOMMAND** macro command looks like:

```
$!ADDONCOMMAND
  ADDONID = string
  COMMAND = string
```

Here the string assigned to **ADDONID** identifies which add-on is to receive the command, and the **COMMAND** parameter identifies the commands to be processed by the add-on.

To tell Tecplot what function to call when it encounters your macro command, you make a call to **TecUtilMacroAddCommandCallback** from the **InitTecAddOn** function in your add-on. The call to **TecUtilMacroAddCommandCallback** is defined as follows:

```
TecUtilMacroAddCommandCallback(MyAddOnID,MyMacroProcessor);
```

Where *MyAddOnID* is a string used in the **ADDONID** part of the **$!ADDONCOMMAND** and *MyMacroProcessor* is the name of a function you write to handle the macro command.

Example:

Suppose you have an add-on that can sum the areas (or volumes) of all cells in a specified list of zones. You want to create a macro command that can tell your add-on which zones to process.

The first task is to create a function in your add-on that can handle these instructions. This function will look something like:

```
Boolean_t STDCALL ProcessSumCellsCommand(char *CommandString,
                                         char **ErrMsg)
{
  Boolean_t IsOk = TRUE;
   /*
    * Process commands in CommandString
    */
   return (IsOk);
}
```

The next task is to add the following line to the **InitTecAddOn** function in your add-on:

```
   ...
  TecUtilMacroAddCommandCallback("SUMCELLS",
                                 ProcessSumCellsCommand);
  ...
```

This tells Tecplot to watch out for macro commands that look like

```
$!ADDONCOMMAND
  ADDONID = "SUMCELLS"
  COMMAND = "1,2,5-9"
```

When Tecplot processes a command like the one above, it turns around and calls the function you registered with the second parameter to **TecUtilMacroAddCommandCallback**, which in this case is **ProcessSumCellsCommand**. **ProcessSumCellsCommand** is called with the command which is the string taken from the **COMMAND** parameter in the macro. In the example above, **ProcessSumCellsCommands** would be called with the string **"1,2,5-9"** as its first parameter.

You may design any syntax you wish for the instructions sent to your add-on. The only restriction is that they must be able to fit into a single string in a Tecplot macro sub-command. In the previous example, the function **ProcessSumCellsCommand** will be coded so that it can scan a comma- and dash-delimited set of numbers, and determines a set of zones for which to sum the areas or volumes.

15.2. Error Processing

If your macro command callback function detects an error in the command string, or during processing, it is required to do the following:

1. Allocate enough space for an error message by calling **TecUtilStringAlloc**. The *ErrMsg* parameter to the callback function must be assigned to this space.
2. Generate an appropriate error message and place it into the space created in Step 1.
3. Return with a value of **FALSE**.

For example, suppose the function **ProcessSumCellsCommand** in the previous section detects that a zone specified in the command does not exist (and this is determined to be an error condition). The coding for `ProcessSumCellsCommand` may then look like:

```
Boolean_t STDCALL ProcessSumCellsCommand(char  *CommandString,
                                         char **ErrMsg)
{
  Boolean_t IsOk = TRUE;
  EntIndex_t CurZone;
  char     *CPtr = CommandString;
  double    SumTotal = 0;

  /*
   *  Scan CommandString and pull out zones to
   *  sum. The function GetNextZone pulls out the
   *  next zone number and advances CPtr. SumNextZone
   *  attempts to calculate a sum in the next zone and
   *  returns FALSE if the zone requested is invalid.
   */

   while (IsOk && GetNextZone(&CPtr,&CurZone))
     {
       double CurSum;
       if (SumNextZone(CurZone,&CurSum))
         SumTotal += CurSum;
       else
         {
           IsOk = FALSE;
           *ErrMsg = TecUtilStringAlloc(200);
           sprintf(*ErrMsg,"Can't calc sum for zone %d",CurZone);
         }
     }
   return (IsOk);
}
```

The functions **GetNextZone** and **SumNextZone** are not provided for reasons of space.

15.3. Recording Custom Macro Commands

The user may choose to have Tecplot record a particular session. If the user is recording a macro and uses your add-on, you will want the action to be translated into a macro command written to the macro record file.

To record a macro command, simply call the function **TecUtilMacroRecordAddOnCommand** after your add-on has successfully performed an operation requested by the user.

For example, the user, via dialogs in your add-on, requests to sum the areas of cells in zones 1 through 5. After the add-on performs a successful operation it makes the following call:

```
if (TecUtilMacroIsRecordingActive())
 TecUtilMacroRecordAddOnCommand("SUMCELLS","1-5");
```

This will write out the following text to the macro file:

```
$!ADDONCOMMAND
  ADDONID = "SUMCELLS"
  COMMAND = "1-5"
```

TecUtilMacroRecordAddOnCommand requires that macro recording is active. This means that you must call **TecUtilMacroIsRecordingActive** and check the return value before calling **TecUtilMacroRecordAddOnCommand**.

CHAPTER 16 *Implementing Data Journaling*

Tecplot version 10 introduced data journaling. The initial loading of a dataset and any changes made to a dataset after loading are actions that can be journaled. Executing the data journal recreates the data by loading data files and performing modifications. When a layout is saved, rather than saving a new data file, the Tecplot can reference the original data file and store only the modifications to the data within the layout file. This eliminates duplication of data and saves disk space. Tecplot allows add-ons that modify data to record their actions to the journal. The actions are recorded as **$!ADDONCOMMAND** macros. When a layout file is loaded, these macros are processed the same way other add-on macros are processed--by calling the add-on's macro command callback routine. For more details on add-on macros, refer to Chapter 15, "Augmenting Tecplot's Macro Language," on page 115. Data set reader add-ons do not need to journal their actions--Tecplot journals their actions for them. For details on the TecUtil functions discussed in this chapter, please see the *ADK Online Reference*.

16.1. Data Journaling Prerequisites

In order to journal a data modification your add-on performs, the following conditions must be met:

- There must be a data set attached to the current frame.
- The data set must have a valid journal.
- The data modification must involve only one data set.
- The data modification must not rely on style settings.

The first two prerequisites are easily verified with ADK functions calls, as will be shown below. The third prerequisite arises from the fact that frames and their data sets can appear in layout files in any order. You can't be certain that one data set will be created prior to another data set, so data journaling cannot rely on the order in which data sets are created. If your add-on creates or modifies one data set based on values in a different data set, this criterion is violated, and the operation cannot be journaled.

The final prerequisite arises from the fact that data journals in layout files are executed prior to any style commands. Style settings include axis, contour and other variable assignments. So if your add-on performs some operation that uses the identity of the axis variables, then you need to include the axis variable assignments in your journal command, because your add-on will not be able to query Tecplot for that information when the journal is executed.

16.2. Determining Whether a Layout is Being Processed

Depending on how you design your add-on's macro commands, your add-on may need to know whether its macro command callback is being called as a result of playing a macro file or processing the journal in a layout file (style settings are available when macro files are played). The function **TecUtilStateIsProcessingJournal** provides this information. A typical add-on macro command callback might use the following logic to determine what data set variable to use for the X axis variable:

1. Determine whether a layout journal is being processed by calling **TecUtil-StateIsProcessingJournal**.
2. If a data journal is being processed, use the X variable number stored in the macro command string. Otherwise, get the variable number of the X axis variable by calling **TecUtilVarGetNumByAssignment('X')**.

16.3. Inhibiting Marking of the Data Set

Unless the add-on inhibits it, any data modification it performs will "mark" the data set. Marking the data set invalidates the data journal and indicates to Tecplot that the data set must be saved to a new data file when a layout is saved. To journal its actions, therefore, an add-on must inhibit data set marking prior to performing its data modifications. Upon completion of its modifications, the add-on must re-enable marking. The **TecUtilDataSetSuspendMarking** ADK function is used for both of these purposes.

16.4. Macro Recording vs. Data Journaling

There are two ways data modifications can be "remembered" in Tecplot. One is using the Data Journal whereby the instructions to modify data are journaled in Tecplot and then saved to layout files. The other is via macro recording.

Although not a requirement, it is often very useful if an add-on has the ability to supply macro record instructions to Tecplot whenever the add-on performs an action changing the state of Tecplot (See the Chapter "Augmenting Tecplot's Macro Language). This way a user can record a macro and play it back to exactly reproduce a set of actions. Thus to satisfy both data journaling and macro recording your code should look like:

```
  TecUtilDataSetSuspendMarking(TRUE);

  //  Do actions that change the state in Tecplot
  //  Make sure to call TecUtilStateChanged if you changed
data.
```

```
  TecUtilDataSetSuspendMarking(FALSE);


  //  Create a command that your addon can parse later when
  //  requested.  This command represents the work done while
the
  //  data was suspended in the above code.
  //  In the following code the variable "MacroCommand" holds
the
  //  macro command for this operation.


  if (TecUtilDataSetJournalIsValid())
   TecUtilDataSetAddJournalCommand(ADDON_NAME, MacroCommand,
NULL);
  if(TecUtilMacroIsRecordingActive())
    TecUtilMacroRecordAddOnCommand(ADDON_NAME, MacroCommand);
```

16.5. Example Data Journaling Code

The below code verifies there is a current data set. If so, it suspends data set marking and performs some action that modifies the data set. It then verifies that the data journal is valid, and records an **$!ADDONCOMMAND** macro command to the data journal. Finally, it reactivates data set marking. Note that it stores the identity of the X axis variable, which the add-on will need to use when the journal is executed:

```
if (TecUtilDataSetIsAvailable())
  {
    EntIndex_t XAxisVar
    char Command[256];
    /* Suspend data set marking. */
    TecUtilDataSetSuspendMarking(TRUE);

    /* Modify the data set. */
    ModifyDataSet();

    XAxisVar = TecUtilVarGetNumByAssignment(X);
    sprintf(Command,
            "DoStuff XAxisVar=%d",
            (int)XAxisVar);

    if (TecUtilDataSetJournalIsValid())
      {
        TecUtilDataSetAddJournalCommand(ADDON_NAME,
                                        Command,
```

```
                                          NULL); /* RawData */
    }
  if (TecUtilMacroIsRecordingActive())
    {
      TecUtilMacroRecordAddOnCommand(ADDON_NAME,
                                     Command);
    }
}
```

CHAPTER 17 *Adding Online Help to Your Add-on*

TGB provides an easy way for you to add on-line help to your add-on. Each modal and modeless dialog created with TGB has a 'Help' button and associated callback. You are free to do any processing or use any help system you wish in the callback. The Tecplot ADK API has also provided a function, **`TecUtilHelp()`** which can display any HTML file when the user presses the help button. This chapter will explain how to use **`TecUtilHelp()`** to add on-line help to your add-on.

17.1. Step 1: Write your Help Pages

Write your help system as one or more HTML files. For simple add-ons, this may be just a single HTML file. For add-ons that require more description, you may also use multiple files or an index page which references multiple files.

17.2. Step 2: Create a Help Directory

When you install your add-on, create a directory under the <*TEC100HOME*>**`/help`** directory. For example, if the name of your add-on is *MyAddOn*, create a directory: <*TEC100HOME*>**`/help/MyAddOn/`**. The name of the directory is not required to be the same as the name of your add-on.

17.3. Step 3: Processing the Help Button Callback

The TGB will generate a help button callback in **`guicb.c`** as follows:

```
static void Dialog1HelpButton_CB(void)
{
  TecUtilLockStart(AddonID);
  TecUtilDialogMessageBox("No help available");
  TecUtilLockFinish(AddonID);
}


Change this to be:

static void Dialog1HelpButton_CB(void)
{
  TecUtilLockStart(AddonID);
  TecUtilHelp("MyAddOn/file.htm",FALSE,0);
```

```
  TecUtilLockFinish(AddonID);
}
```

Where **MyAddOn/file.htm** is the path of the HTML file to be displayed. Typically this will be named **index.htm**, but this is not required. Note that by default, Tecplot will prefix *<TEC100HOME>***/help/** to the file name, so you only specify the path below the Tecplot Help directory. Alternatively, you can always use an absolute path to your HTML file, like **C:\MyDir\index.htm**.

We highly recommend, however, that you install your help files in Tecplot's help directory.

17.4. Using the TecUtilHelp Function

When **TecUtilHelp** is called with an HTML (either a local HTML file or a valid URL), the HTML file is displayed using the default browser installed on the system.

In Windows, the default browser is located for you automatically using the system registry; there is no additional setup needed.

On UNIX systems, you must specify the command which runs your browser by placing the following line in the Tecplot configuration file (**tecplot.cfg**):

```
$!Interface UnixBrowserLaunchCmd = <string>
```

where *<string>* is the command you would type at the command prompt to launch the browser. You must use the **@** symbol to mark the location where the URL or filename should be placed, as shown in the following example:

```
$!Interface UnixBrowserLaunchCmd = "\usr\bin\netscape @"
```

You may use any valid HTTP syntax for the HTML file name, as long as it is accepted by the selected browser at the command line. For example, you may use special characters such as the octothorp (**#**) symbol to specify a particular location in the file.

For complete details on using **TecUtilHelp**, refer to the *ADK Online Reference*.

CHAPTER 18 *Working With Picked Objects*

An *object* in Tecplot is any item which appears in the workspace, can be selected, and on which actions can be performed. Examples of objects are geometries, axes, frames, legends, streamtraces, zones, and XY mappings.

Using the Tecplot interface, there are a number of ways to select, or *pick,* objects. You can use the Selector or Adjuster tool to either click on objects or draw a box around objects. You can also use the Edit/Select All menu to select all objects of a specific type (e.g., all geometries or all zones). When an object is picked, Tecplot draws *selection handles* (graphics that indicate that the object is selected and allow you to manipulate it) or boxes around the object.

With the ADK, picked objects are handled through the pick list. The *pick list* is an indexed list of objects which are currently selected (usually, objects in the list are in the same order in which they were picked).

For details on any TecUtil function mentioned in this chapter, please see the *ADK Online Reference*.

18.1. Object Types

Object types are used in many of the ADK functions related to picked objects. An object type is specified with the **`PickObjects_e`** enumerated type. You may can pick the following types of objects:

```
PickObject_Frame
PickObject_Axis
PickObject_3DOrientationAxis
PickObject_Geom
PickObject_Text
PickObject_ContourLegend
PickObject_ContourLabel
PickObject_ScatterLegend
PickObject_XYLegend
PickObject_ReferenceVector
PickObject_ReferenceScatterSymbol
PickObject_StreamtracePosition
PickObject_StreamtraceTermLine
PickObject_Paper
PickObject_Zone
PickObject_LineMapping
PickObject_StreamTraceCOB
PickObject_SliceCOB
PickObject_IsoSurfaceCOB
```

18.2. Picking Objects

To pick objects with the ADK, you can use the **TecUtilPickSetMouseMode** function to set the mouse mode tool to be either the Selector or Adjuster. This will also clear out the pick list (i.e., unpick all picked objects).

The following functions are used to pick objects (i.e., to add objects to the pick list):

TecUtilPickAtPosition	Pick an object at a specified (X,Y) location.
TecUtilPickAddAll	Add all objects of a specified type to the pick list.
TecUtilPickAddAllInRect	Add all objects of a specified type and within a specified region to the pick list.

The **TecUtilPickAtPosition** function can either add to or replace what is already in the pick list. The **TecUtilPickAddAll** and **TecUtilPickAddAllInRect** functions always add to what is already in the pick list. This makes it easy, for example, to pick all text and all geometries with the following two commands:

```
TecUtilPickAddAll(PickObject_Geom);
TecUtilPickAddAll(PickObject_Text);
```

The functions **TecUtilPickGeom** and **TecUtilPickText** are also available to add a specific text or geometry to the pick list. The function which is used to unpick all objects or to clear out the pick list is **TecUtilPickDeselectAll**.

18.2.1. Picking Multiple Objects

In general, objects can only be selected within the current frame. For example, the following call will pick all zones within the current frame:

```
TecUtilPickAddAll(PickObject_Zone)
```

This is true even with **TecUtilPickAddAllInRect** when the specified region encloses objects in other frames.

An object type of **PickObject_Frame** allows multiple frames to be selected.

Picking objects with **TecUtilPickAtPosition** can change the current frame, but only if objects are not being collected.

18.3. Operating on Picked Objects

Once you are satisfied with the currently picked objects, you can use the following functions to operate on the pick list:

TecUtilPickEdit	Perform a specified action on all currently picked objects.
TecUtilPickCut	Copy all currently picked objects to the paste buffer and then clear them from the plot.
TecUtilPickCopy	Copy all currently picked objects to the paste buffer.
TecUtilPickPaste	Paste all objects which are currently in the paste buffer to the plot.
TecUtilPickClear	Clear all currently picked objects from the plot.
TecUtilPickShift	Move all currently picked objects in the plot.
TecUtilPickMagnify	Grow or shrink the size of all currently picked objects.
TecUtilPickPush	Push all currently picked objects to the back of the plot (so that they are drawn earlier).
TecUtilPickPop	Pop all currently picked objects to the front of the plot (so that they are drawn later).

Some of the above functions can only be used on specific types of objects. (For example, **TecUtilPickMagnify** can only be used on frames, text, and geometries.)

18.3.1. Example: Edit All Objects In the Pick List

The following code will add all geometries to the pick list and change their color, line pattern, position, and size:

```
TecUtilPickSetMouseMode(Mouse_Select);
TecUtilPickAddAll(PickObject_Geom);
TecUtilPickEdit("Color = Blue");
TecUtilPickEdit("LinePattern = Dashed");
TecUtilPickShift(1.2, 1.2, PointerStyle_AllDirections);
TecUtilPickMagnify(1.5);
```

18.4. The Pick List

The pick list is accessed through index values starting at 1. The procedure for enumerating the pick list begins with a call to **TecUtilPickListGetCount** to determine the number of objects that are currently in the pick list. This number can be used as a boundary condition as you loop through the pick list. For each object in the pick list, call **TecUtilPickList-GetType**. This will return the type of object in the pick list at the specified index. Once you have the object type, you can call appropriate functions to get more information:

Function Name	Used Only For Object Type
TecUtilPickListGetFrameName	PickObject_Frame
TecUtilPickListGetAxisKind	PickObject_Axis
TecUtilPickListGetAxisNumber	PickObject_Axis
TecUtilPickListGetZoneNumber	PickObject_Zone
TecUtilPickListGetZoneIndices	PickObject_Zone
TecUtilPickListGetLineMapNumber	PickObject_LineMapping
TecUtilPickListGetLineMapIndex	PickObject_LineMapping
TecUtilPickListGetText	PickObject_Text
TecUtilPickListGetGeom	PickObject_Geom
TecUtilPickListGetGeomInfo	PickObject_Geom

18.4.1. Example: Change Color of Text and Geometries In Pick List

The following piece of code will change the color of all text and geometries in the pick list to purple:

```
int Index;
int Count = TecUtilPickListGetCount();
for (Index = 1; Index <= Count; Index++)
  {
    PickObjects_e ObjectType = TecUtilPickListGetType(Index);
    /* We are only interested in text and
    /* geometries in the pick list.*/
    switch (ObjectType)
      {
        case PickObject_Text :
          {
           Text_ID TextObject = TecUtilPickListGetText(Index);
            TecUtilTextSetColor(TextObject, Purple_C);
          } break;
        case PickObject_Geom :
          {
           Geom_ID GeomObject = TecUtilPickListGetGeom(Index);
            TecUtilGeomSetColor(GeomObject, Purple_C);
          } break;
      }
  }
```

18.4.2. Example: Change Color Of Vectors In Pick List

The following piece of code will change the color of all vectors in the pick list to purple:

```
/* We will collect all the picked zones in a set. */
Set_pa Zones = TecUtilSetAlloc(TRUE);
int    Index;
int    Count = TecUtilPickListGetCount();
for (Index = 1; Index <= Count; Index++)
  {
    PickObjects_e ObjectType = TecUtilPickListGetType(Index);
    /* We are only interested in the zones in the pick list. */
    if (ObjectType == PickObject_Zone)
      {
       EntIndex_t ZoneNum = TecUtilPickListGetZoneNumber(Index);
       TecUtilSetAddMember(Zones, ZoneNum, TRUE);
      }
  }
TecUtilZoneSetVector("COLOR", Zones, 0.0,
                     (ArbParam_t)Purple_C);
TecUtilSetDealloc(&Zones);
```

If instead we wanted to change the color of all objects in the pick list to purple, we could have used the following code:

```
TecUtilPickEdit("COLOR = PURPLE");
```

CHAPTER 19 *Using Argument Lists*

Several TecUtil functions require a flexible, or extended, argument list due to ADK revisions or the need for a varied number of arguments. Utilizing extended argument lists also minimizes **`TecUtil`**'s name space growth which would otherwise be increased by natural revisions and extensions to the ADK. The ADK provides the `ArgList_pa` type and related functions to deal with extended functions.

When an extended function is created or as new capabilities are added to an existing extended function, reasonable default values are assigned whenever appropriate. The defaults also provide backward compatibility for add-ons written and compiled with prior versions of the ADK.

All functions that utilize the extended argument lists end with the capital letter **`X`,** distinguishing them from standard argument list functions. Where appropriate a standard argument list function is provided along with the extended version so that more common uses of the function are not burdened with the additional instructions needed to take advantage of the extended version's flexibility. Of course ADK developers are free to create additional standard argument functions by forwarding calls to the appropriate extended TecUtil functions.

For details on the TecUtil functions discussed in this chapter, please see the ADK Online Reference.

19.1. TecUtilArgListFunctions

Argument lists must be allocated and deallocated.

`TecUtilArgListAlloc` - Create an empty argument list.

`TecUtilArgListDealloc` - Dealloc an argument list.

You can manipulate the argument list using the following functions:

`TecUtilArgListClear` - Remove all members from the argument list.

`TecUtilArgListAppendInt` - Add a named integer argument to the list.

TecUtilArgListAppendDouble - Add a named double argument to the list.

TecUtilArgListAppendString - Add a named string argument to the list.

TecUtilArgListAppendArray - Add a named array argument to the list.

TecUtilArgListAppendFunction - Add a named function argument to the list.

TecUtilArgListAppendSet - Add a named set to the list.

TecUtilArgListAppendStringList - Add a named string list to the list

19.2. TecUtil Functions which use Argument Lists

Currently, argument lists are used in the following **TecUtil** functions:

- **Boolean_t TecUtilSaveLayoutX(ArgList_pa ArgList);**
- **Boolean_t TecUtilReset3DOriginX(ArgList_pa ArgList);**
- **Boolean_t TecUtilAnimateZonesX(ArgList_pa ArgList);**
- **Boolean_t TecUtilAnimateContourLevelsX(ArgList_pa ArgList);**
- **Boolean_t TecUtilAnimateIJKPlanesX(ArgList_pa ArgList);**
- **Boolean_t TecUtilAnimateIJKBlankingX(ArgList_pa ArgList);**
- **Boolean_t TecUtilAnimateStreamX(ArgList_pa ArgList);**
- **Boolean_t TecUtilAnimateSlicesX(ArgList_pa ArgList);**
- **Boolean_t TecUtilImageBitmapCreateX(ArgList_pa ArgList);**
- **Boolean_t TecUtilDataSetAddZoneX(ArgList_pa ArgList);**
- **Boolean_t TecUtilAnimateLineMapsX(ArgList_pa ArgList);**
- **Boolean_t TecUtilStateChangedX(ArgList_pa ArgList);**
- **Boolean_t TecUtilTransformCoordinatesX(ArgList_pa ArgList);**
- **Boolean_t TecUtilZoneCopyX(ArgList_pa ArgList);**
- **Boolean_t TecUtilCreateSliceZoneFromPlneX(ArgList_pa ArgList);**

19.3. Example of Using Argument Lists

Following is an example of how to set up and use the argument lists:

```
ArgList_pa ArgList = NULL;
Boolean_t  LayoutSaved = FALSE;
Boolean_t  IncludeData = FALSE;
Boolean_t  IncludePreview = FALSE;
Boolean_t  UseRelativePaths = FALSE;
char       LayoutFName[MAX_FNAME_LEN+1];
  /* Get the layout save options from the user. */
  .
  .
  .
  /* Create the argument list and initialize it with the save options. */
  ArgList = TecUtilArgListAlloc();
  if (ArgList != NULL)
    {
    TecUtilArgListAppendString(ArgList, SV_FNAME,     LayoutFName);
      TecUtilArgListAppendInt(ArgList,    SV_INCLUDEDDATA, IncludeData);
      if (IncludeData)
        TecUtilArgListAppendInt(ArgList, SV_INCLUDEPREVIEW,
IncludePreview);
      else
        TecUtilArgListAppendInt(ArgList, SV_USERELATIVEPATHS,
UseRelativePaths);


      /* Save the layout and cleanup the argument list. */
      LayoutSaved = TecUtilSaveLayoutX(ArgList);
      TecUtilArgListDealloc(&ArgList);
    }
  .
  .
  .
```

CHAPTER 20 *Using String Lists*

A *string list* in Tecplot's ADK is simply a list or a collection of strings. The *string list* object exists as a convenient way of dealing with groups of strings. The ADK provides the **`StringList_pa`** type and functions to deal with this type. Several TecUtil functions use string lists as parameters or return values. An example of when a string list is used is loading data - the list of data files to load is contained in a string list. The first data file name is the first string in the string list, the second data file name is the second string in the string list, and so on.

Tecplot keeps its own copy of any strings used when dealing with string lists. All strings which are passed to string list functions are copied by Tecplot before being added to the string list. All strings which are accessed from string lists are copied by Tecplot before being returned to the user. This means that you must deallocate your copy of any strings used when dealing with string lists. If the string came from Tecplot, you must use the **`TecUtilStringDealloc`** function to deallocate the string.

For details on the TecUtil functions discussed in this chapter, please see the *ADK Online Reference*.

20.1. TecUtilStringList Functions

String lists, like strings, need to be allocated and deallocated.

`TecUtilStringListAlloc`	Create an empty string list.
`TecUtilStringListDealloc`	Deallocate a string list.

You can get information about what's in a string list and manipulate the string list using the following functions:

`TecUtilStringListGetCount`	Get the number of strings in the string list.
`TecUtilStringListGetString`	Get the string at the specified index in the string list.
`TecUtilStringListAppendString`	Add a string to the string list.
`TecUtilStringListInsertString`	Insert a string at the specified index in the string list.

TecUtilStringListSetString	Set the string at the specified index in the string list.
TecUtilStringListClear	Remove all members from the string list.
TecUtilStringListRemoveString	Remove the string at the specified index from the string list.
TecUtilStringListRemoveStrings	Remove a specified number of strings from the string list at the specified index.
TecUtilStringListCopy	Create a copy of a string list.
TecUtilStringListAppend	Append the contents of one string list to another string list.

String lists can be converted to and from newline-delimited strings. A newline-delimited representation of a string list contains a newline character (**'\n'**) between each string in the string list.

TecUtilStringListToNLString	Create a newline-delimited string representation of a string list.
TecUtilStringListFromNLString	Create a string list from a newline-delimited string.

20.2. TecUtil Functions which use String Lists

Currently, string lists are used in the following **TecUtil** functions:

TecUtilReadDataSet
TecUtilOpenLayout
TecUtilDialogGetFileNames
TecUtilDataSetCreate
TecUtilImportSetLoaderInstr
TecUtilImportWriteLoaderInstr
TecUtilImportGetLoaderInstr
TecUtilReadBinaryData

20.3. Example of Using String Lists

Following is an example of how to set up and use string lists:

```
int           i = 0;
int           count = 0;
char          *file_name = NULL;
Boolean_t     selected = FALSE;
```

```
    StringList_pa selected_file_names = NULL;
    StringList_pa default_file_names = NULL;
    .
    .
    .
    /* Set up the default file name list. */
    default_file_names = TecUtilStringListAlloc();
    TecUtilStringListAppendString(default_file_names, "a.dat");
    TecUtilStringListAppendString(default_file_names, "b.dat");

    /* Ask user to select a bunch of add-on data files. */
    selected = TecUtilDialogGetFileNames(
                    SelectFileOption_ReadMultiFile,
                    &selected_file_names, "Add-on Data",
default_file_names, "*.dat");
    /* We do not need the default list of file names any more. */
    TecUtilStringListDealloc(&default_file_names);

    /* Process the results (for simplicity just print to
       standard output). */
    if (selected)
        {
        /* Ask the string list how many string items it
           maintains. */
        count = TecUtilStringListGetCount(selected_file_names);

        /* Print the header information. *
        printf("You selected the following files:\n");
        printf("---------------------------------\n");

        /* Print each one to standard output. */
        for (i = 1; i <= count; count++)
            {
            file_name =
              TecUtilStringListGetString(selected_file_names,i);
            printf("    %s\n", file_name);

            /* Deallocate return value as it is no longer
               needed. */
            TecUtilStringDealloc(&file_name);
            }

        /* We do not need the list of selected file names
           any more. */
        TecUtilStringListDealloc(&selected_file_names);
```

```
        }
    .
    .
    .
```

CHAPTER 21 *Using Sets*

A *set* in Tecplot's ADK is a group or a collection of zones, variables, or line-mappings. The set exists as a convenient way of dealing with groups of numbers. Each number that exists in a set is referred to as a member of the set. The ADK provides the **`Set_pa`** type and functions to deal with this type.

Several **`TecUtil`** functions use sets as parameters. These include the **`TecUtilZoneSet`*`xxx`*** and **`TecUtilLineMapSet`*`xxx`*** functions, which allow you to set the attributes of zones and Line-mappings. In this case, you use sets to describe which zones or Line-mappings you want to affect. Also included are several **`TecUtil`** functions which alter data. In this case, you use sets to describe which zones and/or variables you want to alter. Another example of when a set is used is with the **`TecUtilZoneDelete`** function. Instead of calling **`TecUtilZoneDelete`** once for each zone you want to delete, you create the set of zones which you want to delete and call **`TecUtilZoneDelete`** only once. The first zone number to be deleted is the first member of the set, the second zone number to be deleted is the second member of the set, and so on.

In many cases where a set is used, you may pass **`NULL`** to indicate all values. For example, **`TecUtilReadDataSet`** takes a parameter which is the set of zones you want to read. If you pass **`NULL`** for this parameter, all zones are loaded. To see whether you can use **`NULL`** in this manner, check the documentation for the function in question.

An important fact to remember is that a set will not accept the addition of a value that already exists in it. In other words, if the set already contains a value of 2, adding another 2 does not change the list of values OR the count of total values. This ensures a compact set without duplicates. If a list is needed that does happen to have repeating values it, is recommended that a *string list* is used in the place of a set.

For details on the **`TecUtil`** functions discussed in this chapter, please see the *ADK Online Reference*.

21.1. TecUtilSet Functions

Sets are allocated and deallocated with the following functions:

TecUtilSetAlloc	Create an empty set.
TecUtilSetDealloc	Deallocate a set.

You can manipulate sets with the following functions:

TecUtilSetAddMember	Add the specified member to the set.
TecUtilSetRemoveMember	Remove the specified member from the set.
TecUtilSetCopy	Copy the contents of one set to another.
TecUtilSetClear	Remove all members from the set.

You can get information about what's in a set with the following functions:

TecUtilSetIsMember	Determine if the specified member is in the set.
TecUtilSetIsEmpty	Determine if the set is **NULL** or contains no members.
TecUtilSetIsEqual	Determine if one set has the same members as another.
TecUtilSetGetMemberCount	Get the number of members in the set.
TecUtilSetGetMember	Get the member of the set at the specified position.
TecUtilSetGetPosition	Get the position in the set at which the specified member is located.
TecUtilSetGetNextMember	Get the member in the set which is after the specified member.

The **TecUtilSetForEachMember** convenience macro (which loops through each member of a set) is available only for C programmers. However, the functionality can easily be duplicated with the **TecUtilSetGetNextMember** function and a **for** or **while** loop.

Example of looping through all members of a set:

```
Set_pa ActiveZones = NULL;
SetIndex_t Zone;
TecUtilZoneGetActive(&ActiveZones);
/* Loop using TecUtilSetForEachMember. */
TecUtilSetForEachMember(Zone, ActiveZones)
  {
    if (TecUtilZoneIsFiniteElement(Zone))
      .
      .
      .
  }

/* Or, loop using TecUtilSetGetNextMember. */
```

```
Zone = TecUtilSetGetNextMember(ActiveZones,
                                    TECUTILSETNOTMEMBER);
while (Zone != TECUTILSETNOTMEMBER)
  {
    if (TecUtilZoneIsFiniteElement(Zone))
      .
      .
      .
    Zone = TecUtilSetGetNextMember(ActiveZones, Zone);
  }
```

21.2. TecUtil Functions which use Sets

Currently, sets are used in the following **TecUtil** functions:

TecUtilReadDataSet
TecUtilWriteDataSet
TecUtilDataAlter
TecUtilCreateMirrorZones
TecUtilPolarToRectangular
TecUtilRotate2D
TecUtilDataRotate2D
TecUtilAverageCellCenterData
TecUtilInverseDistInterpolation
TecUtilKrig
TecUtilLinearInterpolate
TecUtilLineMapGetActive
TecUtilLineMapSetActive
TecUtilLineMapSetName
TecUtilLineMapSetAssignment
TecUtilLineMapSetLine
TecUtilLineMapSetCurve
TecUtilLineMapSetSymbol
TecUtilLineMapSetSymbolShape
TecUtilLineMapSetBarChart
TecUtilLineMapSetErrorBar
TecUtilLineMapSetIndices
TecUtilLineMapDelete
TecUtilLineMapShiftToTop
TecUtilLineMapShiftToBottom
TecUtilTriangulate
TecUtilProbeAtPosition
TecUtilVarGetEnabled
TecUtilPickAddZones
TecUtilPickAddMaps

```
TecUtilZoneDelete
TecUtilZoneGetActive
TecUtilZoneGetEnabled
TecUtilZoneSetActive
TecUtilZoneSetMesh
TecUtilZoneSetContour
TecUtilZoneSetVector
TecUtilZoneSetVectorIJKSkip
TecUtilZoneSetScatter
TecUtilZoneSetScatterIJKSkip
TecUtilZoneSetScatterSymbolShape
TecUtilZoneSetShade
TecUtilZoneSetBoundary
TecUtilZoneSetVolumeMode
```

21.3. Example of Using Sets

The following example shows uses of all of the **TecUtilSet*xxx*** functions:

```
/*
 * Create two sets, A and B.  A will have 1,2,3 for its
 * members, and B will have 4 and 9.
 */
  Set_pa A;
  Set_pa B;
  A = TecUtilSetAlloc(TRUE);
  B = TecUtilSetAlloc(TRUE);
  if (A && B)
    {
      SetIndex_t Position, Member;
      /*
       * Add the members to the sets.
       */
      TecUtilSetAddMember(A,1,TRUE);
      TecUtilSetAddMember(A,2,TRUE);
      TecUtilSetAddMember(A,3,TRUE);
      TecUtilSetAddMember(B,4,TRUE);
      TecUtilSetAddMember(B,9,TRUE);
```

```
/*
 * Check to see if the sets are equal.
 */
if (TecUtilSetIsEqual(A,B))
  TecUtilDialogErrMsg("Something is wrong here");

/*
 * Clear out set A.
 */
TecUtilSetClear(A);
/*
 * Make A a copy of B.
 */
TecUtilSetCopy(A,B,TRUE);

/*
 * Get the position of the member '9' of set B
 * (the result is '2').
 */
Position = TecUtilSetGetPosition(B,9);

/*
 * Get the member located at position '2' of set A
 * (the result is '4').
 */
Member = TecUtilSetGetMember(A,Position);

/*
 * Get the member located after the member '4' of
 * set A (the result is '9').
 */
Member = TecUtilSetGetNextMember(A,Member);
```

```
  /*
   * Remove the first valid member from B.
   */
  if (TecUtilSetGetMemberCount(B) > 0)
    {
     int I = 1;
     while (!TecUtilSetIsMember(B,I))
       I++;
     TecUtilSetRemoveMember(B,I);
    }

  /*
   * Show a warning dialog if B is now empty.
   */
  if (TecUtilSetIsEmpty(B))
    TecUtilDialogMessageBox("B is empty",
                  MessageBox_Warning);

  /*
   * Finally, deallocate the sets.
   */
  TecUtilSetDealloc(&A);
  TecUtilSetDealloc(&B);
}
```

CHAPTER 22 *Using Standardized Auxiliary Data*

Auxiliary data has many uses. In some cases add-ons will want to use auxiliary data to store information exclusively for the use of that add-on alone. In other cases it may be useful to store information in auxiliary data that could be used by other add-ons or even Tecplot itself for further processing.

Add-ons that make exclusive use of auxiliary data must use names that will not collide with names used by other add-ons. For example if an add-on named ABCCalculator wishes to associate some auxiliary data representing the acidic level for each zone then using auxiliary names such as "ABCCalculator_AcidLevel" will reduce the likelihood of overwriting other add-ons' auxiliary data.

Add-ons that wish to communicate information to each other or to Tecplot via auxiliary data must agree on the names under which the data will be stored. To that end, Amtec is compiling a list of standard auxiliary data names. Figure 22-1 contains the preliminary list. Most of these names pertain to fluid dynamics or related physical processes. Data loaders will generally make assignments to these names for subsequent use by some type of post processing add-on such as Amtec's CFD Analyzer. Each new version of Tecplot will likely contain additions to this list. Note that all auxiliary data is stored in Tecplot as character strings. The Type column below indicates the type of data these character strings represent.

Name	Type	Auxiliary data is Assigned to
`Common.Incompressible`	Boolean	Dataset
`Common.Density`	double	Dataset
`Common.SpecificHeat`	double	Dataset
`Common.SpecificHeatVar`	int	Dataset

Figure 22-1.

Common.GasConstant	double	Dataset
Common.GasConstantVar	int	Dataset
Common.Gamma	double	Dataset
Common.GammaVar	int	Dataset
Common.Viscosity	double	Dataset
Common.ViscosityVar	int	Dataset
Common.Conductivity	double	Dataset
Common.ConductivityVar	int	Dataset
Common.AngleOfAttack	double	Dataset
Common.SpeedOfSound	double	Dataset
Common.ReferenceU	double	Dataset
Common.ReferenceV	double	Dataset
Common.UVar	int	Dataset
Common.VVar	int	Dataset
Common.WVar	int	Dataset
Common.VectorVarsAreVelocity	Boolean	Dataset
Common.PressureVar	int	Dataset
Common.TemperatureVar	int	Dataset
Common.DensityVar	int	Dataset
Common.StagnationEnergyVar	int	Dataset
Common.MachNumberVar	int	Dataset
Common.Axisymmetric	Boolean	Dataset
Common.AxisOfSymmetryVarAssignment(1)	int	Dataset
Common.AxisValue	double	Dataset
COMMON.SteadyState	Boolean	Dataset
COMMON.TurbulentKineticEnergyVar	int	Dataset
COMMON.TurbulentDissipationRateVar	int	Dataset
COMMON.TurbulentViscosityVar	int	Dataset
COMMON.TurbulentFrequencyVar	int	Dataset
COMMON.Gravity	double	Dataset
COMMON.IsBoundaryZone	Boolean	Zone

Figure 22-1.

COMMON.BoundaryCondition(2)	BCondition	Zone
COMMON.Time	double	Zone

Figure 22-1.

Table Footnote:

(1) Boolean can be assigned to any of "ON," "OFF," "TRUE," "FALSE," "YES," "NO."

(2) Boundary conditions include "Inflow," "Outflow,", "Wall," "Slip Wall," "Symmetry," and "Extrapolated."

Please note that the use of auxiliary data is optional. Add-on developers should always allow for the situation where a particular piece of auxiliary data does not exist.

CHAPTER 23 *Building Add-ons with FORTRAN*

To use FORTRAN in Windows, you must use Digital (Compaq) Visual Fortran. For the most part, only the standard FORTRAN compilers supplied with UNIX platforms are supported. Other FORTRAN compilers will probably work, but you may have to customize the settings/ build scripts.

For details on the **TecUtil** functions available for FORTRAN, see the Chapter "FORTRAN Glue Functions" in the *ADK Online Reference.*

23.1. FORTRAN Include Files

You must at a minimum include the file **FGLUE.INC** in the header of each function or subroutine that calls TecUtil functions. **FGLUE.INC** sets the return type for **TecUtil** functions and also defines a number of **PARAMETER** values that are handy to use when calling TecUtil functions. If you are using the Tecplot GUI Builder then you should also include **ADDGLBL.INC**, **GUIDEFS.INC**, and **GUI.INC**. **FGLUE.INC** and **GUI.INC** can be found in the **include** sub-directory below the Tecplot Home Directory. **ADDGLBL.INC** and **GUIDEFS.INC** are created uniquely for each add-on by the **CreateNewAddOn** shell script and the Tecplot GUI Builder respectively (UNIX) or the Tecplot GUI Builder Add-on Wizard (Windows). Thus, typical subroutines for FORTRAN add-ons to Tecplot should look like:

```
      SUBROUTINE MYSUB()
      INCLUDE 'ADDGLBL.INC
     INCLUDE 'FGLUE.INC'
      INCLUDE 'GUIDEFS.INC'
      INCLUDE 'GUI.INC'
C.... Code for this add-on
      RETURN
      END
```

23.2. Fortran Glue Functions

The FORTRAN glue functions currently supplied with Tecplot have been revised from those first published with Version 9.0. Even with the increased support for FORTRAN in this version there is still not a complete 1-1 correspondence of functions in the FORTRAN API with those in the C API. For the most part however, those functions missing in the FORTRAN glue

layer can be simulated by using the function **TecUtilMacroExecuteCommand** (see section 23.3, “Using the TecUtilMacroExecuteCommand Function,”) although this at times is not a complete solution.

The main goal in for enhancing the FORTRAN API for add-on development in this release of Tecplot was to allow FORTRAN programmers the ability to do the following:

- Write data loaders.
- Write data converters.
- Monitor state changes.
- Extend Tecplot's macro language.
- Access the pick list and thus all objects that can be picked.
- Access the “Dialog” functions.

Even though there has been a change to some of the functions, unless otherwise explicitly stated, existing add-on binaries will continue to work because they were linked with the former FORTRAN Glue Library and, being a static library, is incorporated with each add-on.

If your add-on is to be recompiled and you use any of the replaced or modified functions below, you will have to make some code changes.

23.2.1. Replaced FORTRAN Glue Functions

The following FORTRAN Glue Functions have been replaced or removed:

Version 7.5 Function	Replacement
TecUtilStateChangeDataSetReset	**TecUtilStateChanged**
TecUtilProbeGetIsNearest	**TecUtilProbeXXX** functions.
TecUtilSetFlagsOnVarValueChange	**TecUtilStateChanged**
fextregistertimeout	**TecUtilTimerAddCallback**
fextremovetimeout	Obsolete.

23.2.2. Modified FORTRAN Glue Functions

The following functions have been modified. In all cases the new function now mirrors exactly the corresponding C function. See the notes section for these functions in the FORTRAN glue function reference.

```
TecUtilDataSetCreate
TecUtilDataSetGetInfo
TecUtilStateChanged
TecUtilTecIni
TecUtilZoneGetInfo
```

23.3. Language Calling Conventions

The FORTRAN function signatures of many **TecUtil** functions vary little from their C counterparts. However, due to language differences some argument changes require additional explanation. Those differences are illustrated in this section and are referred to by several functions in the FORTRAN Glue reference section.

23.3.1. Sending String Parameters to Tecplot

Character strings used as parameters to **TecUtil** functions must be terminated with a character of value zero (not a "**0**"). This is necessary because the **TecUtil** functions really call C glue functions, and strings must be terminated with a **0** character value in C.

For example, calling the function **TecUtilDialogMessageBox** with the string "Hi Mom" will look like:

```
      I = TecUtilDialogMessageBox('Hi Mom'//char(0),
     &                            MESSAGEBOX_INFORMATION)
```

23.3.2. Receiving String Parameters from Tecplot

In order to take advantage of FORTRAN's native strings the **TecUtilStringAlloc** and **TecUtilStringDealloc** functions are not provided. As a result, all TecUtil functions in the C layer that either returned an allocated string as a function return value or did so by modification of an output parameter have been changed to use FORTRAN native strings. In addition, an accompanying **length** argument is also included so you can determine the actual length of the string.

Example:

Use the function **TecUtilDialogGetSimpleText** to prompt the user for their name.

```
INTEGER*4    IErr
INTEGER*4    NameLen
CHARACTER*80 UserName
IErr = TecUtilDialogGetSimpleText('Enter your name'//char(0),
```

```
&                                    'Joe Blow'//char(0),
&                                    UserName,
&                                    NameLen)
 Write(*,*) 'Name = ',UserName(1:NameLen),'<-'
```

In the above example, the variable **UserName** is filled in with the text of the users name. **NameLen** then is used to tell you just how much of the 80 characters available in **UserName** were used. The resulting write statement will have the '**<-**' placed just after the users name.

23.3.3. Handle Parameters

Many function in the **TecUtil** C layer allocate objects that are passed back and forth to the **TecUtil** layer, but are never manipulated directly by the add-on. String lists and sets are two examples of these objects, and they may only be manipulated via a collection of **TecUtil** functions. Objects of this nature are referenced in FORTRAN by the **POINTER** notation.

For example, suppose an add-on defined a subroutine to save variables 1 and 3 of zone 5 of the current data set. Two of several arguments required by the **TecUtilWriteDataSet** function are a zone and variable set. The **TecUtil** layer provides several functions for manipulating Tecplot sets. The code to perform this operation might appear as follows (error handling removed for simplicity):

```
      SUBROUTINE WriteMyData(FileName),
      CHARACTER*(*) FileName

      INCLUDE 'FGLUE.INC'
      POINTER (ZoneSetPtr, ZoneSet)
      POINTER (VarSetPtr, VarSet)
      CHARACTER*256 FileNameZ

c
c...allocate and populate the zone set: [5]
c
      TecUtilSetAlloc(ZoneSetPtr, TRUE)
      TecUtilSetAddMember(ZoneSetPtr, 5, TRUE)

c
c...allocate and populate the variable set: [1,3]
c
      TecUtilSetAlloc(TRUE, VarSetPtr)
      TecUtilSetAddMember(VarSetPtr, 1, TRUE)
      TecUtilSetAddMember(VarSetPtr, 3, TRUE)
```

```
c
c...be sure to pass a zero terminated string to TecUtil layer
c
      FileNameZ = TRIM(FileName)//char(0)

c
c...write the dataset to the specified file name
c
      IsOk = TecUtilWriteDataSet(FileNameZ, TRUE, TRUE, TRUE, TRUE,

     &                           ZoneSetPtr, VarSetPtr, TRUE, TRUE, TRUE)

c
c...cleanup allocations
c
      TecUtilSetDealloc(ZoneSetPtr)
      TecUtilSetDealloc(VarSetPtr)
      END
```

The **POINTER** variables, **ZoneSetPtr** and **VarSetPtr,** are the handles used to access the respective sets and are passed to the **TecUtilWriteDataSet** function while the **POINTER** variables, **ZoneSet** and **VarSet**, are dummy variables and should not be used.

Generally, unless directed by documentation for a specific **TecUtil,** function **POINTER** variables are meaningless to an add-on and should not be used.

Two exceptions to this rule are **TecUtilDataNodeGetRawPtr** and **TecUtilDataValueGetRawPtr,** where the **POINTEE** variables are arrays to Tecplot's internal data arrays and may be manipulated directly.

23.4. Special PARAMETER Values

The include file **FGLUE.INC** mentioned in Section 23.1, "FORTRAN Include Files," not only declares the return types for all **TecUtil** functions, it also defines a large number of **PARAMETER** values useful for inclusion in parameters to **TecUtil** functions. In the example of the previous section, the second parameter in the call to **TecUtilDialogMessageBox** is **MESSAGEBOX_INFORMATION**. If you look in the **FGLUE.INC** file (located in the **include** directory below the Tecplot Home Directory) you will find:

```
 PARAMETER(MESSAGEBOX_ERROR         = 0,
&          MESSAGEBOX_WARNING       = 1,
&          MESSAGEBOX_INFORMATION   = 2,
&          MESSAGEBOX_QUESTION      = 3,
```

```
&           MESSAGEBOX_YESNO        = 4,
&           MESSAGEBOX_INVALID      = 255)
```

It is best to always use the **PARAMETER** name instead of the number. It makes your code more readable, prevents errors, and makes it easier to upgrade later if the underlying value changes. In the *ADK Online Reference,* you can see what the possible values for a parameter are by looking at the C equivalent of the **TecUtil** function. In C, these **PARAMETER** values are defined as "enumerated types" and are listed with the description of the parameter. In C you must use the correct case when typing in the parameter name, but this is not necessary in FORTRAN.

23.5. UNIX Migration Issues

The following sections discuss issues encountered while developing FORTRAN add-ons under UNIX. Under Windows, the use of Visual Studio eliminates these issues.

23.5.1. Compiling Issues

The **Makefile** for add-ons created for Tecplot Version 10 using the **CreateNewAddOn** shell script must be modified as follows:

1. Change **ADDONDEVDIR** to be **TECADDONDEVDIR**.

2. Change **$(TEC100HOME)/lib/ext** to be **$(EXTBASEDIR)**.

23.5.2. Checking Fortran Source Using the fcheck Utility

Provided in the distribution is a UNIX shell script called **fcheck**. Use **fcheck** to examine Version 10 FORTRAN source files. **Fcheck** will report any misuse of functions (such as calling a sub-routine as if it were a function) and will warn you about all functions that have changed and or been removed.

To use **fcheck** type:

fcheck *filelist*

where *filelist* is one or more FORTRAN source files (***.F**).

23.6. Windows Issues

The following issues are specific to FORTRAN add-on development under Windows.

23.6.1. Calling Conventions

Digital (Compaq) Visual FORTRAN's default calling convention is **__stdcall**, with routine names all upper-case. All parameters are passed by reference, and the length of string parameters is passed immediately after the strings themselves. This is in contrast to UNIX, where string length parameters are appended to the list of parameters. It is important to use these default settings when you build your add-on, since libraries **fglue.lib** and **WinGUI.lib** assume that you are using them.

23.6.2. Writing to the Console

Under Windows, Tecplot add-ons do not have access to a console. Therefore, statements intended to write to the console (e.g. **write(*,*)**) will not have their intended effect. In fact, Visual FORTRAN Version 5 actually causes Tecplot to quit when such a statement is encountered. For this reason, you must avoid writing to the console. You may wish to write to a file instead, or call **TecUtilDialogMessageBox** to report information to the user.

Index