Using UIM/X with SoftBench

Integrated Computer
Solutions Incorporated

Integrated Computer Solutions, Inc.

54 Middlesex Turnpike, Bedford, MA 01730

Tel: 617.621.0060

Fax: 617.621.9555

E-mail: info@ics.com

WWW: http://www.ics.com

Contents

Preface

Overview

SoftBench is a development framework that allows utilities used in software development to interact. These utilities include tools such as editors, version control systems, compilers, and debuggers. UIM/X is designed to operate as an integral part of the SoftBench software development environment.

Who Should Use this Guide

This manual assumes you are familiar with the basics of UIM/X. Before using this manual, review the *UIM/X Beginner's Guide* and the *UIM/X User's Guide.*

This manual also assumes that you are a software developer with knowledge of programming, a general understanding of the X Window System, and familiarity with UIM/X and SoftBench. You should also know how to use common items such as menus, buttons, and scroll bars. If you are not familiar with these items, you may find it useful to review the *OSF/Motif User's Guide* and the *UIM/X Motif Developer's Guide*.

Before you begin, check with your system administrator to ensure that the software has been installed as described in the *UIM/X Installation Guide*.

Before You Read this Guide

This guide makes the following assumptions:

- You are familiar with the basic functions of selecting from menus and dialog boxes; opening, moving, resizing and closing windows, and clicking icons.
- You understand the functions of the three mouse buttons, which this guide refers to as the Select button (left button), the Adjust button (middle), and the Menu button (right). See “Using the Mouse” on page ix for more information.

The UIM/X Document Set and Related Books

This section lists the UIM/X document set, and provides a suggested list for further reading.

The following list is the complete UIM/X document set:

- *UIM/X Installation Guide*. Explains how to install and run UIM/X. Includes information on the files provided with UIM/X, backwards compatibility issues, and compiler considerations.
- *UIM/X Beginner's Guide*. Introduces UIM/X by presenting Novice Mode, the simplified Palette that enables new users to be productive immediately. Includes information on a number of important features for creating, testing, and running applications.
- *UIM/X Tutorial Guide***.** A series of step-by-step tutorials, teaching tools and techniques that will greatly assist you in developing your own applications. Features tutorials in Novice Mode, Standard Mode, and on advanced topics.
- *UIM/X User's Guide*. Explores the UIM/X features essential to GUI development. Includes discussions of how to use UIM/ X's editors to set properties, add behavior, etc.
- *UIM/X Motif Developer's Guide*. An in-depth guide to the widgets, features and capabilities of UIM/X as they relate specifically to Motif development.
- *UIM/X Advanced Topics***.** Describes how to customize UIM/X, including integrating new widget and component classes into the executable. Includes reference information of an advanced technical nature.
- *UIM/X Reference Manual*. A comprehensive list of properties, methods, and events, plus more, for Motif development. Designed for the experienced developer.

Suggested Reading

For more information on designing GUIs, see any of the following books:

- *OSF/Motif Style Guide release 1.2* (Prentice Hall, 1993, ISBN 0-13-643123-2)
- *Visual Design with OSF/Motif* (by Shiz Kobara, Addison-Wesley, 1991, ISBN 0-201-56320-7)
- *New Windows Interface: An Application Guide* (Microsoft Corporation, 1994, ISBN 1-55615-679-0)
- *Human Interface Guidelines: The Apple Desktop Interface* (Addison-Wesley, 1987, ISBN 0-201-17753-6)

How this Guide Is Organized

- *Chapter 1, "Using the UIM/X SoftBench Integration,"* describes how to use UIM/X with SoftBench.
- *Chapter 2, "Using Encapsulator Libraries,"* provides an example that describes how to use the SoftBench Encapsulator libraries.
- *Appendix A, "Messages Reference,"* contains reference pages for requests and messages sent and received by UIM/X.
- *Appendix B, "Error Messages,"* lists the SoftBench related error messages that might be displayed in the Messages area of the UIM/X Project Window.

Some Terms You Should Know

Certain basic terms recur throughout this guide, and it helps to understand them from the outset.

An *object* is a building block you can use to build an interface with UIM/X.

A *Motif widget* is an object whose appearance and behavior precisely follows the *OSF/Motif Style Guide*. The novice mode of UIM/X supports a number of popular Motif widgets, including Push Button, Label, Text Field, and more.

A compound object consists of several Motif widgets combined into one object for your convenience. The novice mode of UIM/X supports a number of compound objects, including Application Window and Group Box, that save you the time you might otherwise spend creating them.

An *interface* is a window or dialog box that you build up from objects with UIM/X. The novice mode of UIM/X supports four different types of interfaces: Application Window, Secondary Window, Message dialog box, and File Selection dialog box. Certain menu options refer to an interface, such as Save Interface; these act only on your selected interface.

A *project* contains all the interfaces (i.e., windows and dialog boxes) and their associated files for a certain GUI you are building with UIM/X. The program can automatically save and generate code for an entire project in one step. Certain menu options refer to a project, such as Save Project; these act on all the windows and dialog boxes in your project.

Conventions Used in this Guide

Typographic Conventions

The following table describes the typographic conventions used in this guide.

Typeface or Symbol	Meaning	Example
`AaBbCc12`	The names of commands, files, and directories; or onscreen output; or user input.	Edit your `.login` file. Use `ls -a` to list all the files. `%You have mail.`
AaBbCc12	A placeholder you replace with your actual value; or words to be emphasized; or book titles.	To delete a file, type `rm` *filename.* You *must* be `root` to do this. See Chapter 6 in the *User's Guide.*
File⇒Open	The Open option in the File menu.	Choose the File⇒Open command.
Alt+F4	Press both Alt and F4 at once.	Press Alt+F4 to exit.
Return	The key on your keyboard marked Enter, Return, or .	Press Return.

Installation Directories

Product installation directories can depend on the platform or the user's preferences. To keep things simple, this guide uses general names for product installation directories. The following table lists the name and the corresponding product installation directory:

Name	Description
uimx_directory	The UIM/X installation directory
softbench_directory	The SoftBench installation directory

Using the Mouse

Before starting the tutorial, take a moment to review the location and usage of your mouse buttons, as illustrated in the Figure P-1and the following table:

1: Select 2: Adjust 3: Menu

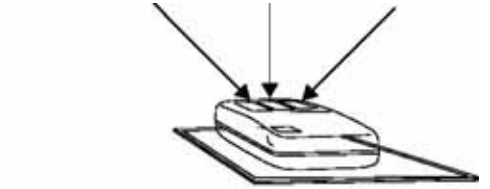

Figure P-1 The Mouse Buttons

Button:	Called:	Is used for:
1	Select	Selecting objects, menus, toggles, and options.
2	Adjust	Resizing and moving objects.
3	Menu	Displaying popup menus.

Throughout this book, you will use the mouse buttons along with the mouse pointer to make selections, move the input pointer, or position the text insertion point. You can perform any of the following mouse operations.

Operation	Description
Point to	Move the mouse to make the pointer go as directed.
Press	Hold down a mouse button.
Release	Release a mouse button after pressing it.
Click	Quickly press and release a mouse button without moving the mouse.
Drag	Move the mouse while pressing a mouse button.
Double-click	Click a mouse button twice in rapid succession without moving the mouse pointer.
Triple-click	Click a mouse button three times in rapid succession without moving the mouse pointer.

In general, instructions for mouse operations include the name of the mouse button. The exceptions are Click, Double-click, and Drag. These common operations may be described without specifying a mouse button. For example:

- Click on the `applWindow1` icon in the Interfaces Area of the Project Window.
- Drag the Push Button icon from the Palette.

In these cases, use the Select button to click and double-click, and the Adjust button to drag.

Setting Application Defaults

Application Defaults configure the way UIM/X looks and set the default preferences for many of its operations. You can set the Application Defaults for all UIM/X users or for a single user. For more details on setting your Application Defaults see *The UIM/X Developer's Guide*.

For optimum performance, set the following resources in your Application Defaults:

```
Mwm*autoKeyFocus: falseMwm*clientAutoPlace:
falseMwm*focusAutoRaise:
falseMwm*focusFollowsPointer:
trueMwm*keyboardFocusPolicy: pointer
```

If you have a gray-scale monitor, you might try the following settings:

```
Mwm*activeBackground: #666666
(gray40)Mwm*activeForeground: #e5e5e5
(gray90)Mwm*background: #666666
(gray40)Mwm*foreground: #e5e5e5
(gray90)Uimx3_0*calculatedColors:
falseUimx3_0*background: #ededed
(gray93)Uimx3_0*BottomShadowColor: #000000
(black)Uimx3_0*foreground: #000000
(black)Uimx3_0*TopShadowColor: #ffffff
(white)Uimx3_0*XmText.background: #b3b3b3
(gray70)Uimx3_0*XmTextField.background: #b3b3b3
(gray70)
```

Note: The resources above prefixed with `Mwm` are specific to the Motif Window Manager. If you are using a different window manager consult your Systems Administrator for the equivalent settings.

Using the UIM/X SoftBench Integration 1

Overview

When UIM/X runs under SoftBench, it uses the SoftBench framework to implement several basic features, such as editing code and make files and compiling project code. New menu entries have been created to access functionality available only through SoftBench.

Starting UIM/X From the SoftBench Tool Status Display Dialog

1. The system must be able to find the SoftBench executable. If you have not already done so, add *softbench_directory* to your path. For example, if your *softbench_directory* is `/opt/softbench/bin`, add the following to your PATH environment variable:

   ```
   /opt/softbench/bin
   ```

2. Start SoftBench by entering the following command at the shell prompt:

   ```
   softbench
   ```

For more details, consult your SoftBench documentation.

3. Click on the Show Running Tools button in the ToolBar window. The Tool Status Display window is displayed on the screen.
4. Choose Tool⇒Start Auxiliary Tools from the Tool Status Display window.

   The Tool Status Display - Start window is displayed on the screen.
5. Double click on the UIBUILD tool in the scrolled list. If this entry is not in the list, SoftBench has not been configured with UIM/X.

   While UIM/X is being initialized, the *Starting* message is displayed in the Tool Status Display window. The copyright notice for UIM/X is also displayed. If you click on the Cancel button, UIM/X exits.

When UIM/X is initialized the *Ready* message is displayed in the SoftBench Tool Status Display window.

Adding UIM/X to the ToolBar

1. The system must be able to find the SoftBench executable. If you have not already done so, add *softbench_directory* to your path. For example, if your *softbench_directory* is `/opt/softbench/bin`, add the following to your PATH environment variable:

   ```
   /opt/softbench/bin
   ```

2. Start SoftBench by entering the following command at the shell prompt:

   ```
   softbench
   ```

   For more details, consult your SoftBench documentation.
3. Choose Options⇒ToolBar Setup in the ToolBar window.
4. Click on the UIM/X tool in the Available Tools scrolled list in the ToolBar Setup window. If the UIM/X tool is not in the list, SoftBench has not been configured with UIM/X.

5. Click on the Add to ToolBar button, then click OK in the ToolBar Setup window.

   The ToolBar Setup window closes and a UIM/X icon appears in the ToolBar window.

6. Double click on the UIM/X icon to start UIM/X.

Starting UIM/X from the SoftBench Development Manager

To start UIM/X:

1. Double click on the Development Manager icon in the ToolBar window. The SoftBench Development Manager window appears.
2. Double click on a UIM/X file. This action will automatically load the file into UIM/X.

OR

2. Select the file by clicking on it and choose Actions⇒UIBuild.

Note: A new UIM/X process is started if one is not already running. A new process is also started if the context directory setting of the SoftBench Development Manager (DM) and UIM/X are different (which depends on UIM/X's context scope setting).

Setting Context for UIM/X

UIM/X will change its context if one of the following occurs:

1. The current context directory is changed when:
   - You enter a new directory in the File Selection box that appears when you select Options⇒Current Directory from the UIM/X startup interface.
   - UIM/X receives a SET-CONTEXT request containing a new directory.

Note: UIM/X does not support the context host for the change in file context.

Command-Line Options

UIM/X supports the following standard SoftBench command-line options:

Option	Description
`-host` *host_name*	Used only when UIM/X is running in the SoftBench environment. Since UIM/X does not support host context, the host context will always be set to the host name of the local machine.
`-dir` *directory_path*	Sets the current directory context to *directory_path*.
`-file` *file_name*	Loads *file_name* into UIM/X, where the file is one of the following types: project, interface, or palette.

Exiting UIM/X From SoftBench

1. Select UIBUILD in the SoftBench Tool Status Display window.
2. Select Tool⇒Stop from the Tool Status Display window.

When you exit UIM/X, a STOP notification is broadcast. This occurs under one of the following conditions:

- You select File⇒Exit from the UIM/X startup interface.
- UIM/X receives a STOP request from another tool.

Before exiting, UIM/X will send a STOP request to all tools that were started from UIM/X.

Note: All temporary files created by UIM/X for use by other tools are removed.

Files saved by the exiting tools after UIM/X has begun exiting will not be incorporated back into UIM/X. Therefore, if you want to save the contents of your edit session, save them under a new file name.

Running the SoftBench Program Builder from UIM/X

From the Generate Code Options dialog you can invoke the SoftBench Program Builder using the following guidelines:

- If only the Run Makefile toggle is set, a build is started.
- SoftBench Program Builder runs a make on the specified target, using the makefile.
- If only the Run Executable toggle is set, a message is sent to a Termsrv tool to run the specified executable using the arguments as input parameters to the executable.
- If both toggles are set, the build is started. At the same time, UIM/X also sends a SET-MAKEFILE-NAME request to SoftBench Program Builder. This allows you to subsequently build the executable from within SoftBench Program Builder and to display the results of the build in Messages Area of the UIM/X startup interface. When UIM/X receives notification that the build was successful, then it will run the executable.

Note: SoftBench status messages are displayed in the Messages Area. Detailed listings of compilation error messages are displayed in the SoftBench Program Builder.

Using Run Mode

The relationship of Run Mode to Test Mode and Design Mode causes the following behaviors:

- Going from Design Mode to Run Mode iconifies SoftBench Program Editor, brings up SoftBench Program Builder, and if the build is successful, the executable runs.
- Going from Run Mode to Design Mode stops SoftBench Program Builder, terminates the executable running via the terminal server, and normalizes any iconified tools.
- Going From Test Mode to Run Mode unblocks any message events, normalizes any iconified terminal server, starts up SoftBench Program Builder and if the build is successful, the executable runs.
- Going From Run Mode to Test Mode stops SoftBench Program Builder, and the executable running via the terminal server.

Using the SoftBench Editor from UIM/X

When running UIM/X under SoftBench, pressing the Text Editor (…) button brings up your SoftBench configured text editor.

UIM/X copies its contents into a temporary file and invokes an EDIT WINDOW request, passing the file name as a handle.

Whenever UIM/X receives a FILE-MODIFIED notification from the EDIT tool that serviced the EDIT WINDOW request (whenever you save the tool's buffer), the contents of the file are read back into the Text widget.

When closing a UIM/X editor (for example, the Declarations Editor), all EDIT sessions initiated from the UIM/X editor are closed, and the temporary files are deleted. Any unsaved changes will *not* be incorporated back into the Text Widget.

Exceptions

The Callback Editor, and Interpreter Window are special cases. These windows already contain a scrollable Text widget for entering text, and do not have a Text Editor (…) button that can call a SoftBench editor. To invoke a SoftBench EDIT tool from these editors choose Modify Code from the Edit pulldown menu in the menu bar.

Using the Version Control Menus

Some of the menus contain a submenu called Version Control that allows you to check in, check out, and check out and lock a file by sending requests to the SoftBench Configuration Manager (CM).

The Version Control submenu appears in the following places:

- The File menu in the Project Window, Palette, and Browser.
- The Selected Interfaces pop-up menu in the Interfaces Area of the Project Window.
- The Selected Palettes pop-up menu in the Palettes Area of the Project Window.
- The Tools menu in the Startup Interface.

The files that can be checked in and out vary, based on what menu is used:

Menu	Associated Files	Acts Upon
Project Window File menu	Project files	The loaded project.
Palette File menu	Palette files	The loaded palette.
Browser File menu	Interface files	The interface loaded in the Browser.
Selected Interfaces pop-up menu	Interface files	The selected Interfaces in the Interfaces Area of the Project Window.
Selected Palettes pop-up menu	Palette files	The selected palettes in the Palettes Area of the Project Window.
Startup Interface Tool menu	Any file	Any file selected in the File Selection box.

All of the Version Control submenus display a three-item submenu containing these options: Check In, Check Out, and Check Out & Lock.

As an example of usage, when one of these menu items is chosen from the Version Control submenu in the startup interface (Project Window, Browser) Tool pulldown menu, UIM/X displays a file selection box where you can enter a file name.

Note: SoftBench status messages are displayed in the Startup Interface Messages Area.

Version Control requirements and restrictions:

- Files must already be under version control.
- The version control directory must be in the current context directory.
- You must be able to write to the current context directory.

Version control on a project will only work on the project file itself. Unlike the Save Project command, which saves the project file and all associated files, version control is applied only to the project file.

Message Integration Between SoftBench and UIM/X

Request messages are accepted by UIM/X from other SoftBench tools. UIM/X will respond with a message if there is an error.

Loading UIM/X Files from the SoftBench Development Manager

When you load a UIM/X file from SoftBench Development Manager, a LOAD-UIFILE request to UIM/X is sent to UIM/X. The context operand specifies the file to load.

UIM/X will send a failure message only if the load failed. If the file was loaded in its entirety, but errors occurred while parsing the file, a notification is sent rather than a failure message.

When loading a project file into a UIM/X session that already contains a project, all SoftBench tools that were opened from this session are sent a STOP request and UIM/X resets and loads in the new project. Before loading the project UIM/X displays a warning dialog box, allowing you to cancel the request.

Importing Motif UIL Files

You can import a Motif UIL (User Interface Language) file from other SoftBench tools by sending an IMPORT-UIL request to UIM/X. The context operand specifies the file to import. A reply is sent back indicating the result of the command.

Doing Window Control

You can iconify or normalize all the windows in UIM/X by sending an ICONIFY or NORMALIZE request to UIM/X, respectively. In addition, any SoftBench tool started by UIM/X will also be affected.

Putting UIM/X in Test Mode

When you put UIM/X in Test Mode, all SoftBench tools that were opened from your session will be sent an ICONIFY request. UIM/X will block all message events from the BMS. This action protects the code from being changed during execution.

When you change back to Design Mode, all SoftBench tools that were iconified are sent a NORMALIZE request to open their windows and UIM/X accepts all blocked messages that were queued by the BMS.

Note: If you open some of the iconified EDIT tool windows, make modifications and save them while UIM/X is in Test Mode, these changes will not take effect in UIM/X until you change back to Design Mode.

Putting UIM/X in Run Mode

When you put UIM/X in Run Mode, all SoftBench tools that were opened from your session, will be sent an ICONIFY request. UIM/X will block all message events from the BMS. This action protects the code from being changed during execution.

The SoftBench Builder tool will be sent a MAKEFILE-BUILD request that will initiate the compilation of the application. The SoftBench Termsrv tool will be sent a NO-STDIO request that will run the application.

When you change back to Design Mode, the SoftBench Builder tool is sent a STOP request, the Termsrv tool is sent a KILL-ALL request, all SoftBench tools that were iconified are sent a NORMALIZE request to open their windows, and UIM/X accepts all blocked messages that were queued by the BMS.

Saving and Generating Files

A FILE-MODIFIED notification (one per file) is broadcast when one of the following occurs:

- Saving interfaces, palettes or a project.
- Generating code.

Using Encapsulator Libraries

2

Overview

The combination of UIM/X and the SoftBench encapsulator is a powerful tool for building applications that provide a messaging-aware GUI to an existing command line-driven tool. In this section, an example demonstrates using

UIM/X and the SoftBench Encapsulator libraries to develop GUI's for encapsulations. The Encapsulator libraries provide subprocess control and message handling. UIM/X produces the GUI, main program body, and Makefile.

This section shows by example the steps required in creating an encapsulation with UIM/X and the SoftBench encapsulator libraries. The goal is to provide a graphical user interface (GUI) and a broadcast message server (BMS) interface to an existing application. Understanding this example requires familiarity with UIM/X and the SoftBench Encapsulator.

The Application Before Encapsulation

The application in the example is a simple tty-based application to lookup phone numbers from a simple database. The phone database application is command line driven.

The existing application prompts the user for a command, reads and parses the command, performs the query indicated, outputs the results, and then prompts again.

The commands supported by this application are:

Command	Description
Print	Print the contents of the database.
GetByLastName	Take one argument and return any entries whose Last Name field matches this argument.
GetByFirstName	Take one argument and return any entries whose First Name field matches this argument.
GetByNumber	Take one argument and return any entries whose Number field matches this argument.
GetByAreaCode	Take one argument and return any entries whose Area Code field matches this argument.
Quit	Exit the application.

When the application comes up, it prints the string `Command?` to standard out to prompt the user for a command. The user enters any of the above commands, providing an argument if necessary. The application will perform the query and print any results. It will then print the `Command?` prompt and await further user input. The application performs this loop until the user enters the `Quit` command.

Encapsulating the Application

The encapsulated application will have a graphical user interface (GUI). The message interface will allow other tools to initiate queries. All of the functionality of the original application will be available via the GUI.

There will be buttons to invoke each of the user commands described above. The “Print” and “Quit” buttons don’t require any arguments, so these actions will correspond directly to commands of the application. The other buttons will correspond to commands which take an argument, so an input area is

needed to provide for the user to supply the argument. For this example, there is an input box named `text1`. Output from any of these commands will be displayed in a text output box, `text2`.

The Steps in This Tutorial

This tutorial takes about 90 minutes to complete. It contains the following steps:

Step #1: Building the Application Interface

Step #2: Setting Up the Main Event Loop

Step #3: Creating the Subprocess Object

Step #4: Integrating the GUI with the Subprocess Object

Step #5: Adding Messaging to the Application

Step #6: Initializing the Encapsulation from the GUI

Step #7: Writing Files and Compiling the Application

Step #1: Building the Application Interface

Use UIM/X to build an interface with buttons to initiate actions, a text area to enter search parameters, and a text area to display search results.

To allow for external access to the interface, use the Method Editor to define methods like the following to clear the display area, enter text into the display area, and retrieve text from the display area:

```
phone_clear_display( swidget UxThis, Environment
   * pEnv )

phone__get_display_text( swidget UxThis,
   Environment *pEnv)

phone__set_display_text( swidget UxThis, char
   *data, Environment *pEnv)
```

Step #2: Setting Up the Main Event Loop

UIM/X uses a different Motif application context (UxAppContext) than the Encapsulator. Because of this, the Encapsulator main loop hook functions must be set to use this different application context. The main program and make file templates include an implementation of this event loop. These can be enabled by performing the following steps in the Program Layout Editor.

1. Select Options⇒Loop⇒Explicit Loop.
2. Edit the Makefile template to add the following lines:

```
ENCAP_LIBPATH= -L$(SB_DIR)/lib/SB5.0
   -R$(SB_DIR)/lib/SB5.0ENCAP_LIBS= -lencapinit
   -lencap_now -lbms -lspc -lfw -lsoftlib

ENCAP_CFLAGS= -I$(SB_DIR)/include/SB5.0
   -DSB_ENCAP
```

More details on how to modify the Encapsulator main event loop can be found in the *SoftBench Encapsulator User's Guide*.

Step #3: Creating the Subprocess Object

A subprocess object is created to execute and communicate with the application program. An application event is set up to monitor all output from the subprocess; all output, excluding prompts, is displayed on the GUI using the `phone__set_display_text()` method. Another application event handles prompts, retrieving text from the GUI with the `phone__get_display_text()` method and including it in a notification message.

Define Functions to Send Commands to the Application

In this example, it is possible to write a single routine, `send_phone_command()` which handles all input to the subprocess. This routine takes two arguments, the command name and a command argument. This argument can be `NULL` for commands which don't take an argument ("Print", "Quit"). This implementation of `send_phone_command()` will also clear the text output box before it sends the output to the subprocess:

```
/* send_phone_command
    *
    * This function is used to send commands to
   the subprocess.
    * First, the GUI is cleared. Then, a command
   string is
   * built and sent to the subprocess with
   encap_send_command().
    */

int send_phone_command(const char *cmd, const
   char *data)
   {


encap_boolean result = encap_False;
   encap_boolean have_data = (data && *data !=
   '\0'); phone_clear_display(phone, &UxEnv);
   result |= encap_send_command(spc_obj, cmd,
   !have_data); if(have_data) {

    result |= encap_send_command(spc_obj, " ",
   encap_False);
    result |= encap_send_command(spc_obj, data,
   encap_True);

}
    return result;


}
```

Each time a command is sent, `send_phone_command()` will clear the text output box. This is the desired behavior that causes queries to the database to be logically distinct.

Define Functions to Manage Application Events

This example application has two kinds of output:

- Regular output which is the result of any of the queries.

  The system does not guarantee that all data from the subprocess will be delivered to the encapsulation at one time. Because of this, make sure the callbacks for these application events concatenate the results to the text output box. Again, because `send_phone_command()` clears this output box, you don't have to put any logic in the callback handler to tell the difference between an initial burst of data and a continuation burst.

- `Command?` prompt.

  Logically, this signals that the previous query has completed and the application is ready to perform another. In order to avoid cluttering up the text output box, you don't want to display this string. In addition, because this logically signals the end of the query, you might want to do some additional processing at this time. This will be revisited in Step 5 on messaging.

```
/* print_results
    *


 * This callback is invoked when the encapsulation
   receives * output from the subprocess. The
   "Command?" prompt is

* handled specially. If the output is not
   "Command?", it is


 * displayed on the GUI with
   phone__set_display_text(). */static void
   print_results (encap_object obj,
   encap_instance inst, void* data){ encap_string
   this_string =
   encap_get_instance_pattern_var(inst, 0);
   if(strncmp("Command", this_string,
   7))phone__set_display_text(phone, this_string,
   &UxEnv);


 encap_free_string(this_string);
   }
```

```
/* notify_results
    *

 * This callback procedure is invoked when the
   encapsulation

* receives the "Command?" prompt from the
   subprocess. Text

 * stored in the GUI is retrieved via the function

 * phone__get_display_text(), and a reply message
   is

* constructed and sent. */static void
   notify_results (encap_object obj,
   encap_instance inst, void* data)


{

 char *txt = phone__get_display_text(phone,
   &UxEnv);

 encap_send_reply(msg_obj, reply, (txt && *txt) ?
   "PASS" :

"FAIL", txt); reply=encap_NULL;}
```

Define a Function to Create the Subprocess Object

Create a subprocess object and prepare it to handle application events. When creating the subprocess object, differentiate between the two different kinds of output by using two different application events.

- The first application event handles the output from queries, and is created with a very general regular expression (`"(.*)\n"`). Its callback simply takes the output from the application event and displays it in the text output window. You don't want to display the `Command?` prompt, so it is filtered out in the generic output callback.
- The second application event is called when the `Command?` string is sent by the subprocess, and is created with this very specific regular expression (`Command?`). When creating the subprocess object, specify both of these events:

```
/* set_up_application_events
    *

 * This function is used to create the subprocess
   object and
    * to register application event handlers.
   Finally, the
    * subprocess is started.
    */

void set_up_application_events(char *cmd)
   { encap_event app_event1, app_event2;
   app_event1=encap_make_event(encap_Application,
   "Command",

notify_results, 0);
   app_event2=encap_make_event(encap_Application,
   "(.*)\n",print_results, 0);

 spc_obj = encap_make_object(encap_NULL, "spc",
   encap_Subprocess, encap_NULL,
   encap_merge_attribute(encap_COMMAND(cmd),encap
   _MODE("TerminalMode")),app_event1, app_event2,
   0);

 encap_free_event(app_event1);
   encap_free_event(app_event2);
   encap_start_process (spc_obj);

}
```

Step #4: Integrating the GUI with the Subprocess Object

You can now attach application behavior to the GUI elements. For this example, you need to send some input to the application when buttons are pressed, and take the output from the application and display it in the GUI.

To access functions defined in the encapsulation interface, the interface will need to include the function prototypes. In the "Includes, defines, global variables" section of the Declarations Editor, include all necessary header files:

```
#include <stdio.h> #include <encap_object.h>
```

Note: *encap_object.h* represents a header file that includes declarations for functions that you have defined for your encapsulated object.

To send input to the subprocess when buttons are pressed, we need to provide some code for the `ActivateCallback` property under the Behavior category in the Property Editor. This example is simple enough that all the buttons can call `send_phone_command()` directly in this callback. In the cases where an argument is wanted, call the function `UxGetText(text1)` to get the value out of the text input box. Thus, the `ActivateCallback` for the Print button would be:

```
send_phone_command("Print", "")
```

The `ActivateCallback` for the Get By First Name button would be:

```
send_phone_command("GetByFirstName",
   UxGetText(text1))
```

The rest of the callbacks would be:

```
send_phone_command("GetByLastName",
   UxGetText(text1));

send_phone_command("GetByAreaCode",
   UxGetText(text1));

send_phone_command("GetByNumber",
   UxGetText(text1));

send_phone_command("Quit", "")
```

Step #5: Adding Messaging to the Application

A message object is created to receive and send messages. Message requests are parsed, and corresponding commands are sent to the subprocess via `send_command()`. Reply notifications are constructed and sent via the message object.

Define Functions to Handle Messages

It is necessary to send a message to the encapsulation, have it use this message as a query to the application, and return the results as the reply to

the message. For convenience, use the application commands (“Print”, “GetByFirstName”, etc.) as the messaging commands. This saves having to translate the command field from the inbound messages.

When creating the message object, specify which event handler will be invoked for the incoming messages. Because the sample application is so simple, we have only two classes of messages to handle:

- `STOP` requests
- Database query requests

For simplicity, arrange it so that all database query requests are handled by the same function, `receive_msg()`. The `STOP` request can be handled by a separate function, `stop()`.

```
/* stop
    *


 * This callback procedure is invoked when the


* encapsulation receives a STOP message. Its
   purpose is


 * to ensure that a reply message is sent and that
   the


* encapsulation terminates. */static void
   stop(encap_object obj, encap_instance inst,
   void* data)
```

```
{
   /* When exiting a program, an explicit reply
   MUST be

 * sent to the procedure exiting before the
   default

* reply can be sent. */encap_send_reply(msg_obj,

encap_get_instance_replyto(inst), "PASS",
   0);#ifndef
   DESIGN_TIMEencap_quit(encap_True);#endif /*
   DESIGN_TIME */
}
```

The `receive_msg()` function does two things:

- Takes the command name and any message data and passes them to `send_phone_command()`, the application object function discussed in the previous section.
- Record the *replyto* from the incoming message in a global variable for later use, and indicate to the messaging system that the reply will be delayed. This is done to collect the output from the command and return it as the reply for the message.

As mentioned in Step 3, “Create the Subprocess Object”, the application event callback for the `Command?` string is the place to do any special processing when queries are complete. In this case, this callback calls `encap_send_reply()` with the value of the global *replyto* variable, and the contents of the text output buffer. This sends a response message to the initial request message, which is what the messaging system expects. In effect, there are three events involved in the proper servicing of a message based request to this application:

- the incoming message event, which causes a command to be sent to the subprocess
- the application event generated by receiving the result of the query
- the application event generated by the `Command?` prompt

This first event handler has to exit back to the main loop in order for the next event, the output from the application, to be sent to the text output window. Again, this event handler has to exit before the third event, the application event triggered by the `Command?` prompt, to take the data from the text output window and return it as the reply value to the initial message.

For example, you may wish to send notification messages in the callback for a specific application event:

```
/* receive_msg
    *

 * This callback procedure is invoked when the

* encapsulation receives a request message. The
   requested

* action and any accompanying data are parsed from
   the

* message. Then the appropriate command is sent to
   the

* subprocess via send_phone_command. The message
   object

* is notified that a reply message will be sent
   later.
   */



static void receive_msg(encap_object
   obj,encap_instance inst, void*
   data){encap_string inst_action =
   encap_get_instance_action(inst);encap_string
   inst_data =

encap_get_instance_data(inst);reply =
   encap_get_instance_replyto(inst);send_phone_co
   mmand(inst_action,
```

```
   inst_data);encap_send_reply(msg_obj, reply,
   "REPLYLATER",

encap_NULL);
   }
```

Define a Function to Create a Message Object

With functions defined to handle the various message events, you are ready to define a function to create the message object:

```
/* set_up_messages *

 * This function is used to create a message
   object and to


* register message event (service request)
   handlers.


* Finally, the message connection is started.



 */
   void set_up_messages(void)
   {


#define mer(x)
   encap_make_event(encap_ServiceRequest, \x,
   receive_msg, 0)#define meq(x)
   encap_make_event(encap_ServiceRequest, \x,
   stop, 0)

int i;encap_event event[6];

/* Make the request events and add them to the
   application.*/

msg_obj = encap_make_object(encap_NULL,


"msg", encap_Message, encap_NULL,
   encap_TOOLCLASS("PHONE"),       event[0] =
   meq("STOP"),      event[1] = mer("Print"),
   event[2] = mer("GetByLastName"),      event[3]
```

```
   = mer("GetByFirstName"),       event[4] =
   mer("GetByNumber"),       event[5] =
   mer("GetByAreaCode"), 0);
for(i=0; i<6;
   i++)encap_free_event(event[i]);encap_start_mes
   sage_connection(msg_obj);return;
}
```

Step #6: Initializing the Encapsulation from the GUI

For convenience, write a function that will initialize the subprocess object and the message object, as well as keep track of the GUI so that the appropriate methods may be called. This function can also perform any other checking that may be required to ensure that the application will run successfully.

```
/* set_up_encapsulation
    *

* This function is used to set up message and
   application

* events for the subprocess, and to start the
   subprocess.

* The swidget passed in is used to invoke methods
   for

* displaying back to the user.



 */
   void set_up_encapsulation(swidget gui)
   {

char *cmd = "phone_app";char *dat =
   "phone_app.book";
```

```
if(0 != access(cmd, X_OK)) {

fprintf(stderr, "Cannot find executable: %s.\n",
   cmd);

exit(1);

}

if(0 != access(dat, R_OK)) {

fprintf(stderr, "Cannot find data file: %s.\n",
   dat);

exit(1);

}

phone = gui;

set_up_messages();

set_up_application_events(cmd);
   }
```

Next, arrange for this initialization function to be called from the GUI. A convenient place is in the explicit loop of the main program. Using the Program Layout Editor, edit the explicit loop to add the call. Be sure to include the header file to define the function.

```
/*-----------------------------------------

* The event loop used to make SoftBench
   encapsultions.
   *---------------------------------------*/
#ifdef _NO_PROTOvoid EncapInit();void
   EncapEventLoop();
```

```
#elsevoid EncapInit( int *argc, char
   *argv[]);void EncapEventLoop(void);
#endif /* _NO_PROTO */

#include "encap_phone.h"

EncapInit(&argc, argv);
```

Step #7: Writing Files and Compiling the Application

1. Select Options⇒Code Generation... and set ANSI C.
2. In the Program Layout Editor, set the Application Class to Phone.
3. In the Program Layout Editor, edit the Makefile to compile the necessary extra files:

   `APPL_OBJS = $PJ_SPECIAL_MODULES` *`encap_object.o`*

Note: *encap_object.o* is the object file for the source that will contain your encapsulated application.

---

4. In the Program Layout Editor, press Apply.
5. Select File⇒Save Project As and save the project.
6. Select File⇒Generate Code As, click on Run Makefile, and click OK.

Conclusion

To recap how to write an encapsulation with UIM/X:

- Build the application interface, including methods.
- Set up the main event loop for Encapsulator events by selecting Explicit Loop and by adding the encapsulator macros in the Makefile.
- Create the subprocess object, defining functions to send data to the subprocess and setting up handlers for application events.
- Integrate the GUI with the subprocess object by writing callback functions.
- Add messaging to the application by creating a message object and defining handlers for message events.
- Initialize the encapsulation from the GUI by invoking the functions to create the subprocess object and the message object. This can be done in the Explicit Loop of the main program.
- Write the project and interface files, and compile the application.

Messages Reference

A

Summary of Messages

Requests Accepted

Requests Accepted by UIM/X	Replies Sent
R UIBUILD ICONIFY	N / F UIBUILD ICONIFY
R UIBUILD IMPORT-UIL	N / F UIBUILD IMPORT-UIL
R UIBUILD LOAD-UIFILE	N / F UIBUILD LOAD-UIFILE
R UIBUILD NORMALIZE	N / F UIBUILD NORMALIZE
R UIBUILD SET-CONTEXT	N / F UIBUILD SET-CONTEXT
R UIBUILD STATUS	N UIBUILD STATUS
R UIBUILD STOP	N UIBUILD STOP

Requests Sent

Requests Sent by UIM/X	Replies Accepted
R BUILD MAKEFILE-BUILD	N / F BUILD MAKEFILE-BUILD
R BUILD SET-MAKEFILE-NAME	F BUILD SET-MAKEFILE-NAME
R BUILD STOP	Ignored
R CM VERSION-CHECK-IN	N / F CM VERSION-CHECK-IN
R CM VERSION-CHECKOUT - - CO-	N / F CM VERSION-CHECK-OUT - - CO-
R CM VERSION-CHECKOUT - - CO - LOCK-	N / F CM VERSION-CHECK-OUT- - CO - -LOCK-
R EDIT STOP	Ignored
R EDIT WINDOW	N / F EDIT WINDOW
R TERM KILL-ALL	N TERM KILL-ALL
R TERM NO-STDIO	N / F TERM NO-STDIO
R TERM STOP	N TERM USER-MESSAGE

Notify and Failure Messages Accepted

N/F Messages Accepted by UIM/X (as a result of successful request to other tools)
N / F BUILD BUILD-TARGET
N BUILD STOP
N EDIT FILE-MODIFIED
N EDIT STOP

Notify and Failure Messages Sent

N/F Messages Sent by UIM/X
N UIBUILD FILE-MODIFIED
N UIBUILD SET-CONTEXT
N UIBUILD STOP
N UIBUILD STATUS

Request Messages Accepted by UIM/X

The REQUEST messages in this section are accepted by UIM/X from other SoftBench Tools.

Note: All messages include the *host*, *dir*, and *operand* context parameters. They are described in the Parameters and Reply sections below only when they are of primary importance.

R UIBUILD ICONIFY *host dir operand*

Description: Request to iconify all open UIM/X windows.

Parameters: Not used by UIM/X.

Action: Iconifies all opened UIM/X windows.All tools initiated by UIM/X are also iconified, using their own icons.

Reply: N UIBUILD ICONIFY The operation was successful.
F UIBUILD ICONIFY The operation was not successful.

R UIBUILD IMPORT-UIL *host dir operand*

Description: Request to import a Motif UIL file into UIM/X and create the necessary interface files.

Parameters:

dir:	The directory path.
operand:	The file name, including the path and the extension `.uil`. The path is absolute or relative to the *dir* parameter.

Action: Converts the UIL file into UIM/X format and loads the converted files.

Reply: N UIBUILD IMPORT-UIL *dir operand* The file has been imported successfully.
F UIBUILD IMPORT-UIL *dir operand error_message* The file has not been imported.

R UIBUILD LOAD-UIFILE *host dir operand*

Description: Request to load a UIM/X format file.

Parameters:

dir: The context path.

operand: The file name. The absolute path or the path relative to the *dir* parameter.

Action: Loads a project, interface, or palette into UIM/X. Errors are displayed in the Messages Area.

When loading a project file into a UIM/X session that already contains a project, all SoftBench tools that were opened from this session are sent a STOP request and UIM/X loads in the new project.

Before loading a new project, UIM/X displays a warning dialog box, allowing you to cancel the request.

Reply: N UIBUILD LOAD-UIFILE *dir operand* The file has been loaded successfully. Note that a successfully loaded file does not imply that there are no errors in the file.

F UIBUILD LOAD-UIFILE *dir operand error_message* The file has not been loaded. The file will be loaded only if the file header is valid.

R UIBUILD NORMALIZE *host dir operand*

Description: Request to open all UIM/X windows previously iconified via SoftBench messages.

Parameters: Not used by UIM/X.

Action: Opens all UIM/X windows that were iconified by an ICONIFY request. All tools that were initiated by UIM/X are also normalized.

Reply: NUIBUILD NORMALIZE

The operation was successful.

F UIBUILD NORMALIZE

The operation was not successful.

R UIBUILD SET-CONTEXT *host dir operand newhost newdir newoperand*

Description: Request to change the UIM/X context.

Parameters:

newhost:	host (required, but ignored by UIM/X)
newdir:	directory
newoperand:	project file

Action: UIM/X changes current directory to *newdir* and loads *newoperand* if and only if the operand is a project file.

If the new project file is replacing an existing one, UIM/X sends a STOP request to all tools that were called from UIM/X. UIM/X displays a warning dialog box allowing you to cancel the request.

Note: UIM/X does not support remote hosts. Therefore, while the context host can be set, it is ignored by UIM/X.

Reply: N UIBUILD SET-CONTEXT *host dir operand newhost newdir newoperand*

The context has been successfully set.

F UIBUILD SET-CONTEXT *current_host current_dir current_operand error_message*

The context change has been unsuccessful. The request returns the current context.

R UIBUILD STATUS *host dir operand*

Description: Request to return the current status of UIM/X.

Parameters: Not used by UIM/X.

Action: Not Applicable.

Reply: N UIBUILD STATUS *status* The value of *status* can be READY or BUSY.

R UIBUILD STOP *host dir operand*

Description: Request to exit UIM/X.

Parameters: Not used by UIM/X.

Action: Commences the SHUTDOWN procedure. All SoftBench tools started by UIM/X are also closed.

Reply: N UIBUILD STOP
The shutdown procedure has been completed.

Request Messages Sent by UIM/X

The REQUEST messages in this section are sent by UIM/X to other SoftBench Tools.

Note: All messages include the *host*, *dir*, and *operand* parameters. They are listed in the Parameters and Expected Reply sections below only when they are of primary importance.

R BUILD MAKEFILE-BUILD *host dir operand*

Description:

Request to BUILD tool to run `make` using *operand* as the name of the makefile. The action is executed when you select the Run Makefile option in the Generate Code dialog box.

The status, given by the reply message, is displayed in the Messages Area.

Parameters:

dir: The directory containing the makefile.

operand: The makefile name, relative to *dir*.

Expected Reply:

N BUILD MAKEFILE-BUILD *operand* The BUILD tool finished the make successfully.

F BUILD MAKEFILE-BUILD *operand error_message* The make terminated with an error.

Related Messages:

N/F BUILD BUILD-TARGET

R BUILD SET-MAKEFILE-NAME

R BUILD STOP

R BUILD SET-MAKEFILE-NAME *host dir operand*

Description:

Request to BUILD tool to set the name of the makefile. If the command fails in the BUILD tool, you will not be able to perform subsequent builds within the BUILD tool because the makefile name is not set to the correct file name. To rectify this problem manually, set the makefile through the BUILD tool's user interface.

Parameters:

dir: The directory containing the makefile.

operand: The name of the makefile, relative to *dir*.

Expected Reply: F BUILD SET-MAKEFILE-NAME *operand error_message*

The BUILD tool failed to change the makefile. The *error_message* is displayed in the Messages Area.

Note: UIM/X does not listen for N BUILD SET-MAKEFILE-NAME message because no action will be taken as a result of the reply.

Related Messages: R BUILD MAKEFILE-BUILD

N/F BUILD BUILD-TARGET

R BUILD STOP *host dir operand*

Description: Request to exit the BUILD tool. This request is sent when UIM/X exits or a user loads another project file.

Parameters:

dir**:** The directory containing the makefile.

operand**:** The name of the makefile, relative to *dir***.**

Expected Reply: The reply is ignored.

Related Messages: R BUILD MAKEFILE-BUILD

R CM VERSION-CHECK-IN *host dir operand*

Description: Request to Configuration Manager tool to check in a *file* already under version control. The file must already be checked out and locked by you.

To check in a project file, select Check In from the Version Control submenu in the File pulldown menu in the Project Window.

To check in an interface, select Check In from the Version Control submenu in either the Selected Interfaces popup menu in the Project Window, or the File pulldown menu in the Browser.

To check in a palette, select Check In from the Version Control submenu in either the Selected Palettes popup menu in the Project Window, or the File pulldown menu in the Palette.

The Check In item in the Version Control submenu in the Tools pulldown menu in the Startup Interface brings up a file selection box where you can enter the file name to be checked-in.

The status, given by the reply message, is displayed in the Messages Area.

Parameters:

operand**:** The file name, including the full path.

Expected Reply:

N CM VERSION-CHECK-IN *operand message* The file was placed under version control successfully.

F CM VERSION-CHECK-IN *operand error_message* The file was not placed under version control.

Related Messages:

R CM VERSION-CHECK-OUT - - CO -

R CM VERSION-CHECK-OUT - - CO - LOCK -

R CM VERSION-CHECK-OUT *host dir operand* "- - CO -"

Description:

Request to the Configuration Management tool to copy the latest version of *operand* under version control into current directory—that is, the file is not locked.

To check out a project file, select Check Out from the Version Control submenu in the File pulldown menu in the Project Window.

To check out an interface, select Check Out from the Version Control submenu in either the Selected Interfaces popup menu in the Project Window, or the File pulldown menu in the Browser.

To check out a palette, select Check Out from the Version Control submenu in either the Selected Palettes popup menu in the Project Window, or the File pulldown menu in the Palette.

The Check Out item in the Version Control submenu in the Tools pulldown menu in the Startup Interface brings up a file selection box where you can enter the file name to be checked-out.

The status, given by the reply message, is displayed in the Messages Area.

Parameters:

operand**:** The file name, including full path.

Expected Reply:

N CM VERSION-CHECK-OUT *operand message* The file was checked out successfully.

F CM VERSION-CHECK-OUT *operand error_message* The file was not checked out.

Related Messages:

R CM VERSION-CHECK-IN

R CM VERSION-CHECK-OUT - - CO - LOCK -

R CM VERSION-CHECK-OUT *host dir operand* "- - CO - LOCK -"

Description: Request to the Configuration Management tool to copy the latest version of file under version control into the current directory, locking the operand to prevent other users from checking out and locking the same file.

To check out and lock a project file, select Check Out & Lock from the Version Control submenu in the File pulldown menu in the Project Window.

To check out and lock an interface, select Check Out & Lock from the Version Control submenu in either the Selected Interfaces popup menu in the Project Window, or the File pulldown menu in the Browser.

To check out and lock a palette, select Check Out & Lock from the Version Control submenu in either the Selected Palettes popup menu in the Project Window, or the File pulldown menu in the Palette.

The Check Out & Lock item in the Version Control submenu in the Tools pulldown menu in the Startup Interface brings up a file selection box where you can enter the file name to be checked-out and locked.

The status, given by the reply message, is displayed in the Messages Area.

Parameters: operand**:** The file name, including full path.

Expected Reply: N CM VERSION-CHECK-OUT *file message* The file was checked out successfully.

F CM VERSION-CHECK-OUT *file error_message* The file was not checked out.

Related Messages: R CM VERSION-CHECK-IN

R CM VERSION-CHECK-OUT - - CO -

R EDIT STOP *host dir operand*

Description: Request to exit the EDIT tool. This action is executed when a user closes or modifies the context of the UIM/X editor, loads another project file, or exits UIM/X.

For example, selecting a new item in the Menu Editor changes the widget context within the editor. This will close the edit tool window.

Parameters: operand: Generated temporary file, including the full path.

Expected Reply: The reply is ignored.

Related Messages: R EDIT WINDOW

R EDIT WINDOW *host dir operand*

Description: Request to edit the contents of a text widget. The user selects UIM/X's edit button (…). UIM/X writes the contents of the text widget to a temporary *file* and sends a request to the EDIT tool to edit the *file*. (When UIM/X is not running in the SoftBench environment, this normally brings up UIM/X's own text editor.)

The file is created in the current directory and is given the prefix `.UxSb` and a unique identifier. The name of the file indicates the source of the text. A typical name, for example, is the following:

```
.UxSbCAAa24372_Declarations_Editor_Instance_specific_variables
```

Parameters: operand: The file to be edited, including the full path.

Expected Reply: N EDIT WINDOW *operand* The tool successfully loaded the *file*.

F EDIT WINDOW *operand error_message* The tool did not load the *file*. The *error_message* is displayed in the Messages Area.

Related Messages: N EDIT FILE-MODIFIED

R EDIT STOP

R TERM KILL-ALL *host dir operand terminal-id* <how to kill>

Description: UIM/X sends this Request to kill the stand alone application process executed from Run Mode and Run Executable. By default a SIGTERM signal will be sent the process instructing it to terminate. If the string ''KILL'' is used in the <how-to-kill> field, a SIGKILL will be used instead.

Parameters: host, dir operand: Ignored and have no impact on the message.

terminal-id: This is the terminal-id field returned in the Notify reply from TERMINAL, NO-STDIO, SHELL-TERMINAL and DBOX-TERMINAL messages.

how-to-kill: This option may be omitted to have a SIGTERM signal used to kill the specified terminal emulator or the string KILL may be used to have s SIGKILL used instead.

Expected Reply: N TERM KILL-ALL.

Related Messages:

R TERM NO-STDIO
R TERM STOP
N/F TERM NO-STDIO
N TERM KILL-ALL
N TERM USER-MESSAGE

R TERM NO-STDIO *host dir operand exec-args exec-host program args*

Description:

UIM/X sends this Request to start up a terminal server (without a window) from Run Mode or Run Executable to run the built stand alone application. This is useful for running programs that create their own windows (e.g. X11 programs).

Parameters:

host:	Not used by UIM/X.
dir	Not used by UIM/X.
operand:	Not used by UIM/X.
exec-args:	This is a comma separated list of arguments to be passed onto the terminal emulator. No blanks are allowed.
exec-host:	The host where the terminal emulator window will be created. The type of emulator window (for example, hpterm, xterm, or shelltool) is determined by identifying the type of machine exec-host is and looking up the appropriate entry in the softtermsrv configuration file.
program:	The program to execute.
arg:	The arguments to pass to the program being executed (if any).

Expected Reply:

N TERM NO-STDIO
F TERM NO-STDIO

Related Messages:

R TERM KILL-ALL
R TERM STOP N TERM KILL-ALL
N TERM USER-MESSAGE

R TERM STOP *host dir operand*

Description:

UIM/X send this Request to stop the terminal server that is running the executable.

Parameters: file**:** Not used by UIM/X.

Expected Reply: The reply is ignored.

Related Messages:
R TERM KILL-ALL
R TERM NO-STDIO
R TERM STOP
N/F TERM NO-STDIO
N TERM USER-MESSAGE

Notify and Failure Messages Accepted by UIM/X

The NOTIFY messages in this section are accepted by UIM/X from other SoftBench Tools, as a result of successfully starting the tools.

Note: All messages include the *host*, *dir*, and *operand* parameters. They are described in the Parameters section below only when they are of primary importance.

N/F BUILD BUILD-TARGET *host dir operand error_message*

Description: Notification/Failure received from BUILD tool indicating the user successfully/unsuccessfully completed a build through the Tools interface. This message is accepted only if the BUILD tool has been called.

Parameters:
operand: The name of the Makefile originally sent in the REQUEST to SETMAKEFILE-NAME in the BUILD tool.

Action: Displays message indicating success/failure in the Messages Area.

Related Messages:
R BUILD MAKEFILE-BUILD
R BUILD SET-MAKEFILE-NAME

N BUILD STOP *host dir operand error_message*

Description: Notification sent by a BUILD tool that it has been closed.

Parameters: Not used by UIM/X.

Action: Display a message in Messages Area. All BUILD tool messages are unregistered with the BMS.

Related Messages: R BUILD MAKEFILE-BUILD

N EDIT FILE-MODIFIED *host dir operand*

Description: When a temporary UIM/X *file* is saved by an EDIT tool, notification is broadcast indicating that *file* has been saved by that tool.

Parameters:
operand: The name of modified file-contains the absolute path.

Action: Loads the contents of the file into the text widget from which the text was originally taken.

Related Messages: R EDIT WINDOW

N EDIT STOP *host dir operand*

Description: Notification sent by an EDIT tool that it has been closed.

Parameters: Not used by UIM/X.

Action: The temporary file is deleted and the text widget from which the EDIT tool was started is made sensitive. All EDIT messages with the same context are unregistered with the BMS.

Related Messages: R EDIT WINDOW

Notify and Failure Messages Sent by UIM/X

The NOTIFY and FAILURE messages in this section are broadcast by UIM/X to other SoftBench Tools.

Note: All messages include the *host*, *dir*, and *operand* parameters. They are listed in the Parameters area below only when they are of primary importance.

N UIBUILD FILE-MODIFIED *host dir operand*

Description: Broadcast notification that UIM/X has written a file to disk. This can occur in the following situations:

- A project, interface, or palette has been saved.
- While generating code-that is, while the interface source, header, main, makefile, UIL, or `.rf` files are generated.

Parameters:

dir**:** The absolute directory path where the file was saved.

operand: The name of the saved file, including full path.

N UIBUILD SET-CONTEXT *host dir operand*

Description: Broadcast notification that UIM/X has changed context. This can occur if the current directory has been changed through the Current Directory in the Options pulldown.

Parameters:

dir**:** The absolute directory path where the file was saved.

operand: The name of the saved file, including full path.

Note: UIM/X does not support remote hosts. Therefore, the context host will always be the hostname on which UIM/X is executing.

N UIBUILD STATUS *host dir operand*

Description: Notification sent by UIM/X of its own status.

N UIBUILD STOP *host dir operand*

Description: Notification sent by UIM/X that it is shutting down. This occurs when selecting Exit from the File pulldown in the menu bar of the startup interface.

Parameters:

host	The current context host.
dir**:**	The current context directory.
operand:	The current context operand.

A Notify and Failure Messages Sent by UIM/X

Error Messages

B

Overview

The following SoftBench-related error messages are displayed in the Messages Area.

Error	Error Message	Description
206	SoftBench tool (*tool_name*) - error occurred while executing *command*: *returned_error_message*	An error occurred while executing the specified command in the called tool.
266	Failed to call SoftBench tool (*tool_name*)	The tool is not configured inSoftBench.
267	Failed to generate temporary file name for SoftBench tool (*tool_name*)	Problem creating a file in the current directory. Verify permissions.
268	Failed to open *file_name*	The file either does not exist or permissions do not allow the user to open the file.
269	Failed to write to *file_name*	This is an internal error. UIM / X cannot write a temporary file in the context directory.
270	Failed to read *file_name*	This is an internal error. UIM / X cannot open a temporary file in the context directory.
757	SoftBench Error: Unknown message type used to display returned message from tool. Returned Message: *SoftBench_message*	Invalid input parameter used when calling `UxSbDisplayMsg()`
760	SoftBench command line argument - host does not support remote hosts. Setting context to (current_host).	The host context will be set to the host name of the machine on which UIM / X is running.

Note: Any strings in the data field of a Notify or Failure message will also be displayed.

Note: Text set in italics—*file_name*, *tool_name*, *command*, *returned_error_message*, *current_host* and *SoftBench_message*—is replaced with the relevant information.

Index

Index

S

T

U

V

W

Index