Remote Procedure Calls I. Sun RPC

Francisco Moya Fernndez

One of the most useful abstractions for a software developer is the procedure. The expressive power of a set of procedures in some programming languages (ML, CAML, Lisp, Scheme, etc.) can even emulate any other commonly used feature such as local variables, composite types, abstract data types, dynamic dispatching and object orientation support. As we shall see in the following pages, it is also a useful concept to encapsulate data communication operations between several independent processes in a convenient way.

A pair of communicating processes is usually described following a client/server model. One process called the *server* waits for another, (the *client*), to initiate the communication by issuing a request. That request is processed by the server which usually sends a report back to the client.

This model is very easy to map to a procedure. The client is the piece of code that invokes the procedure and the server is the implementation of the procedure. It is that simple!

The objective of RPCs is to offer a set of tools and libraries that turn inter-process communication operations into simple procedure calls. The real implementation of the procedure must be executed by the server (possibly in another computer) and the code that invokes the procedure must be implemented by the client.

As we said communication operations may involve different computers. That tiny detail is the source of some other new difficulties. The RPC system must be careful when selecting the proper byte ordering and word size of both computers involved in each remote procedure call. This is not a trivial problem. Let's illustrate that with an example. Consider the following fragment of code:

Server side

```
int neg(int i)
{
   if(i<0) return 1;
   else return 0;
}
```

Client side

```
void client()
{
   /* ... */
   if(neg(-1)) printf("OK\n");
   /* ... */
}
```

Let's consider the case of a client running in a 32 bit machine and a server running in a 64 bit machine. In this situation, the program would not work as expected, even if the RPC system took care of the byte ordering. The reason is simple: int is a completely different type in both computers. It is even different in size!

This problem is easily overcome by using a different language to specify the interface between communicating processes. This language will have a complete definition of data types and even a specified byte ordering. But don't worry, it is usually a fairly simple language.

1 Your first RPC application

Ok, enough theory, let's do something useful. We will start with the well-known *hello world* example, however,this time with the distributed version. We will use the most common variety of RPCs in the free software community, Sun RPC (also known as ONC RPC), the basis of NFS (Network File System) and some other important utilities. Any GNU/Linux distribution includes all the necessary tools.

1.1 What do you need?

GNU libc based systems (all the variants of GNU/Linux included) do not require any additional library since the libc system already provides all the Sun RPC functions. You just need to look for the utility rpcgen. Debian includes it in the netbase package, which is a standard package, that you have probably already installed.

You need a full set of normal development tools (an editor, the GNU C compiler and GNU make). We assume you have already made some C programs under some variety of Unix.

1.2 The protocol description file

A normal *hello world* looks like this:

```
#include <stdio.h>
int main()
{
   puts("Hello World!\n");
}
```

We will make a remote version of puts, which we'll call `hello_puts`, that just displays the text in the argument in the remote terminal.

The interface for Sun RPCs is described in a language called XDR (eXternal Data Representation). If you know C then you can learn all the features of XDR in a few minutes by just looking at some examples in `/usr/include/rpcsvc/*.x`, or you might want to look at our notes on XDR.

The protocol (i.e. the interface) for the hello world example is trivial (file `src/hello/hello.x`):

```
program HELLO {
   version HELLO_VERS_1 {
      void HELLO_NULL(void) = 0;
      int HELLO_PUTS(string text<>) = 1;
   } = 1;
} = 400000;
```

Each protocol is called a *program* and may contain several versions of the protocol (a server may be interested in servicing several versions simultaneously for backward compatibility reasons). Each program has a unique associated number (in this case 400000) that must be supplied by the user. A list with program numbers assigned to standard RPC services is usually found in the file `/etc/rpc`. At any given time you may get a listing of the numbers associated with the servers currently running with the command `rpcinfo -p`.

In the above example, we have declared the program `HELLO` with the number 400000 and a single protocol version (number 1) with two remote procedures: `HELLO_NULL` and `HELLO_PUTS` with procedure numbers 0 and 1 respectively (which must be unique for each protocol version). By convention the first procedure should always be a `NULL` procedure, which does nothing. It can be used to find out if a service is available without actually calling any remote procedure (using rpcinfo). Uppercase letters are also used by convention. As you will see later lowercase versions of some symbols will be automatically generated, so it is better not to mix cases in the specification.

The argument of `HELLO_PUTS` is in string type. This is one of the special types provided by XDR language. Unfortunately there aren't any free manual pages for these XDR types, but you will learn a lot by just looking at a more complete example or the specifications provided with your system (`/usr/include/rpcsvc/*.x`). The angle brackets describe the maximum length of the string (in this case unspecified, that is to say, unlimited). The return type is an integer. However, you should be aware that this is not exactly the same as a C int. XDR specifies that this is a 32 bit signed integer on the wire, and so in a 64 bit machine the XDR int may be mapped to a C short or it could also be translated to a 64 bit int if the library code takes care of the appropriate conversions. Don't worry, all of the conversions are automatically generated.

1.3 Code generation

It's time to use rpcgen to generate everything related to automatic communication. Use the following command:

```
rpcgen -N -a hello.x
```

Now you have a bunch of new files:

`hello.h`	Type and function declarations
`hello_svc.c`	Server side stubs
`hello_clnt.c`	Client side stubs
`hello_server.c`	Sample server implementation
`hello_client.c`	Sample client implementation
`hello_xdr.c`	C/XDR type conversion and streaming

Well, in this case `hello_xdr.c` will be missing because we have used standard XDR types and all of the conversion routines are already defined in the GNU C Library.

The first thing you should do is forget about all of the stubs and XDR conversion routines (`*_svc.c`, `*_clnt.c`, `*_xdr.c`). You will not need to modify them. They are only needed for the RPC subsystem.

So we only have to modify the sample implementations `hello_server.c` and `hello_client.c`. These two files are only generated if you specify the `-a` switch to rpcgen. Take a look at them.

1.4 Tweaking the client

The client `hello_client.c` just calls the two procedures declared in the protocol in the correct order with bogus arguments. This is just an example to show you how to set up the communication channel with the server and how to perform a procedure call. Modify it to do just what we want (delete everything that is not needed), as in the example below:

```
CLIENT *clnt;
clnt = clnt_create(host, HELLO, HELLO_VERS_1, "tcp");
if (clnt == NULL) {
   clnt_pcreateerror(host);
   exit(1);
}
hello_puts_1("Hello, World\n", clnt);
clnt_destroy( clnt );
```

As you can see, there is very little added, most of the modifications are deletions. We will proceed to describe it step by step:

1. Create a client structure that takes care of the communications with the server. You need to specify the host running the server, the RPC protocol and version (as given in the XDR specifications), and the transport protocol. This can be either “udp” or “tcp”. Please, if you are a beginner use “tcp” (it is not the default!) and you can then safely ignore some subtleties.

2. Call the remote procedure (note that it is lowercase) or set of procedures. The remote procedure has the version number attached so you can use any version available and don't forget that all of the remote procedures have an additional argument specifying the client structure. The return value for the procedure call is not exactly what was specified in the XDR file, but it is a pointer to that. This allows us to identify an error condition in the communication path (the pointer is then `NULL`, even if the procedure was declared void).

3. Destroy the client structure to release internally used resources.

You may use the same client structure during the whole program execution and perform any number of remote procedure calls before destroying the structure. You may also even have several client structures of the same protocol (for example, use several versions of the protocol at the same time for testing purposes).

1.5 Tweaking the server

Now edit `hello_server.c` to implement the guts of the remote procedures. As you can see it is quite unimportant to know where to place our puts. Don't touch the implementation for the NULL procedure. It's ok and almost empty. The implementation for `hello_puts` should be similar to this:

```
#include <stdio.h>
int *
hello_puts_1_svc(char *text,  struct svc_req *rqstp)
{
   static int  result;
   result = puts(text);
   return(eturn(&result);
}
```

The suffix `_svc` is added automatically to allow programs to act as client and server simultaneously. As you can see it returns a pointer to the type specified in the XDR file (similar to that on the client side). Be careful not to return a pointer to an automatic variable. `NULL` is used to signal an error condition. For this simple minimal example we just added an include directive and the puts call.

As in the client's case, we have an extra argument. This time it is an identifier for the request, with for example, full information about who made the RPC.

1.6 Solving the puzzle

The last thing we must do is to put all the pieces together with a minimal `Makefile`:

```
all: hello_client hello_server
hello_client: hello_client.o hello_clnt.o
hello_server: hello_server.o hello_svc.o
```

and issue a simple make command to generate both executables. The server is executed without command line arguments and the client needs the name of the host in which the server is running (e.g. localhost). Easy, isn't it? Ok, I agree that it is a bit more difficult than just calling a procedure, but compare it with any other IPC mechanism. The RPC subsystem takes care of most of the details for you.

2 More fun with Sun RPC and GTK

You don't seem to be impressed by the results... ok, let's do something more visually attractive. We'll make a tool similar to that message window popper available on MS systems. That involves enough material to keep you busy for a little while longer.

We want to make a server that waits until someone sends a message (hey! but that is almost the same as before!). When the message arrives, a nice pop-up window will show you the contents (ok, we won't use a simple puts, but the idea is the same). On the other side, the client is also similar but with a nice window interface instead of a command line interface.

Now we face an important design decision... which widget toolkit are we going to use? There are a lot of free GUI toolkits out there, and some are quite nice... Ok, let me decide for you, let's use GTK (GIMP toolkit). Why? It is really fast, has low memory footprint, supports dozens of languages, is completely free, and is used by one of the most exciting projects undertaken by the free software community (GNOME). From the technical point of view, we can say that GTK is extremely well written, very complete, very well maintained and has lots of activity around it.

No matter what GUI toolkit we choose, it is always a headache to write down all of the code lines required to build the user interface. Fortunately we have a very nice GUI builder to simplify the task, Glade[1]. Be careful! It is alpha software, and so is incomplete and a bit unstable. (Yep! That means it's a lot more fun!) Don't be nervous, we are using it to build skeletons for quite complex applications and even in its present state we consider Glade to be a great help. Figures 1 and 2 show the user interface of Glade.

Your first homework assignment is to install Glade and build a user interface for the client and for the server. Something like figures 3 and 4.

Ok, ok, we know Glade comes with no documentation... Take a look at these Glade hints and come back when you are ready.

2.1 Adding RPC support to GTK

It was easy, wasn't it? But now we face the main difficulty. Take a look at the server code generated by Glade and the server code of the hello world example. Pay attention to the program flow. The server main function of the hello world example is in `hello_svc.c` and the main function for the Glade generated code is in `gladesig.c`.

The hello world example enters a function called `svc_run` and nothing more. The server main loop is implemented in this library function and the GTK applications call `gtk_main` that also implements the main loop of a GUI application. Now we need to call both of them at the same time... Most of you are already considering threads, aren't you? Ok, that could be a solution, and we will devote an entire article to that topic in the near future. But multi-thread applications are a lot harder to develop. It seems to be a bit of an overkill...

Fortunately GTK is very well thought out. It provides mechanisms to incorporate new event sources in the main event loop processing. The problem is that Sun RPC API is brain damaged and there is no portable and efficient way to know when a new event source (a new connection from a client) has been incorporated. We will return to this topic in a later article to show you how easily this can be solved. Unfortunately Sun RPC is a very widespread system (even in the proprietary software world) and that makes it a lot harder to introduce changes in the API.

The solution is a compromise between efficiency and portability (a polled implementation). At regular intervals we will check if there are pending RPCs and if so we will dispatch them using the Sun API. This is the main part of the implementation:

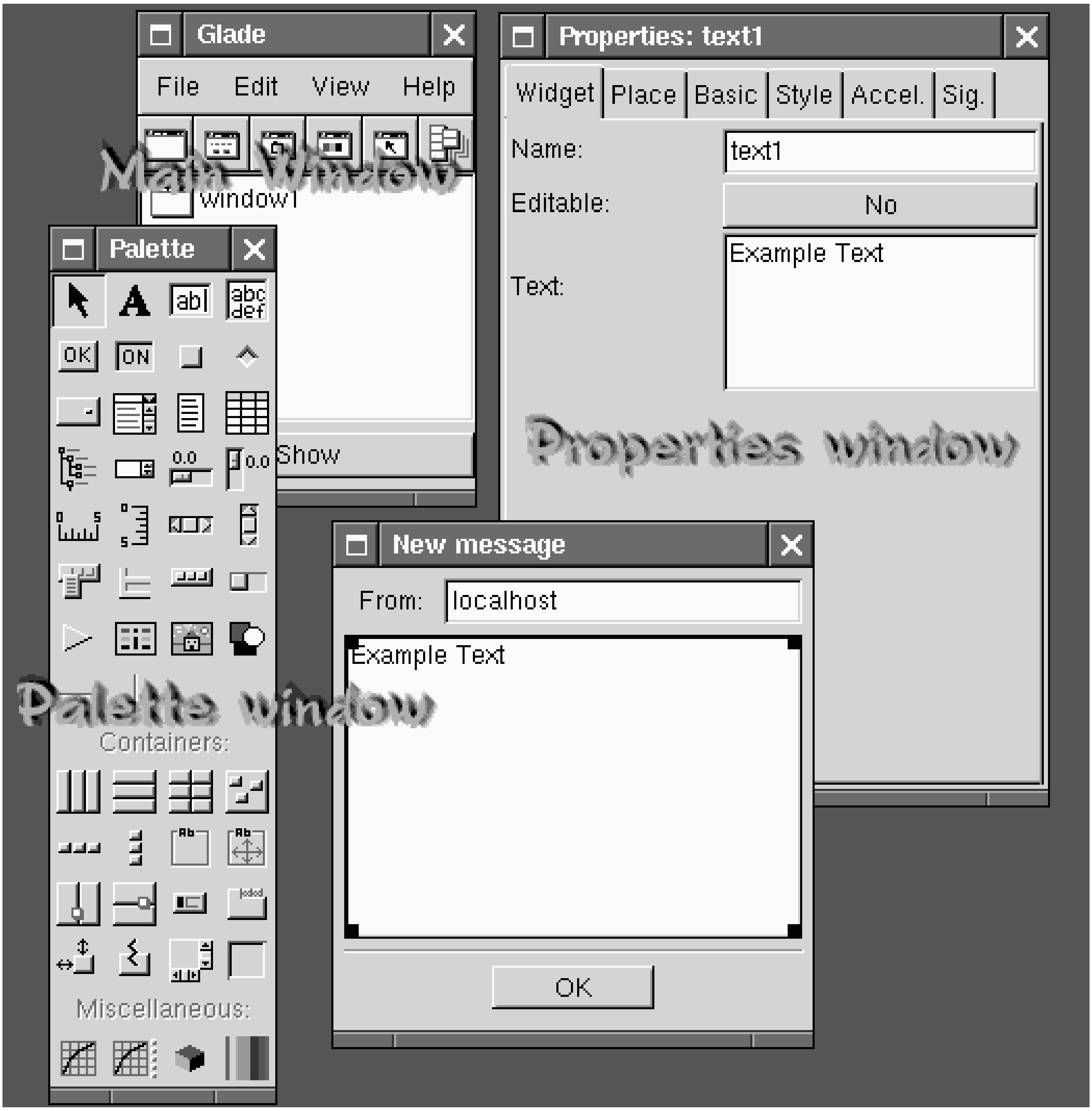
```
void
activate_rpc_services()
{
   SVCXPRT* xprt;

   xprt = svctcp_create(RPC_ANYSOCK, 0, 0);
   pmap_unset(RPCMSG, RPCMSG_VERS_1);
   svc_register(xprt, RPCMSG, RPCMSG_VERS_1, rpcmsg_1, IPPROTO_TCP);
   gtk_timeout_add(200, handle_rpc, NULL);
}
```

If you make a careful examination of the hello world server generated by rpcgen you will find that it is very similar. It is simpler because we are not interested now in the UDP transport implementation. We have just created a TCP transport, registered it in the *portmapper* service (so that you could get information with `rpcinfo~-p` as shown earlier), and registered it in the RPC subsystem. But the main difference is that we didn't call `svc_run`, and instead we installed a GTK periodic event each 200 milliseconds that is handled by the `handle_rpc` function. That function should check if there is any pending RPC. For example:

```
int
handle_rpc(gpointer data)
{
   int ret;
   struct rlimit lim;
   struct timeval tv;
```

---

[1] `http://glade.pn.org/`



Name: text1

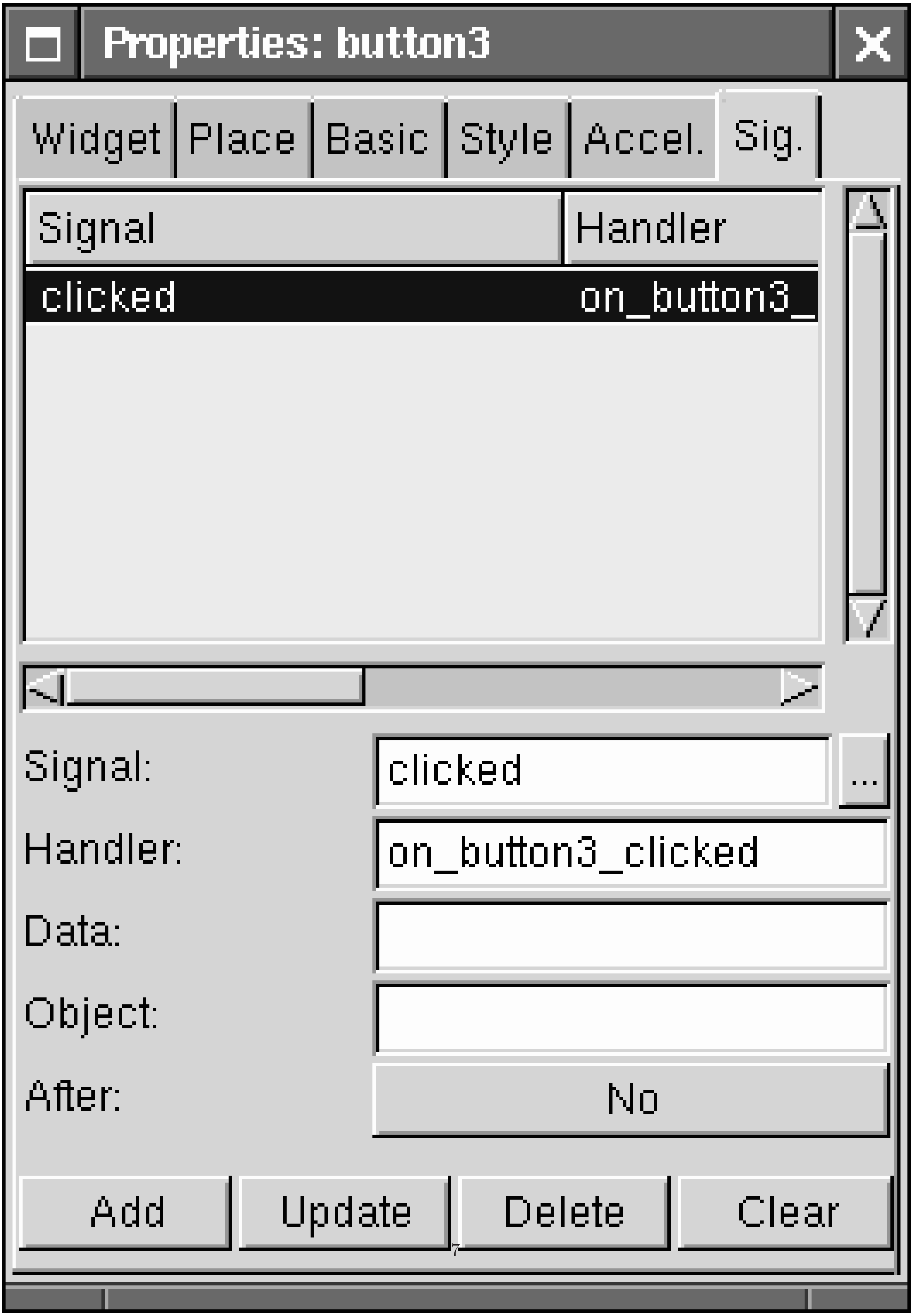
Editable: No

Text: Example Text

From: localhost

Example Text

Figure 1: Working with Glade



Signal	Handler
clicked	on_button3_

Signal: clicked

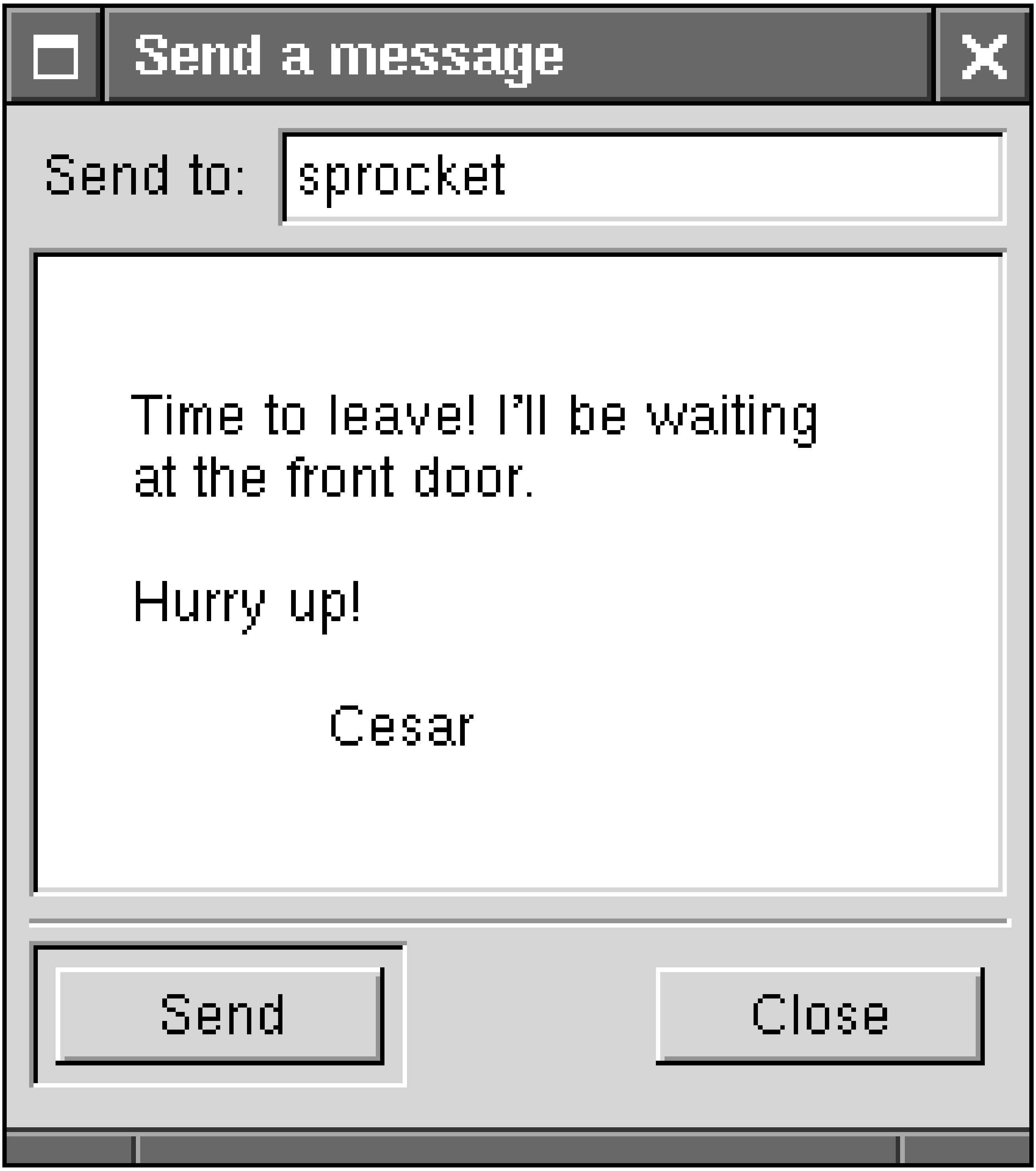
Handler: on_button3_clicked

Data:

Object:

After: No

Figure 2: Setting the properties of a button in Glade



sprocket

Time to leave! I'll be waiting
at the front door.

Hurry up!

Cesar

Figure 3: The client

From: cesar@marvin

Time to leave! I'll be waiting
at the front door.

Hurry up!

Cesar

Figure 4: The server

```
      fd_set tmp_rd_set = svc_fdset;

      tv.tv_usec = 0; tv.tv_sec = 0; /* we are polling, don't wait */
      getrlimit (RLIMIT_NOFILE, &lim);
      ret = select(lim.rlim_cur, &tmp_rd_set, NULL, NULL, &tv);
      if (ret>0)
         svc_getreqset(&tmp_rd_set); /* process pending RPCs */
      return 1; /* always active */
   }
```

All the event sources of the RPC subsystem are in a file descriptor set called `svc_fdset`. File descriptors are a simple abstraction in Unix-like environments to identify any event source (a socket through the network, an opened file, pipes, terminals, a scanner, etc). We will cover all of this in greater detail in a future article. For now, you just need to know that file descriptors represent event sources.

The heart of the polling activity is the select system call. This function is able to know if there is input data pending in a given set of file descriptors. The timeval struct is used to indicate how much time we are able to wait for an event. We indicate a zero time because we are just interested in already pending events (pure polling). We call getrlimit to find the maximum number of file descriptors available for the running process. This is needed because Sun RPC API does not provide any information on the file descriptors already registered in `svc_fdset`. We just assume that there are as many as possible.

If there is input pending, it is processed with the `svc_getreqset` call of the Sun RPC library. Just call it and forget it.

To make this example a bit more interesting we have made a little change to the protocol specification:

```
struct RPCMsg_s {
   string from<>;
   string text<>;
};

typedef struct RPCMsg_s RPCMsg;

program RPCMSG {
   version RPCMSG_VERS_1 {
      void RPCMSG_NULL(void) = 0;
      void RPCMSG_NEW_MSG(RPCMsg msg) = 1;
   } = 1;
} = 400004;
```

Note that we could also have specified two arguments in the second remote procedure, but we chose to define a new type just for fun (now rpcgen also generates a `_xdr.c` file with type conversions). Previous versions of Sun RPC were only able to pass a single argument. This is no longer true whenever you use the `-N` switch with rpcgen (N stands for *new style RPC*).

Now we cannot use `rpcgen -N -a` to generate all of the files at once. For example, the server should not define the main function (we have provided another main loop). That is accomplished by the -m switch to rpcgen, which is incompatible with `-a`. Don't worry, just take a look at the `Makefile.am` for this example (in the source section below) and use it as a template in your future developments.

2.2 Multiple windows

You probably have noticed that the server code should actually open one window each time a remote user sends a message. That is very easy to do. Take another look at the Glade generated code (`gladesrc.c`). There is a function that creates the window you defined earlier. You just need to add arguments to this function for everything that changes (form, and text) and invoke this function each time you get an RPC. Look in the next section to see how we implemented the details.

2.3 Details and polishing

Glade always generates the same set of source files. That means you cannot use the same directory for the client and for the server. Copy the XDR specification file in both of the directories and remember you cannot use the rpcgen from another directory (it is a very awkward tool and would mess all of the include lines).

This is a simple example with some deficiencies. For example, there is no way to send a message to a specific person, it goes instead to a host. This is due to the host wide nature of the RPC program numbers. An interesting project could be to overcome this limitation by using RPCMSG + n as the program number (n being any natural number) and registering any number of servers. You should also provide a system wide RPC server that takes care of the bijective relation between each n and each registered user. The client should now contact the system wide server to ask for the particular n assigned to the target user, and each individual server should register in the system wide server to get an assigned program number.

3 Appendix: XDR Interface Description Language

XDR is very similar to a set of C declarations. It is so similar that some people include normal system headers in their RPC protocol specifications. But be careful! Never do that! XDR and C are different languages and there are lots of subtleties such as precise type sizes, byte ordering and member alignment that a C header file does not need to take care of.

Just as in C you have several kinds of basic integer types (`char`, `short`, `int`, `long`) with `signed` and `unsigned` versions, single and double precision real numbers (`float` and `double`). You also have `void` to specify empty (void), and pointers to build dynamic structures.

Basic support for user defined types such as type definition clauses (`typedef`), structs, unions and arrays are also provided.

Unlike C, XDR enums are true types. They are not mere enumerations of integer constants, but discrete ranges for variables (even though they are not checked...)

XDR defines some extra types such as booleans (`bool`), strings (`string`) and opaque types (`opaque`). XDR also provides some extensions to standard C constructions such as discriminated unions.

3.1 XDR examples

Let's go through most of those features with concrete examples:

```
enum Priority {
    PRIO_LOW = 1,
    PRIO_MED = 2,
    PRIO_HIGH = 3,
};
struct Alarm {
    Priority prio;
    string text<>;
};
```

The above segment defines a composite type `Alarm` with two fields, a priority (which is an enumeration with three possible values) and a variable width string with no maximum size limit.

The easiest way to get acquainted with XDR is by experimenting with `rpcgen`. Enter the above lines in a `test.x` file and run `rpcgen -h -N test.x > test.h`. The `-h` option generates the header file with the equivalent C definitions for your particular machine.

If you examine the `test.h` you will discover the equivalences. XDR strings are simple C strings (`char` pointers). Also note that there is a type definition for type `Alarm`, so you don't need to type `struct Alarm myAlarm;`. All you have to do is type `Alarm myAlarm;`. The same applies to the enumeration `Priority`.

The example below defines a file with a name that has up to 255 characters and data:

```
const MAXFILENAMELEN = 256;
typedef string FileName<MAXFILENAMELEN>;
struct File {
    FileName name;
    opaque data<>;
};
```

It is possible that the contents of the file may not be clearly structured or we may not want to specify the exact structure. In this case we just use an `opaque` type with variable length (variable length is indicated by angle brackets) and no maximum size limit. You could also specify a fixed size between square brackets (as an array) but obviously that doesn't make sense in this example. XDR will preserve the byte ordering of the file in both communicating machines, making it the responsibility of the application to read the data correctly for each architecture. Note that consts in XDR (unlike C) may be used to specify dimensions of arrays (they are translated into preprocessor constants).

3.1.1 Discriminated unions

XDR provides a useful extension for `union` called *discriminated union*. This is already present in other languages such as Ada or Pascal:

```
enum ReturnType {
    RESULT_NAME = 1
    RESULT_CODE = 2
};
union Result switch(ReturnType type) {
case RESULT_NAME:
    string name<>;
case RESULT_CODE:
    int code;
default:
    void;
};
```

This means that `Result` has a field called `type` and depending on its value it could also have a field `name`, a field `code` or no more fields at all. Obviously, it is implemented as a struct with a union inside it. If you rerun `rpcgen` over this example you will see that it defines a struct named `Result` with a union inside it called `Result_u`. You will need to know this in order to access that kind of structure from C.

3.1.2 Dynamically allocated structures

The next example shows you how to specify dynamic structures in XDR:

```
struct List {
   List* next;
   string data<>;
};
typedef List* ListPtr;

program RPCDEMO {
    version RPCDEMO_VERS_ORIG {
        void RPCDEMO_PRINT(ListaPtr lp) = 1;
    } = 1;
} = 600000;
```

The example is pretty easy to understand but there are two small details you should be aware of:

- We have defined a new type for the list pointer and you may think that it is rather useless. Well, this is not quite true. Look at the generated code with and without the type. As you probably have already noticed, `rpcgen` is not exactly brilliant.

- Whenever you use dynamic structures there is a question hanging in the air. Who allocates memory and who is responsible for freeing it? The usual answer is: the same module that allocates memory is responsible for freeing it when it is no longer useful.

  In a distributed environment it is a bit more difficult. If the client allocates a dynamic data structure and passes it to the server in a procedure call, then the server automatically allocates memory for it and frees the memory after calling your implementation (beware, you cannot assume that dynamic arguments will persist in your code). But if the server returns a dynamic structure, then the client allocates memory but cannot free anything because it doesn't know when it will cease to be useful. It is your responsibility to free the memory that has been allocated by using `xdr_free(xdr_ListPtr, list_ptr)`. The first parameter is the conversion function that is generated automatically with rpcgen (try `rpcgen -c -N text.x > text_xdr.c`) and the second argument is the pointer returned by any remote procedure call.

  Remember that this also applies to built-in types (such as `string`). But in this case, no conversion routine is generated, it is in the system library. Beware! don't use `xdr_string`, use `xdr_wrapstring`. We will leave the reason as an exercise for you.

One last comment about dynamic types. You cannot use `opaque` as a type for any argument in an RPC. As a way around this you may define a type which carries out the same functions as `opaque` type. The reason is closely connected to the exercise we suggested above. To clarify things we'll set another example:

```
/* This is correct XDR */
typedef opaque op<>;
program DEMO {
    version DEMO_1 {
        void DEMO_PROC_1(op o) = 1;
    } = 1;
} = 600000;

/* This is incorrect XDR */
program DEMO2 {
    version DEMO2_1 {
        void DEMO2_PROC_1(opaque o<>) = 1;
    } = 1;
} = 600001;
```

Look at the generated code for the right version, then examine the declaration of `xdr_string` and `xdr_wrapstring` in `/usr/include/rpc/xdr.h`. Now just think for a second. Without looking at the Sun RPC library source, write down the implementation of `xdr_wrapstring`.

3.1.3 Protocol specification

As you have already seen in many previous examples, the protocol between client and server is specified as a set of procedures with an optional return value and any number of arguments. Each set of procedures is grouped together in a protocol version, with an associated number, and all of the versions are grouped together in an RPC program. This is a complete template:

```
program PROG {
    version PROG1 {
        void PROG_PROC1(int a) = 1;
        void PROG_PROC2(string str<>) = 2;
    } = 1;
    version PROG2 {
        int PROG_PROC1(int a, int b) = 1;
        int PROG_PROC2(string str<>) = 2;
    } = 2;
} = 600000;
```

Procedure numbers and names must be unique in a version, version numbers and identifiers must be unique in each program, and each program number and identifier must be unique in the whole system.

The most important subtlety of Sun RPC has to do with the underlying implementation protocol. All the data needed to perform a call in a client's program may be transmitted using a datagram oriented protocol called UDP. You don't need a detailed knowledge of how the RPC implementation works, but you should be aware that UDP provides non-reliable communication between peers. That is to say, the datagrams may not reach the other end. In this situation Sun RPC implementation tries to send the datagram again, but then there are risks of duplicates at the other end. There are also some other situations that could cause datagram replicas at the other end. Sun RPC implementation does not take care of duplicates, so you may end up with multiple calls to a given procedure when you only wanted a single call.

This is called *at least once* semantics. That means you are guaranteed that the remote procedure will be called but nobody assures you that it will be called just once.

For this reason all the protocols should be specified in such a way that multiple calls to a procedure with the same arguments will lead to exactly the same results. These are called *idempotent protocols*. For example, if you want to read a file then you must either read the entire file in one procedure call or specify the offset from which you want to read data.

If you use TCP as the transport protocol of your RPCs then you have full guarantees of delivery and integrity of data streams and you don't have to worry about these details for any practical purposes.

3.1.4 Preprocessing features

XDR has some features that allow for better control over the `rpcgen` generated code. First of all, `rpcgen` passes the XDR specification through the C preprocessor permitting you to include any legal preprocessor construct. There is also a set of defined preprocessor constants that tell you when `rpcgen` is generating a particular file:

`RPC_HDR`	defined when compiling into header files
`RPC_XDR`	defined when compiling into XDR conversion routines
`RPC_SVC`	defined when compiling into server-side stubs
`RPC_CLNT`	defined when compiling into client-side stubs

Besides that `rpcgen` passes all of the lines starting with '%' to the output file (removing the initial '%'). For example:

```
#ifdef RPC_HDR
%#include "config.h"
%/* This is also the only way to put a comment in
%   the generated file */
#endif
```

The generated header will contain that include file (possibly with some configuration data) and the comment. All of the other comments in the XDR file will disappear in the preprocessing phase.

4 Appendix: GLADE, GTK Interface Builder

Glade[2] is a user interface builder for GTK developed by Damon Chaplin. Although it is in its early stages of development it has reached a situation that we consider useful for initial GTK development. Debian includes the latest releases of Glade conveniently packaged in section *devel*.

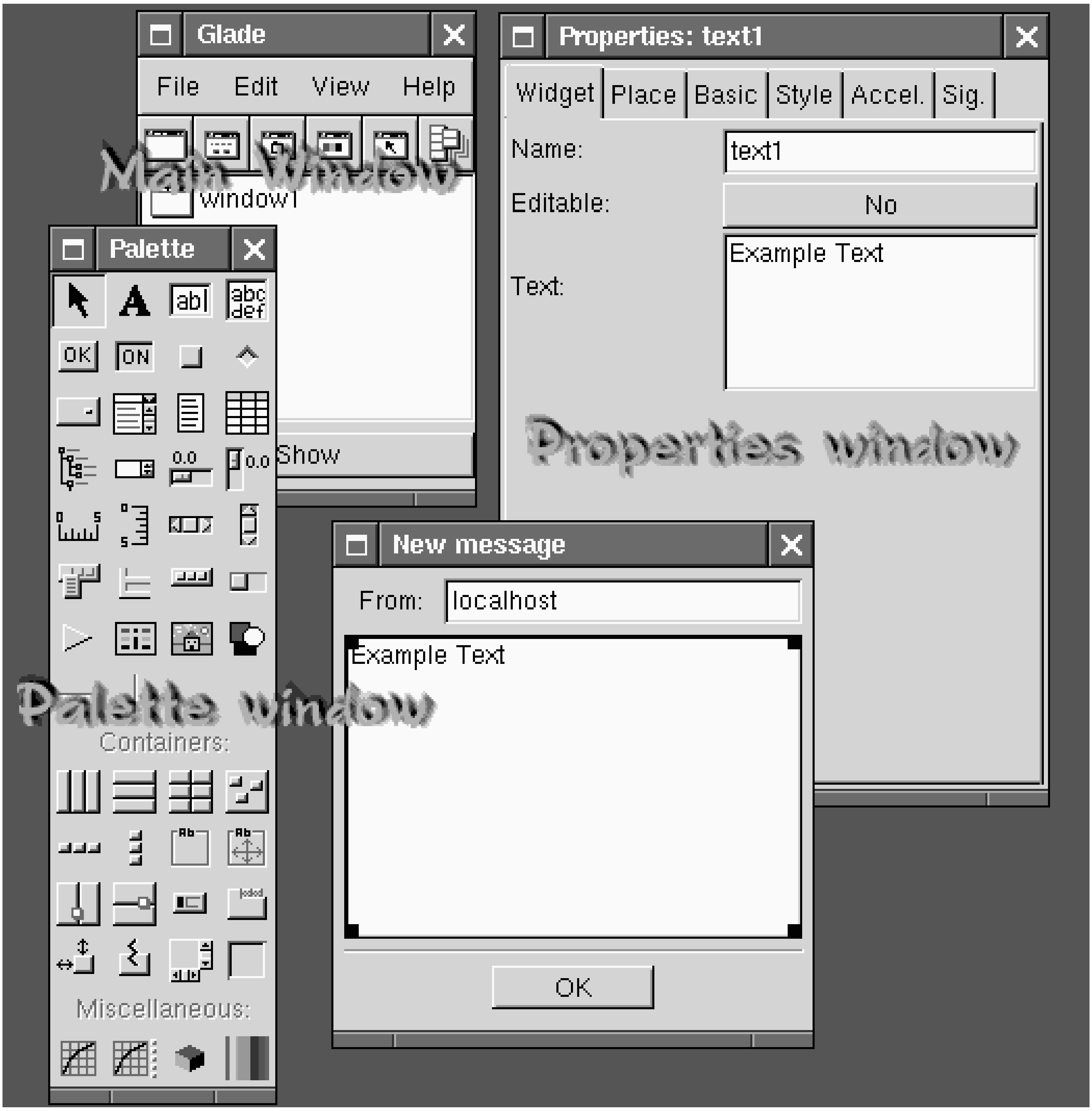
The first problem you will encounter is that it lacks documentation. But don't worry, it is supposed to be very intuitive and you will have some help in these pages.

If you run Glade you will see the windows shown in figure 5.

The main window lists the set of top level widgets defined in your application. You can create a top level widget by clicking one of the toolbar buttons in the main window. The first button creates normal windows, the second is for dialogs, the next three for standard GTK dialogs (file selection, color selection and input device selection) and the last one for popup menus.

---

[2] `http://glade.pn.org/`



window1

Name: text1

Editable: No

Text: Example Text

From: localhost

Example Text

Figure 5: Working with Glade

Click on the *new window* button to create the main window. Glade automatically assigns a name `window1` and some default properties. You can change those default properties in the *properties window*. New widgets can be inserted by selecting the widget type in the *palette window* and then clicking over your window. The widgets on the lower part of the palette are containers or placeholders for more widgets that perform some kind of layout. As an example of this, let's carry out the application shown in fig. 6 (the client part of the GTK/RPC example of section 2).

The properties window shows the properties for the currently selected widget (remember that a window is also a widget). Edit the title property of the window (it is in the *Widget* tab of the properties window) and see how it is automatically updated in your window.

4.1 Widget packing

If you try to put a widget in the window you will see that the widget is expanded to fill the entire window space. If you have already tried it then delete the unwanted widget by just selecting it (clicking on it) and pressing the Delete key. Before we can start creating any current widget we must first set up the placeholders for the widgets in the correct corresponding positions.

Look for the *Vertical Box* container widget in the lower part of the palette window and click on it. You may know what each element of the Glade interface is by just positioning the mouse pointer over it and a tooltip with useful information will appear in a second. Then click on your new window (`window1`). A dialog will ask for the number of rows: enter three (one for the destination host, another for the message text, and another for the buttons). Now you will see your window split in three parts.

Click now in the *Horizontal Box* and put it into the upper row of your window (with just another click). Now enter two columns (one for the label and another for the input text widget).

We won't need more packing elements for this simple example, but it may interest you to try more of them. For example, the table widget is for tabular designs; the frame just adds some decoration; the aspect frame enables you to force a given aspect ratio for a particular zone of the user interface, etc. There is one packing widget called *Fixed Positions*. It allows user specified positioning of the widgets in much the same way as Microsoft Visual Basic. Don't use it, relative positions are much better for carrying out resizing and also save on development time.

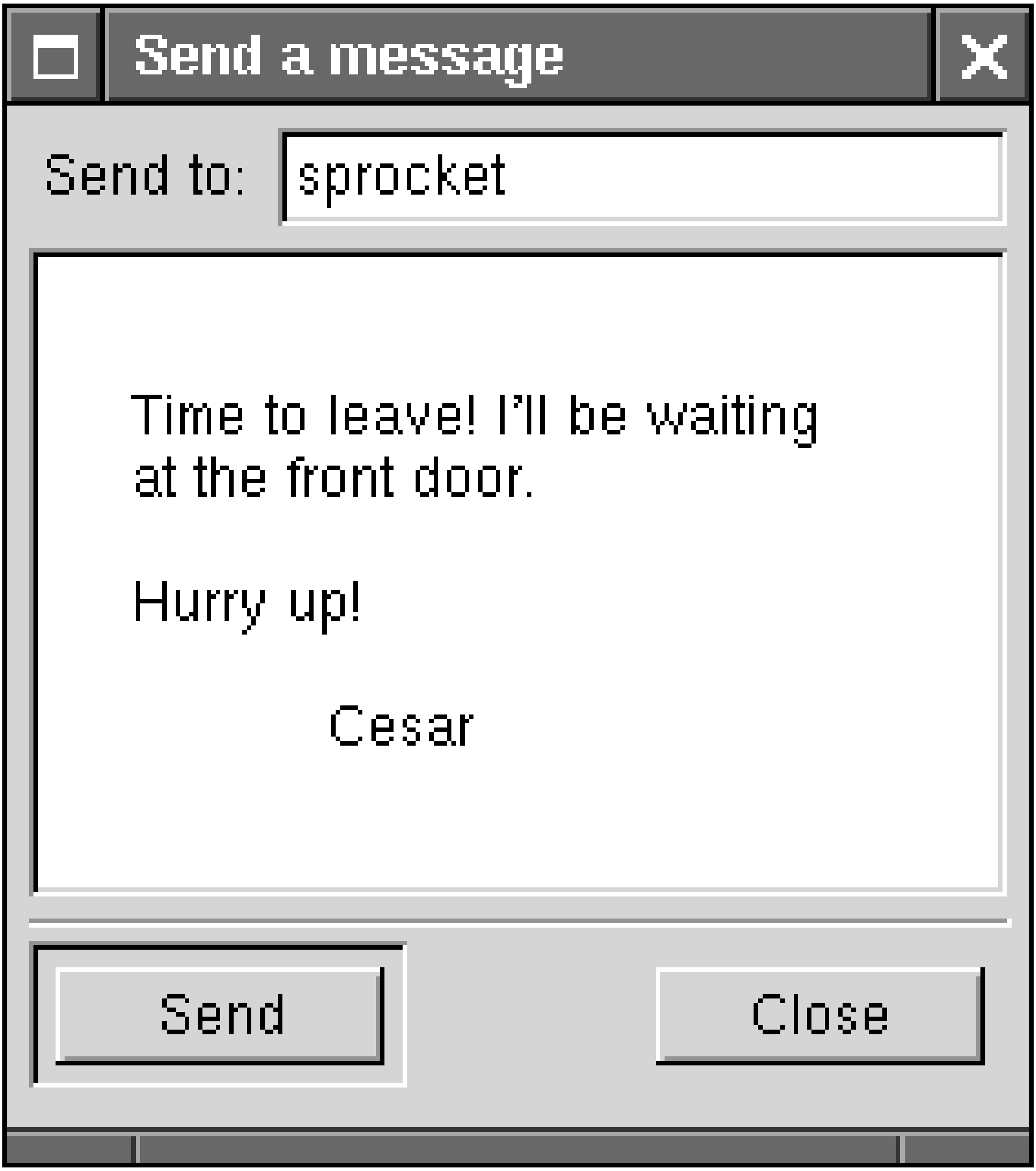
4.2 Adding widgets

It is time to create a current widget. Place a *label* in the upper left-hand corner of your window (click on the label widget of the palette window, and then click on the placeholder in the upper left-hand corner). Change the *Label* property to something similar to "Send to:". Don't change the name of the widget. The name of the widget will be the name for the variable holding the widget, so it cannot contain spaces. Now place the *Text Entry* widget in the upper right-hand corner and the *Text Box* in the middle.

You can place the buttons using a Horizontal Box and some other widgets for proper alignment. But it is easier to just put a *Button Box* with two buttons (in the containers section of the palette window). Edit the labels for each button.

If you look with detail at the original project window, there is another widget, a horizontal separator (an engraved line), that we have missed. It is there on purpose to illustrate new row creation. Click with the right button over the *Text Box* in your window. You will see a popup menu. This menu lists all of the widgets in that particular zone of your window and some useful tasks that can be carried out with each of them. For example, you could select the window again (to change the title or any other property) by clicking on the `window1` item and then in the `Select` item. It is also useful to change the number of rows or columns in packing widgets. Click over the `vbox1` item and then on the *Insert After* subitem. You now have a placeholder in which to put a horizontal separator.

Now there is only a small difference in margins between widgets. Select the window (right click at any point in the window, then select the `window1` submenu and click on the *Select* item). Change the border to 5 in the properties window. Now select the text box in the middle of the window, and go to a tab labeled *Place* in the properties window. Change the padding to 5.



Send to: sprocket

Time to leave! I’ll be waiting
at the front door.

Hurry up!

Cesar

Figure 6: Sample GTK application

There is a final detail that could put the finishing touch to the interface, namely the tooltips. Go to the *Basic* tab of the properties window for each widget that you think should have a tooltip and fill in the property labeled *Tooltip*.

4.3 Signal handlers

Obviously your application is more than just visual aspect. All of the functionality of a GTK application is included as signal handlers. A signal is a concept that abstracts any significant event that could happen to a given widget (for example, positioning the mouse pointer over it, clicking over it, changing some property, etc). We just need to take care of the clicks over the two buttons. All of the other widgets are passive.

Select a button and go to the *Sig.* tab in the properties window. Click on the three dots button to the right of the Signal field (in the lower part of the window). You will see a list of all available signals for that particular selected widget. Select the *clicked* signal and click on the OK button. Now you will see that Glade has automatically selected a name for your signal handler. Click on the *Add* button to accept the new signal handler. Do the same with the other button.

4.4 Code generation

First of all you should save your design as often as possible in the Glade format. This is as simple as selecting the *Save As* item in the *File* menu of the main window. Glade is a bit unstable and you should save as often as you can and make several backup copies. Use a meaningful extension such as `.g` or `.glade`

To generate the C source for that application just click on the *Write Source Code* item of the main window File menu. You will have a lot of new files:

`clnt.glade`	Glade design (project) file
`gladesrc.c`	Glade widget creation functions
`gladesrc.h`	Glade widget creation declarations
`gladesig.c`	Glade signal handler templates and main function
`gladesig.h`	Glade signal handler declarations
`configure.in`	File to allow autoconfiguration of the source
`Makefile.am`	Source for automatic Makefile generation
`Makefile`	Dummy Makefile to generate the real one

In order to compile it you need to install the following GNU utilities: *autoconf*, *automake*, *m4*, *perl* and *gawk*. You should also install *libtool* if you are a serious developer. Most of these utilities will probably already have been installed. Just look for the missing ones. Debian provides all of these utilities under the *devel* section.

The Makefile that is generated is only a stub to generate the real one. Take a good look at it. You will need to perform all those steps to regenerate the Makefile if you change the Makefile.am

Edit the Makefile.am to change the program name (GLADE always uses `project1`, independently of the file saved). Also add any other source file you use in your project. Now you just need to run `make`. You have an application running although it does nothing useful. Edit `gladesig.c` to add your own functionality and read all of the source file to learn how to program with GTK. We highly recommend reading the GTK tutorial[3] for any GTK programmer.

Whenever you wish to make changes to `Makefile.am` you must run `aclocal; automake; autoconf`. If you want to compile your application in another system you should run `./configure; make`. The `configure` script is automatically created by autoconf and is responsible for guessing the particular configuration in which it is being compiled. It was automatically executed the first time you typed `make`.

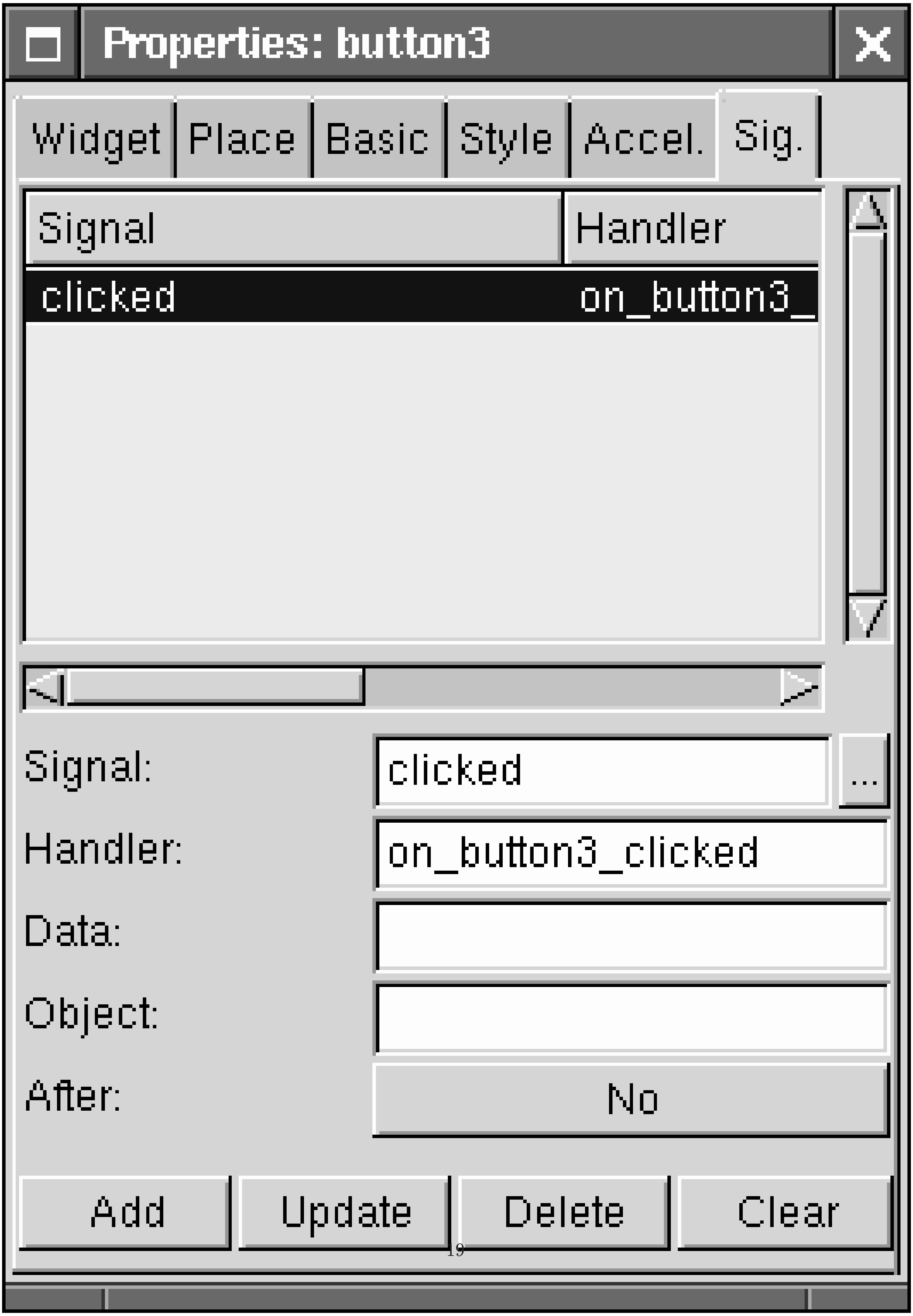
4.5 Some final notes

The server GUI is as simple as the client. Just make sure that the *editable* property is not set in the text box and the text entry.

Glade is in the early stages of development. All of these comments and examples about `glade-0.3.8` are known to be true. We will try to keep these pages updated with the latest versions.

---

[3] `http://www.gtk.org/`



Signal	Handler
clicked	on_button3_

Signal: clicked
Handler: on_button3_clicked
Data:
Object:
After: No

Figure 7: Setting the signal handlers of a button in Glade

5 Appendix: Source code for *hello world* example

This section contains the source code for the hello world example described in section1. You may download[4] all the examples as a compressed tar archive. The file `hello.x` contains the protocol specification. We also provide a `Makefile` to simplify the compilation procedure. The remaining files are those generated by `rpcgen` with minor modifications.

5.1 hello.x

```
/*
  Sample XDR specification for hello world
 */
program HELLO {
    version HELLO_VERS_1 {
/* the following procedure allows testing
   whether the server is running with:

      rpcinfo -t localhost 400000 1  ;: tests tcp transport
      rpcinfo -u localhost 400000 1  ;: tests udp transport

   see "man rpcinfo" for further info */
        void HELLO_NULL(void) = 0;
/*                      ^-- procedure number must be 0 for this */


/* the following procedure is a remote version of puts
   see "man puts" for further info */
        int HELLO_PUTS(string text<>) = 1;
/*                              ^-- procedure number */
    } = 1;  /* the version number */
} = 400000; /* the program number */
```

5.2 hello.h

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#ifndef _HELLO_H_RPCGEN
#define _HELLO_H_RPCGEN

#include <rpc/rpc.h>


#define HELLO ((u_long)400000)
#define HELLO_VERS_1 ((u_long)1)

#ifdef __cplusplus
#define HELLO_NULL ((u_long)0)
extern "C" void * hello_null_1(CLIENT *);
extern "C" void * hello_null_1_svc(struct svc_req *);
#define HELLO_PUTS ((u_long)1)
```

---

[4] `examples.tar.gz`

```
extern "C" int * hello_puts_1(char *, CLIENT *);
extern "C" int * hello_puts_1_svc(char *, struct svc_req *);

#elif __STDC__
#define HELLO_NULL ((u_long)0)
extern  void * hello_null_1(CLIENT *);
extern  void * hello_null_1_svc(struct svc_req *);
#define HELLO_PUTS ((u_long)1)
extern  int * hello_puts_1(char *, CLIENT *);
extern  int * hello_puts_1_svc(char *, struct svc_req *);

#else /* Old Style C */
#define HELLO_NULL ((u_long)0)
extern  void * hello_null_1();
extern  void * hello_null_1_svc();
#define HELLO_PUTS ((u_long)1)
extern  int * hello_puts_1();
extern  int * hello_puts_1_svc();
#endif /* Old Style C */

#endif /* !_HELLO_H_RPCGEN */
```

5.3 hello_client.c

```
/*
 * This is sample code generated by rpcgen
 * with some modifications for the hello world example
 */

#include "hello.h"


void
hello_1( char* host )
{
CLIENT *clnt;
/* we have deleted all of the dummy arguments and result variables */

clnt = clnt_create(host, HELLO, HELLO_VERS_1, "tcp");
/*  we changed the default transport to tcp ---^^^ */

if (clnt == NULL) {
clnt_pcreateerror(host);
exit(1);
}

/* we have deleted the call to procedure null, it is not useful */

hello_puts_1("Hello World!\n", clnt);
/* we added  ^^^^^^^^^^^^^^^^ the nice hello world string as arg */
/* we are not interested in the return value, so we deleted it */

clnt_destroy( clnt );
}
```

```
main( int argc, char* argv[] )
{
char *host;

if(argc < 2) {
printf("usage: %s server_host\n", argv[0]);
exit(1);
}
host = argv[1];
hello_1( host );
}
```

5.4 hello_server.c

```
/*
 * This is sample code generated by rpcgen
 * modified for the hello world example
 */

#include "hello.h"
#include <stdio.h>
/*       ^^^^^^^^^ We have added required headers */

void *
hello_null_1_svc(struct svc_req *rqstp)
{

static char* result;

/*
 * insert server code here
 */

return((void*) &result);
}

int *
hello_puts_1_svc(char *texti,  struct svc_req *rqstp)
{

static int  result;

/* we have inserted the actual implementation of the
   remote procedure, a simple puts */
result = puts(texti);

return(&result);
}
```

5.5 hello_clnt.c

```
/*
```

```
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include <memory.h> /* for memset */
#include "hello.h"

/* Default timeout can be changed using clnt_control() */
static struct timeval TIMEOUT = { 25, 0 };

void *
hello_null_1(CLIENT *clnt)
{
static char clnt_res;

memset((char *)&clnt_res, 0, sizeof(clnt_res));
if (clnt_call(clnt, HELLO_NULL, xdr_void, NULL, xdr_void, &clnt_res, TIMEOUT) != RPC_SUCCESS) {
return (NULL);
}
return ((void *)&clnt_res);
}

int *
hello_puts_1(char *text,  CLIENT *clnt)
{
static int clnt_res;

memset((char *)&clnt_res, 0, sizeof(clnt_res));
if (clnt_call(clnt, HELLO_PUTS, (xdrproc_t) xdr_wrapstring, &text, (xdrproc_t) xdr_int, &clnt_res, TIM
return (NULL);
}
return (&clnt_res);
}
```

5.6 hello_svc.c

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include "hello.h"
#include <stdio.h>
#include <stdlib.h>/* getenv, exit */
#include <rpc/pmap_clnt.h> /* for pmap_unset */
#include <string.h> /* strcmp */
#include <memory.h>
#include <sys/socket.h>
#include <netinet/in.h>

#ifdef __STDC__
#define SIG_PF void(*)(int)
#endif
```

```
static void *
_hello_null_1(void  *argp, struct svc_req *rqstp)
{
return(hello_null_1_svc(rqstp));
}

static int *
_hello_puts_1(char * *argp, struct svc_req *rqstp)
{
return(hello_puts_1_svc(*argp, rqstp));
}

static void
hello_1(struct svc_req *rqstp, register SVCXPRT *transp)
{
union {
char *hello_puts_1_arg;
} argument;
char *result;
xdrproc_t xdr_argument, xdr_result;
char *(*local)(char *, struct svc_req *);

switch (rqstp->rq_proc) {
case HELLO_NULL:
xdr_argument = (xdrproc_t) xdr_void;
xdr_result = (xdrproc_t) xdr_void;
local = (char *(*)(char *, struct svc_req *)) _hello_null_1;
break;

case HELLO_PUTS:
xdr_argument = (xdrproc_t) xdr_wrapstring;
xdr_result = (xdrproc_t) xdr_int;
local = (char *(*)(char *, struct svc_req *)) _hello_puts_1;
break;

default:
svcerr_noproc(transp);
return;
}
(void) memset((char *)&argument, 0, sizeof (argument));
if (!svc_getargs(transp, xdr_argument, (caddr_t) &argument)) {
svcerr_decode(transp);
return;
}
result = (*local)((char *)&argument, rqstp);
if (result != NULL && !svc_sendreply(transp, xdr_result, result)) {
svcerr_systemerr(transp);
}
if (!svc_freeargs(transp, xdr_argument, (caddr_t) &argument)) {
fprintf(stderr, "unable to free arguments");
exit(1);
}
return;
}
```

```
int
main(int argc, char **argv)
{
register SVCXPRT *transp;

(void) pmap_unset(HELLO, HELLO_VERS_1);

transp = svcudp_create(RPC_ANYSOCK);
if (transp == NULL) {
fprintf(stderr, "cannot create udp service.");
exit(1);
}
if (!svc_register(transp, HELLO, HELLO_VERS_1, hello_1, IPPROTO_UDP)) {
fprintf(stderr, "unable to register (HELLO, HELLO_VERS_1, udp).");
exit(1);
}

transp = svctcp_create(RPC_ANYSOCK, 0, 0);
if (transp == NULL) {
fprintf(stderr, "cannot create tcp service.");
exit(1);
}
if (!svc_register(transp, HELLO, HELLO_VERS_1, hello_1, IPPROTO_TCP)) {
fprintf(stderr, "unable to register (HELLO, HELLO_VERS_1, tcp).");
exit(1);
}

svc_run();
fprintf(stderr, "svc_run returned");
exit(1);
/* NOTREACHED */
}
```

5.7 Makefile

```
# The initial source code was generated with:
# rpcgen -N -a hello.x

# to compile:
# make

# to run:
# execute first the server in the background:
# ./hello_server &
# execute then the client
# ./hello_client localhost
# kill the server when not wanted anymore:
# killall hello_server

# default rule: make hello_client and hello_server
all: hello_client hello_server

# to make hello_server make first hello_client.o and hello_clnt.o
```

```
hello_client: hello_client.o hello_clnt.o

# to make hello_server make first hello_server.o and hello_svc.o
hello_server: hello_server.o hello_svc.o

clean:
$(RM) *.o *~ hello_server hello_client
```

---

6 Appendix: Files generated by GLADE

This section contains the source code files generated by Glade for the example shown in fig reffig:clientui. The file `clnt.glade` is the project file for the client part of the message popper example described in section2. The remaining files in this section are automatically generated using the *Write Source Code* menu option.

6.1 `clnt.glade`

```
-- This line is to convince the browser that this is not XML --
<?xml version="1.0"?>
<GTK-Interface>


<widget>
  <class>GtkWindow</class>
  <name>window1</name>
  <title>Send a message</title>
  <type>GTK_WINDOW_TOPLEVEL</type>
  <position>GTK_WIN_POS_NONE</position>
  <allow_shrink>True</allow_shrink>
  <allow_grow>True</allow_grow>
  <auto_shrink>False</auto_shrink>

  <widget>
    <class>GtkVBox</class>
    <name>vbox1</name>
    <border_width>5</border_width>
    <width>230</width>
    <homogeneous>False</homogeneous>
    <spacing>0</spacing>

    <widget>
      <class>GtkHBox</class>
      <name>hbox1</name>
      <child>
        <padding>0</padding>
        <expand>True</expand>
        <fill>True</fill>
      </child>
      <homogeneous>False</homogeneous>
      <spacing>0</spacing>

      <widget>
        <class>GtkLabel</class>
        <name>label1</name>
```

```
    <child>
      <padding>0</padding>
      <expand>True</expand>
      <fill>True</fill>
    </child>
    <label>Send to: </label>
    <justify>GTK_JUSTIFY_CENTER</justify>
    <xalign>0.5</xalign>
    <yalign>0.5</yalign>
    <xpad>0</xpad>
    <ypad>0</ypad>
  </widget>

  <widget>
    <class>GtkEntry</class>
    <name>entry1</name>
    <child>
      <padding>0</padding>
      <expand>True</expand>
      <fill>True</fill>
    </child>
    <tooltip>Enter the destination host</tooltip>
    <can_focus>True</can_focus>
    <editable>True</editable>
    <text_visible>True</text_visible>
    <text_max_length>0</text_max_length>
    <text>localhost</text>
  </widget>
</widget>

<widget>
  <class>GtkText</class>
  <name>text1</name>
  <child>
    <padding>5</padding>
    <expand>True</expand>
    <fill>True</fill>
  </child>
  <tooltip>Enter the message</tooltip>
  <can_focus>True</can_focus>
  <editable>True</editable>
  <text>Sample text</text>
</widget>

<widget>
  <class>GtkHSeparator</class>
  <name>hseparator1</name>
  <child>
    <padding>0</padding>
    <expand>True</expand>
    <fill>True</fill>
  </child>
</widget>
```

```
    <widget>
      <class>GtkHButtonBox</class>
      <name>hbuttonbox1</name>
      <child>
        <padding>0</padding>
        <expand>True</expand>
        <fill>True</fill>
      </child>
      <height>40</height>
      <layout_style>GTK_BUTTONBOX_DEFAULT_STYLE</layout_style>
      <spacing>30</spacing>
      <child_min_width>85</child_min_width>
      <child_min_height>27</child_min_height>
      <child_ipad_x>7</child_ipad_x>
      <child_ipad_y>0</child_ipad_y>

      <widget>
        <class>GtkButton</class>
        <name>button1</name>
        <tooltip>Click to send the message</tooltip>
        <can_default>True</can_default>
        <has_default>True</has_default>
        <can_focus>True</can_focus>
        <has_focus>True</has_focus>
        <Signal>
          <name>clicked</name>
          <handler>on_button1_clicked</handler>
        </Signal>
        <label>Send</label>
      </widget>

      <widget>
        <class>GtkButton</class>
        <name>button2</name>
        <tooltip>Click to exit the application now</tooltip>
        <can_default>True</can_default>
        <can_focus>True</can_focus>
        <Signal>
          <name>clicked</name>
          <handler>on_button2_clicked</handler>
        </Signal>
        <label>Close</label>
      </widget>
    </widget>
  </widget>
</widget>

</GTK-Interface>
```

6.2 gladesig.h

```
/*  <PROJECT NAME>
 *  Copyright (C) <YEAR> <AUTHORS>
 *
```

```
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/

void
on_button1_clicked                     (GtkButton       *button,
                                        gpointer         user_data);
void
on_button2_clicked                     (GtkButton       *button,
                                        gpointer         user_data);
```

6.3 gladesig.c

```
/*  <PROJECT NAME>
 *  Copyright (C) <YEAR> <AUTHORS>
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/

#include <gtk/gtk.h>
#include "gladesrc.h"
#include "gladesig.h"

int
main (int argc, char *argv[])
{
  GtkWidget *window1;

  gtk_set_locale ();
  gtk_init (&argc, &argv);
```

```
  /*
   * The following code was added by Glade to create one of each component,
   * just so that you see something after building the project.
   * Delete any components that you don't want shown initially.
   */
  window1 = create_window1 ();
  gtk_widget_show (window1);

  gtk_main ();
  return 0;
}

void
on_button1_clicked                     (GtkButton       *button,
                                        gpointer         user_data)
{

}

void
on_button2_clicked                     (GtkButton       *button,
                                        gpointer         user_data)
{

}
```

6.4 gladesrc.h

```
/*  <PROJECT NAME>
 *  Copyright (C) <YEAR> <AUTHORS>
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/

/*
 * This function returns a widget in a component created by Glade.
 * Call it with the toplevel widget in the component (i.e. a window/dialog),
 * or alternatively any widget in the component, and the name of the widget
 * you want returned.
 */
GtkWidget*
```

```
get_widget                              (GtkWidget       *widget,
                                         gchar           *widget_name);

GtkWidget* create_window1 (void);
```

6.5 gladesrc.c

```
/*  <PROJECT NAME>
 *  Copyright (C) <YEAR> <AUTHORS>
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/

#include <gtk/gtk.h>
#include "gladesig.h"
#include "gladesrc.h"

GtkWidget*
get_widget                              (GtkWidget       *widget,
                                         gchar           *widget_name)
{
  GtkWidget *found_widget;

  if (widget->parent)
    widget = gtk_widget_get_toplevel (widget);
  found_widget = (GtkWidget*) gtk_object_get_data (GTK_OBJECT (widget),
                                                   widget_name);
  if (!found_widget)
    g_warning ("Widget not found: %s", widget_name);
  return found_widget;
}

/* This is an internally used function to set notebook tab widgets. */
static void
set_notebook_tab                        (GtkWidget       *notebook,
                                         gint             page_num,
                                         GtkWidget       *widget)
{
  GtkNotebookPage *page;
  GtkWidget *notebook_page;

  page = (GtkNotebookPage*) g_list_nth (GTK_NOTEBOOK (notebook)->children, page_num)->data;
```

```
  notebook_page = page->child;
  gtk_widget_ref (notebook_page);
  gtk_notebook_remove_page (GTK_NOTEBOOK (notebook), page_num);
  gtk_notebook_insert_page (GTK_NOTEBOOK (notebook), notebook_page,
                            widget, page_num);
  gtk_widget_unref (notebook_page);
}

GtkWidget*
create_window1 ()
{
  GtkWidget *window1;
  GtkWidget *vbox1;
  GtkWidget *hbox1;
  GtkWidget *label1;
  GtkWidget *entry1;
  GtkWidget *text1;
  GtkWidget *hseparator1;
  GtkWidget *hbuttonbox1;
  GtkWidget *button1;
  GtkWidget *button2;
  GtkTooltips *tooltips;

  tooltips = gtk_tooltips_new ();

  window1 = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_object_set_data (GTK_OBJECT (window1), "window1", window1);
  gtk_window_set_title (GTK_WINDOW (window1), "Send a message");
  gtk_window_set_policy (GTK_WINDOW (window1), TRUE, TRUE, FALSE);

  vbox1 = gtk_vbox_new (FALSE, 0);
  gtk_object_set_data (GTK_OBJECT (window1), "vbox1", vbox1);
  gtk_widget_show (vbox1);
  gtk_container_add (GTK_CONTAINER (window1), vbox1);
  gtk_widget_set_usize (vbox1, 230, -1);
  gtk_container_border_width (GTK_CONTAINER (vbox1), 5);

  hbox1 = gtk_hbox_new (FALSE, 0);
  gtk_object_set_data (GTK_OBJECT (window1), "hbox1", hbox1);
  gtk_widget_show (hbox1);
  gtk_box_pack_start (GTK_BOX (vbox1), hbox1, TRUE, TRUE, 0);

  label1 = gtk_label_new ("Send to: ");
  gtk_object_set_data (GTK_OBJECT (window1), "label1", label1);
  gtk_widget_show (label1);
  gtk_box_pack_start (GTK_BOX (hbox1), label1, TRUE, TRUE, 0);

  entry1 = gtk_entry_new ();
  gtk_object_set_data (GTK_OBJECT (window1), "entry1", entry1);
  gtk_widget_show (entry1);
  gtk_box_pack_start (GTK_BOX (hbox1), entry1, TRUE, TRUE, 0);
  gtk_tooltips_set_tip (tooltips, entry1, "Enter the destination host", NULL);
  gtk_entry_set_text (GTK_ENTRY (entry1), "localhost");
```

```
  text1 = gtk_text_new (NULL, NULL);
  gtk_object_set_data (GTK_OBJECT (window1), "text1", text1);
  gtk_widget_show (text1);
  gtk_box_pack_start (GTK_BOX (vbox1), text1, TRUE, TRUE, 5);
  gtk_tooltips_set_tip (tooltips, text1, "Enter the message", NULL);
  gtk_text_set_editable (GTK_TEXT (text1), TRUE);
  gtk_widget_realize (text1);
  gtk_text_insert (GTK_TEXT (text1), NULL, NULL, NULL,
                   "Sample text", 11);

  hseparator1 = gtk_hseparator_new ();
  gtk_object_set_data (GTK_OBJECT (window1), "hseparator1", hseparator1);
  gtk_widget_show (hseparator1);
  gtk_box_pack_start (GTK_BOX (vbox1), hseparator1, TRUE, TRUE, 0);

  hbuttonbox1 = gtk_hbutton_box_new ();
  gtk_object_set_data (GTK_OBJECT (window1), "hbuttonbox1", hbuttonbox1);
  gtk_widget_show (hbuttonbox1);
  gtk_box_pack_start (GTK_BOX (vbox1), hbuttonbox1, TRUE, TRUE, 0);
  gtk_widget_set_usize (hbuttonbox1, -1, 40);

  button1 = gtk_button_new_with_label ("Send");
  gtk_object_set_data (GTK_OBJECT (window1), "button1", button1);
  gtk_widget_show (button1);
  gtk_container_add (GTK_CONTAINER (hbuttonbox1), button1);
  GTK_WIDGET_SET_FLAGS (button1, GTK_CAN_DEFAULT);
  gtk_widget_grab_focus (button1);
  gtk_widget_grab_default (button1);
  gtk_tooltips_set_tip (tooltips, button1, "Click to send the message", NULL);
  gtk_signal_connect (GTK_OBJECT (button1), "clicked",
                      GTK_SIGNAL_FUNC (on_button1_clicked),
                      NULL);

  button2 = gtk_button_new_with_label ("Close");
  gtk_object_set_data (GTK_OBJECT (window1), "button2", button2);
  gtk_widget_show (button2);
  gtk_container_add (GTK_CONTAINER (hbuttonbox1), button2);
  GTK_WIDGET_SET_FLAGS (button2, GTK_CAN_DEFAULT);
  gtk_tooltips_set_tip (tooltips, button2, "Click to exit the application now", NULL);
  gtk_signal_connect (GTK_OBJECT (button2), "clicked",
                      GTK_SIGNAL_FUNC (on_button2_clicked),
                      NULL);

  return window1;
}
```

6.6 Makefile

```
all:
aclocal
automake -a --foreign
autoconf
./configure
```

```
make
```

6.7 Makefile.am

```
## Process this file with automake to produce Makefile.in

bin_PROGRAMS = project1

project1_SOURCES = \
        gladesrc.c \
        gladesig.c

noinst_HEADERS = \
        gladesrc.h \
        gladesig.h
```

6.8 configure.in

```
dnl Process this file with autoconf to produce a configure script.

AC_INIT(gladesrc.c)


AM_INIT_AUTOMAKE(project1, 0.1)

AC_PROG_CC

AM_PATH_GTK(1.0.0,
            [LIBS="$LIBS $GTK_LIBS" CFLAGS="$CFLAGS $GTK_CFLAGS"],
            AC_MSG_ERROR(Cannot find GTK: Is gtk-config in path?))

dnl Only use -Wall if we have gcc
if test "x$GCC" = "xyes"; then
  if test -z "`echo "$CFLAGS" | grep "\-Wall" 2> /dev/null`" ; then
    CFLAGS="$CFLAGS -Wall"
  fi
fi

AC_OUTPUT(Makefile)
```

7 Appendix: Message popper source code

This section contains the source code for the message popper example described in section2. You may download[5] all the examples as a compressed tar archive. Remember that the files needed to build the server must be in a different directory than those needed to build the client to avoid name clashes in Glade generated source code. Some files are not shown here because they are automatically generated by Glade the first time you execute make.

7.1 Client side

The file `clnt.glade` is the Glade project for the client user interface. Refer to section2 and to our comments in the source code for an explanation of each file.

---

[5] `examples.tar.gz`

7.1.1 rpcmsg.x

```
struct RPCMsg_s {
    string from<>;
    string text<>;
};

typedef struct RPCMsg_s RPCMsg;

program RPCMSG {
    version RPCMSG_VERS_1 {
        void RPCMSG_NULL(void) = 0;
        void RPCMSG_NEW_MSG(RPCMsg msg) = 1;
    } = 1;
} = 400004;
```

7.1.2 clnt.glade

```
<?xml version="1.0"?>
<GTK-Interface>


<widget>
  <class>GtkWindow</class>
  <name>window1</name>
  <title>Send a message</title>
  <type>GTK_WINDOW_TOPLEVEL</type>
  <position>GTK_WIN_POS_NONE</position>
  <allow_shrink>True</allow_shrink>
  <allow_grow>True</allow_grow>
  <auto_shrink>False</auto_shrink>

  <widget>
    <class>GtkVBox</class>
    <name>vbox1</name>
    <border_width>5</border_width>
    <width>230</width>
    <homogeneous>False</homogeneous>
    <spacing>0</spacing>

    <widget>
      <class>GtkHBox</class>
      <name>hbox1</name>
      <child>
        <padding>0</padding>
        <expand>True</expand>
        <fill>True</fill>
      </child>
      <homogeneous>False</homogeneous>
      <spacing>0</spacing>

      <widget>
        <class>GtkLabel</class>
        <name>label1</name>
        <child>
```

```
      <padding>0</padding>
      <expand>True</expand>
      <fill>True</fill>
    </child>
    <label>Send to: </label>
    <justify>GTK_JUSTIFY_CENTER</justify>
    <xalign>0.5</xalign>
    <yalign>0.5</yalign>
    <xpad>0</xpad>
    <ypad>0</ypad>
  </widget>

  <widget>
    <class>GtkEntry</class>
    <name>entry1</name>
    <child>
      <padding>0</padding>
      <expand>True</expand>
      <fill>True</fill>
    </child>
    <tooltip>Enter the destination host</tooltip>
    <can_focus>True</can_focus>
    <editable>True</editable>
    <text_visible>True</text_visible>
    <text_max_length>0</text_max_length>
    <text>localhost</text>
  </widget>
</widget>

<widget>
  <class>GtkText</class>
  <name>text1</name>
  <child>
    <padding>5</padding>
    <expand>True</expand>
    <fill>True</fill>
  </child>
  <tooltip>Enter the message</tooltip>
  <can_focus>True</can_focus>
  <editable>True</editable>
  <text>Sample text</text>
</widget>

<widget>
  <class>GtkHSeparator</class>
  <name>hseparator1</name>
  <child>
    <padding>0</padding>
    <expand>True</expand>
    <fill>True</fill>
  </child>
</widget>

<widget>
```

```
   <class>GtkHButtonBox</class>
   <name>hbuttonbox1</name>
   <child>
    <padding>0</padding>
    <expand>True</expand>
    <fill>True</fill>
   </child>
   <height>40</height>
   <layout_style>GTK_BUTTONBOX_DEFAULT_STYLE</layout_style>
   <spacing>30</spacing>
   <child_min_width>85</child_min_width>
   <child_min_height>27</child_min_height>
   <child_ipad_x>7</child_ipad_x>
   <child_ipad_y>0</child_ipad_y>

   <widget>
    <class>GtkButton</class>
    <name>button1</name>
    <tooltip>Click to send the message</tooltip>
    <can_default>True</can_default>
    <has_default>True</has_default>
    <can_focus>True</can_focus>
    <has_focus>True</has_focus>
    <Signal>
     <name>clicked</name>
     <handler>on_button1_clicked</handler>
    </Signal>
    <label>Send</label>
   </widget>

   <widget>
    <class>GtkButton</class>
    <name>button2</name>
    <tooltip>Click to exit the application now</tooltip>
    <can_default>True</can_default>
    <can_focus>True</can_focus>
    <Signal>
     <name>clicked</name>
     <handler>on_button2_clicked</handler>
    </Signal>
    <label>Close</label>
   </widget>
  </widget>
 </widget>
</widget>

</GTK-Interface>
```

7.1.3 gladesrc.h

```
/*  <PROJECT NAME>
 *  Copyright (C) <YEAR> <AUTHORS>
 *
 *  This program is free software; you can redistribute it and/or modify
```

```
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/

/*
 * This function returns a widget in a component created by Glade.
 * Call it with the toplevel widget in the component (i.e. a window/dialog),
 * or alternatively any widget in the component, and the name of the widget
 * you want returned.
 */
GtkWidget*
get_widget                             (GtkWidget       *widget,
                                        gchar           *widget_name);

GtkWidget* create_window1 (void);
```

7.1.4 gladesrc.c

```
/*  <PROJECT NAME>
 *  Copyright (C) <YEAR> <AUTHORS>
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/

#include <gtk/gtk.h>
#include "gladesig.h"
#include "gladesrc.h"

GtkWidget*
get_widget (GtkWidget       *widget,
    gchar           *widget_name)
{
```

```
  GtkWidget *found_widget;

  if (widget->parent)
    widget = gtk_widget_get_toplevel (widget);
  found_widget = (GtkWidget*) gtk_object_get_data (GTK_OBJECT (widget),
                                                   widget_name);
  if (!found_widget)
    g_warning ("Widget not found: %s", widget_name);
  return found_widget;
}

GtkWidget*
create_window1 ()
{
  GtkWidget *window1;
  GtkWidget *vbox1;
  GtkWidget *hbox1;
  GtkWidget *label1;
  GtkWidget *entry1;
  GtkWidget *text1;
  GtkWidget *hseparator1;
  GtkWidget *hbuttonbox1;
  GtkWidget *button1;
  GtkWidget *button2;
  GtkTooltips *tooltips;

  tooltips = gtk_tooltips_new ();

  window1 = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_object_set_data (GTK_OBJECT (window1), "window1", window1);
  gtk_window_set_title (GTK_WINDOW (window1), "Send a message");
  gtk_window_set_policy (GTK_WINDOW (window1), TRUE, TRUE, FALSE);

  vbox1 = gtk_vbox_new (FALSE, 0);
  gtk_object_set_data (GTK_OBJECT (window1), "vbox1", vbox1);
  gtk_widget_show (vbox1);
  gtk_container_add (GTK_CONTAINER (window1), vbox1);
  gtk_widget_set_usize (vbox1, 230, -1);
  gtk_container_border_width (GTK_CONTAINER (vbox1), 5);

  hbox1 = gtk_hbox_new (FALSE, 0);
  gtk_object_set_data (GTK_OBJECT (window1), "hbox1", hbox1);
  gtk_widget_show (hbox1);
  gtk_box_pack_start (GTK_BOX (vbox1), hbox1, TRUE, TRUE, 0);

  label1 = gtk_label_new ("Send to: ");
  gtk_object_set_data (GTK_OBJECT (window1), "label1", label1);
  gtk_widget_show (label1);
  gtk_box_pack_start (GTK_BOX (hbox1), label1, TRUE, TRUE, 0);

  entry1 = gtk_entry_new ();
  gtk_object_set_data (GTK_OBJECT (window1), "entry1", entry1);
  gtk_widget_show (entry1);
  gtk_box_pack_start (GTK_BOX (hbox1), entry1, TRUE, TRUE, 0);
```

```
  gtk_tooltips_set_tip (tooltips, entry1, "Enter the destination host", NULL);
  gtk_entry_set_text (GTK_ENTRY (entry1), "localhost");

  text1 = gtk_text_new (NULL, NULL);
  gtk_object_set_data (GTK_OBJECT (window1), "text1", text1);
  gtk_widget_show (text1);
  gtk_box_pack_start (GTK_BOX (vbox1), text1, TRUE, TRUE, 5);
  gtk_tooltips_set_tip (tooltips, text1, "Enter the message", NULL);
  gtk_text_set_editable (GTK_TEXT (text1), TRUE);

  hseparator1 = gtk_hseparator_new ();
  gtk_object_set_data (GTK_OBJECT (window1), "hseparator1", hseparator1);
  gtk_widget_show (hseparator1);
  gtk_box_pack_start (GTK_BOX (vbox1), hseparator1, TRUE, TRUE, 0);

  hbuttonbox1 = gtk_hbutton_box_new ();
  gtk_object_set_data (GTK_OBJECT (window1), "hbuttonbox1", hbuttonbox1);
  gtk_widget_show (hbuttonbox1);
  gtk_box_pack_start (GTK_BOX (vbox1), hbuttonbox1, TRUE, TRUE, 0);
  gtk_widget_set_usize (hbuttonbox1, -1, 40);

  button1 = gtk_button_new_with_label ("Send");
  gtk_object_set_data (GTK_OBJECT (window1), "button1", button1);
  gtk_widget_show (button1);
  gtk_container_add (GTK_CONTAINER (hbuttonbox1), button1);
  GTK_WIDGET_SET_FLAGS (button1, GTK_CAN_DEFAULT);
  gtk_widget_grab_focus (button1);
  gtk_widget_grab_default (button1);
  gtk_tooltips_set_tip (tooltips, button1, "Click to send the message", NULL);
  gtk_signal_connect (GTK_OBJECT (button1), "clicked",
                      GTK_SIGNAL_FUNC (on_button1_clicked),
                      NULL);

  button2 = gtk_button_new_with_label ("Close");
  gtk_object_set_data (GTK_OBJECT (window1), "button2", button2);
  gtk_widget_show (button2);
  gtk_container_add (GTK_CONTAINER (hbuttonbox1), button2);
  GTK_WIDGET_SET_FLAGS (button2, GTK_CAN_DEFAULT);
  gtk_tooltips_set_tip (tooltips, button2, "Click to exit the application now", NULL);
  gtk_signal_connect (GTK_OBJECT (button2), "clicked",
                      GTK_SIGNAL_FUNC (on_button2_clicked),
                      NULL);

  return window1;
}
```

7.1.5 gladesig.h

```
/*  <PROJECT NAME>
 *  Copyright (C) <YEAR> <AUTHORS>
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
```

```
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/

void
on_button1_clicked                     (GtkButton       *button,
                                        gpointer         user_data);
void
on_button2_clicked                     (GtkButton       *button,
                                        gpointer         user_data);
```

7.1.6 gladesig.c

```
/*  <PROJECT NAME>
 *  Copyright (C) <YEAR> <AUTHORS>
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/

#include <pwd.h>
#include <unistd.h>
#include <sys/types.h>
#include <gtk/gtk.h>
#include "gladesrc.h"
#include "gladesig.h"
#include "rpcmsg_client.h"

int
main (int argc, char *argv[])
{
  GtkWidget *window1;

  gtk_set_locale ();
```

```
  gtk_init (&argc, &argv);

  window1 = create_window1();
  gtk_widget_show (window1);

  gtk_main ();
  return 0;
}

void
on_button1_clicked (GtkButton       *button,
    gpointer         user_data)
{
    char myhost[256];
    char from[256];
    char msg[4096];
    char* host;
    int i;
    guint len;
    GtkText* w;
    struct passwd* pw;

    /* extract message text */
    w = GTK_TEXT(get_widget(GTK_WIDGET(button), "text1"));
    len = gtk_text_get_length(w);
    for (i=0; i < len; ++i) {
msg[i] = GTK_TEXT_INDEX(w, i);
    }
    msg[len] = '\0';

    /* extract destination host name */
    host = gtk_entry_get_text(GTK_ENTRY(get_widget(GTK_WIDGET(button),
   "entry1")));

    /* extract user name and hostname */
    pw = getpwuid(getuid());
    gethostname(myhost, 256);
    sprintf(from, "%s@%s", pw->pw_name, myhost);

    rpcmsg_send_msg(host, from, msg);
}

void
on_button2_clicked (GtkButton       *button,
    gpointer         user_data)
{
    gtk_main_quit();
}
```

7.1.7 rpcmsg_client.h

```
void rpcmsg_send_msg(char* host, char* from, char* msg);
```

7.1.8 rpcmsg_client.c

```
#include "rpcmsg.h"

void
rpcmsg_send_msg(char* host, char* from, char* msg)
{
CLIENT *clnt;
RPCMsg  m;
clnt = clnt_create(host, RPCMSG, RPCMSG_VERS_1, "tcp");
if (clnt == NULL) {
clnt_pcreateerror(host);
exit(1);
}
m.from = from;
m.text = msg;
rpcmsg_new_msg_1(m, clnt);
clnt_destroy( clnt );
}
```

7.1.9 rpcmsg_clnt.c

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include <memory.h> /* for memset */
#include "rpcmsg.h"

/* Default timeout can be changed using clnt_control() */
static struct timeval TIMEOUT = { 25, 0 };

void *
rpcmsg_null_1(CLIENT *clnt)
{
static char clnt_res;

memset((char *)&clnt_res, 0, sizeof(clnt_res));
if (clnt_call(clnt, RPCMSG_NULL, xdr_void, NULL, xdr_void, &clnt_res, TIMEOUT) != RPC_SUCCESS) {
return (NULL);
}
return ((void *)&clnt_res);
}

void *
rpcmsg_new_msg_1(RPCMsg msg,  CLIENT *clnt)
{
static char clnt_res;

memset((char *)&clnt_res, 0, sizeof(clnt_res));
if (clnt_call(clnt, RPCMSG_NEW_MSG, (xdrproc_t) xdr_RPCMsg, &msg, (xdrproc_t) xdr_void, &clnt_res, TIM
return (NULL);
}
return ((void *)&clnt_res);
```

```
}
```

7.1.10 rpcmsg_xdr.c

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include <rpc/types.h>
#include <rpc/xdr.h>

#include "rpcmsg.h"

bool_t
xdr_RPCMsg_s(XDR *xdrs, RPCMsg_s *objp)
{

 register long *buf=buf;

 if (!xdr_string(xdrs, &objp->from, ~0)) {
 return (FALSE);
 }
 if (!xdr_string(xdrs, &objp->text, ~0)) {
 return (FALSE);
 }
return (TRUE);
}

bool_t
xdr_RPCMsg(XDR *xdrs, RPCMsg *objp)
{

 register long *buf=buf;

 if (!xdr_RPCMsg_s(xdrs, objp)) {
 return (FALSE);
 }
return (TRUE);
}
```

7.1.11 rpcmsg.h

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#ifndef _RPCMSG_H_RPCGEN
#define _RPCMSG_H_RPCGEN

#include <rpc/rpc.h>
```

```
struct RPCMsg_s {
char *from;
char *text;
};
typedef struct RPCMsg_s RPCMsg_s;
#ifdef __cplusplus
extern "C" bool_t xdr_RPCMsg_s(XDR *, RPCMsg_s*);
#elif __STDC__
extern  bool_t xdr_RPCMsg_s(XDR *, RPCMsg_s*);
#else /* Old Style C */
bool_t xdr_RPCMsg_s();
#endif /* Old Style C */


typedef RPCMsg_s RPCMsg;
#ifdef __cplusplus
extern "C" bool_t xdr_RPCMsg(XDR *, RPCMsg*);
#elif __STDC__
extern  bool_t xdr_RPCMsg(XDR *, RPCMsg*);
#else /* Old Style C */
bool_t xdr_RPCMsg();
#endif /* Old Style C */


#define RPCMSG ((u_long)400004)
#define RPCMSG_VERS_1 ((u_long)1)

#ifdef __cplusplus
#define RPCMSG_NULL ((u_long)0)
extern "C" void * rpcmsg_null_1(CLIENT *);
extern "C" void * rpcmsg_null_1_svc(struct svc_req *);
#define RPCMSG_NEW_MSG ((u_long)1)
extern "C" void * rpcmsg_new_msg_1(RPCMsg , CLIENT *);
extern "C" void * rpcmsg_new_msg_1_svc(RPCMsg , struct svc_req *);

#elif __STDC__
#define RPCMSG_NULL ((u_long)0)
extern  void * rpcmsg_null_1(CLIENT *);
extern  void * rpcmsg_null_1_svc(struct svc_req *);
#define RPCMSG_NEW_MSG ((u_long)1)
extern  void * rpcmsg_new_msg_1(RPCMsg , CLIENT *);
extern  void * rpcmsg_new_msg_1_svc(RPCMsg , struct svc_req *);

#else /* Old Style C */
#define RPCMSG_NULL ((u_long)0)
extern  void * rpcmsg_null_1();
extern  void * rpcmsg_null_1_svc();
#define RPCMSG_NEW_MSG ((u_long)1)
extern  void * rpcmsg_new_msg_1();
extern  void * rpcmsg_new_msg_1_svc();
#endif /* Old Style C */

#endif /* !_RPCMSG_H_RPCGEN */
```

7.1.12 `Makefile.am`

```
bin_PROGRAMS = msg

msg_SOURCES = gladesrc.c gladesig.c gladesrc.h gladesig.h \
rpcmsg_xdr.c rpcmsg_clnt.c rpcmsg_client.c rpcmsg.h

noinst_HEADERS = gladesrc.h gladesig.h rpcmsg.h

BUILT_SOURCES = rpcmsg.h rpcmsg_xdr.c rpcmsg_clnt.c

EXTRA_DIST = rpcmsg.x

%_clnt.c: %.x %.h
$(RM) $@
rpcgen -C -N -l -o $@ $<

%_xdr.c : %.x
$(RM) $@
rpcgen -C -N -c -o $@ $<

%.h : %.x
$(RM) $@
rpcgen -C -N -h -o $@ $<

mrproper: clean
$(RM) *~ $(BUILT_SOURCES)
```

7.2 Server side

The file `srv.glade` is the Glade project for the server user interface. Refer to section2 and to our comments in the source code for an explanation of each file.

7.2.1 `rpcmsg.x`

```
struct RPCMsg_s {
    string from<>;
    string text<>;
};

typedef struct RPCMsg_s RPCMsg;

program RPCMSG {
    version RPCMSG_VERS_1 {
        void RPCMSG_NULL(void) = 0;
        void RPCMSG_NEW_MSG(RPCMsg msg) = 1;
    } = 1;
} = 400004;
```

7.2.2 `srv.glade`

```
<?xml version="1.0"?>
<GTK-Interface>
```

```
<widget>
  <class>GtkWindow</class>
  <name>window1</name>
  <title>New message</title>
  <type>GTK_WINDOW_TOPLEVEL</type>
  <position>GTK_WIN_POS_NONE</position>
  <allow_shrink>True</allow_shrink>
  <allow_grow>True</allow_grow>
  <auto_shrink>False</auto_shrink>

  <widget>
    <class>GtkVBox</class>
    <name>vbox1</name>
    <border_width>5</border_width>
    <width>230</width>
    <homogeneous>False</homogeneous>
    <spacing>0</spacing>

    <widget>
      <class>GtkHBox</class>
      <name>hbox1</name>
      <child>
        <padding>0</padding>
        <expand>True</expand>
        <fill>True</fill>
      </child>
      <homogeneous>False</homogeneous>
      <spacing>0</spacing>

      <widget>
        <class>GtkLabel</class>
        <name>label1</name>
        <child>
          <padding>0</padding>
          <expand>True</expand>
          <fill>True</fill>
        </child>
        <label>From: </label>
        <justify>GTK_JUSTIFY_CENTER</justify>
        <xalign>0.5</xalign>
        <yalign>0.5</yalign>
        <xpad>0</xpad>
        <ypad>0</ypad>
      </widget>

      <widget>
        <class>GtkEntry</class>
        <name>entry1</name>
        <child>
          <padding>0</padding>
          <expand>True</expand>
          <fill>True</fill>
        </child>
        <can_focus>True</can_focus>
```

```
    <editable>False</editable>
    <text_visible>True</text_visible>
    <text_max_length>0</text_max_length>
    <text>localhost</text>
  </widget>
</widget>

<widget>
  <class>GtkText</class>
  <name>text1</name>
  <child>
    <padding>5</padding>
    <expand>True</expand>
    <fill>True</fill>
  </child>
  <can_focus>True</can_focus>
  <editable>False</editable>
  <text>sample</text>
</widget>

<widget>
  <class>GtkHSeparator</class>
  <name>hseparator1</name>
  <child>
    <padding>0</padding>
    <expand>True</expand>
    <fill>True</fill>
  </child>
</widget>

<widget>
  <class>GtkHButtonBox</class>
  <name>hbuttonbox2</name>
  <child>
    <padding>0</padding>
    <expand>True</expand>
    <fill>True</fill>
  </child>
  <layout_style>GTK_BUTTONBOX_DEFAULT_STYLE</layout_style>
  <spacing>30</spacing>
  <child_min_width>85</child_min_width>
  <child_min_height>27</child_min_height>
  <child_ipad_x>7</child_ipad_x>
  <child_ipad_y>0</child_ipad_y>

  <widget>
    <class>GtkButton</class>
    <name>button3</name>
    <width>75</width>
    <height>32</height>
    <can_default>True</can_default>
    <has_default>True</has_default>
    <can_focus>True</can_focus>
    <label>OK</label>
```

```
      </widget>
    </widget>
  </widget>
</widget>

</GTK-Interface>
```

7.2.3 gladesrc.h

```
/*  <PROJECT NAME>
 *  Copyright (C) <YEAR> <AUTHORS>
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/

/*
 * This function returns a widget in a component created by Glade.
 * Call it with the toplevel widget in the component (i.e. a window/dialog),
 * or alternatively any widget in the component, and the name of the widget
 * you want returned.
 */
GtkWidget* get_widget (GtkWidget *widget, gchar *widget_name);
GtkWidget* create_window1 (char* from, char* text);
```

7.2.4 gladesrc.c

```
/*  <PROJECT NAME>
 *  Copyright (C) <YEAR> <AUTHORS>
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
```

```
*/

#include <gtk/gtk.h>
#include "gladesig.h"
#include "gladesrc.h"

GtkWidget*
get_widget                             (GtkWidget       *widget,
                                        gchar           *widget_name)
{
  GtkWidget *found_widget;

  if (widget->parent)
    widget = gtk_widget_get_toplevel (widget);
  found_widget = (GtkWidget*) gtk_object_get_data (GTK_OBJECT (widget),
                                                   widget_name);
  if (!found_widget)
    g_warning ("Widget not found: %s", widget_name);
  return found_widget;
}

GtkWidget*
create_window1 (char* from, char* msg)
{
  GtkWidget *window1;
  GtkWidget *vbox1;
  GtkWidget *hbox1;
  GtkWidget *label1;
  GtkWidget *entry1;
  GtkWidget *text1;
  GtkWidget *hseparator1;
  GtkWidget *hbuttonbox2;
  GtkWidget *button1;

  window1 = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_object_set_data (GTK_OBJECT (window1), "window1", window1);
  gtk_window_set_title (GTK_WINDOW (window1), "New message");
  gtk_window_set_policy (GTK_WINDOW (window1), TRUE, TRUE, FALSE);

  vbox1 = gtk_vbox_new (FALSE, 0);
  gtk_object_set_data (GTK_OBJECT (window1), "vbox1", vbox1);
  gtk_widget_show (vbox1);
  gtk_container_add (GTK_CONTAINER (window1), vbox1);
  gtk_widget_set_usize (vbox1, 230, -1);
  gtk_container_border_width (GTK_CONTAINER (vbox1), 5);

  hbox1 = gtk_hbox_new (FALSE, 0);
  gtk_object_set_data (GTK_OBJECT (window1), "hbox1", hbox1);
  gtk_widget_show (hbox1);
  gtk_box_pack_start (GTK_BOX (vbox1), hbox1, TRUE, TRUE, 0);

  label1 = gtk_label_new ("From: ");
  gtk_object_set_data (GTK_OBJECT (window1), "label1", label1);
  gtk_widget_show (label1);
```

```
  gtk_box_pack_start (GTK_BOX (hbox1), label1, TRUE, TRUE, 0);

  entry1 = gtk_entry_new ();
  gtk_object_set_data (GTK_OBJECT (window1), "entry1", entry1);
  gtk_widget_show (entry1);
  gtk_box_pack_start (GTK_BOX (hbox1), entry1, TRUE, TRUE, 0);
  gtk_entry_set_editable (GTK_ENTRY (entry1), FALSE);
  gtk_entry_set_text (GTK_ENTRY (entry1), from);

  text1 = gtk_text_new (NULL, NULL);
  gtk_object_set_data (GTK_OBJECT (window1), "text1", text1);
  gtk_widget_show (text1);
  gtk_box_pack_start (GTK_BOX (vbox1), text1, TRUE, TRUE, 5);
  gtk_widget_realize (text1);
  gtk_text_insert (GTK_TEXT (text1), NULL, NULL, NULL, msg, strlen(msg));

  hseparator1 = gtk_hseparator_new ();
  gtk_object_set_data (GTK_OBJECT (window1), "hseparator1", hseparator1);
  gtk_widget_show (hseparator1);
  gtk_box_pack_start (GTK_BOX (vbox1), hseparator1, TRUE, TRUE, 0);

  hbuttonbox2 = gtk_hbutton_box_new ();
  gtk_object_set_data (GTK_OBJECT (window1), "hbuttonbox2", hbuttonbox2);
  gtk_widget_show (hbuttonbox2);
  gtk_box_pack_start (GTK_BOX (vbox1), hbuttonbox2, TRUE, TRUE, 0);

  button1 = gtk_button_new_with_label ("OK");
  gtk_object_set_data (GTK_OBJECT (window1), "button1", button1);
  gtk_widget_show (button1);
  gtk_container_add (GTK_CONTAINER (hbuttonbox2), button1);
  gtk_widget_set_usize (button1, 75, 32);
  GTK_WIDGET_SET_FLAGS (button1, GTK_CAN_DEFAULT);
  gtk_widget_grab_default (button1);
  gtk_signal_connect (GTK_OBJECT (button1), "clicked",
                      GTK_SIGNAL_FUNC (on_button1_clicked),
                      NULL);

  return window1;
}
```

7.2.5 gladesig.h

```
/*  <PROJECT NAME>
 *  Copyright (C) <YEAR> <AUTHORS>
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
```

```
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/

void on_button1_clicked (GtkButton *button, gpointer user_data);
```

7.2.6 gladesig.c

```
/*  <PROJECT NAME>
 *  Copyright (C) <YEAR> <AUTHORS>
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/
#include <sys/time.h>
#include <sys/resource.h>
#include <stdlib.h>
#include <string.h>
#define PORTMAP
#include <rpc/rpc.h>
#include <rpc/pmap_clnt.h>

#include <gtk/gtk.h>
#include "gladesrc.h"
#include "gladesig.h"
#include "rpcmsg.h"
void rpcmsg_1(struct svc_req *rqstp, SVCXPRT *transp);

int
handle_rpc(gpointer data)
{
    fd_set tmp_rd_set = svc_fdset;
    int retval = 0;
    struct rlimit lim;
    struct timeval tv;

    tv.tv_usec = 0; tv.tv_sec = 0; /* we are polling, don't wait */
    getrlimit (RLIMIT_NOFILE, &lim);
    retval=select(lim.rlim_cur, &tmp_rd_set, NULL, NULL, &tv);
    if (retval>0) {
```

```
svc_getreqset (&tmp_rd_set);
    }
    return 1; /* always active */
}

void
activate_rpc_services()
{
    SVCXPRT* xprt;

    xprt = svctcp_create(RPC_ANYSOCK, 0, 0);
    pmap_unset(RPCMSG, RPCMSG_VERS_1);
    pmap_set(RPCMSG, RPCMSG_VERS_1, IPPROTO_TCP, xprt->xp_port);
    svc_register(xprt, RPCMSG, RPCMSG_VERS_1, rpcmsg_1, IPPROTO_TCP);
    gtk_timeout_add(200, handle_rpc, NULL);
}

void
deactivate_rpc_services()
{
    pmap_unset(RPCMSG, RPCMSG_VERS_1);
}

int
main (int argc, char *argv[])
{
  gtk_set_locale ();
  gtk_init (&argc, &argv);

  /* Enable RPC processing */
  activate_rpc_services();
  gtk_main ();
  deactivate_rpc_services();
  return 0;
}

void
on_button1_clicked (GtkButton       *button,
    gpointer         user_data)
{
    gtk_widget_destroy(gtk_widget_get_toplevel(GTK_WIDGET(button)));
}
```

7.2.7 rpcmsg_server.c

```
#include "rpcmsg.h"
#include <gtk/gtk.h>
#include "gladesrc.h"

void *
rpcmsg_null_1_svc(struct svc_req *rqstp)
{
    static char* result;
    return((void*) &result);
```

```
}

void *
rpcmsg_new_msg_1_svc(RPCMsg msg,  struct svc_req *rqstp)
{
    static char* result;
    GtkWidget *window1;
    window1 = create_window1 (msg.from, msg.text);
    gtk_widget_show (window1);
    return((void*) &result);
}
```

7.2.8 rpcmsg_svc.c

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include "rpcmsg.h"
#include <stdio.h>
#include <stdlib.h>/* getenv, exit */
#include <rpc/pmap_clnt.h> /* for pmap_unset */
#include <string.h> /* strcmp */
#include <memory.h>
#include <sys/socket.h>
#include <netinet/in.h>

#ifdef __STDC__
#define SIG_PF void(*)(int)
#endif

static void *
_rpcmsg_null_1(void  *argp, struct svc_req *rqstp)
{
return(rpcmsg_null_1_svc(rqstp));
}

static void *
_rpcmsg_new_msg_1(RPCMsg  *argp, struct svc_req *rqstp)
{
return(rpcmsg_new_msg_1_svc(*argp, rqstp));
}

void
rpcmsg_1(struct svc_req *rqstp, register SVCXPRT *transp)
{
union {
RPCMsg rpcmsg_new_msg_1_arg;
} argument;
char *result;
xdrproc_t xdr_argument, xdr_result;
char *(*local)(char *, struct svc_req *);
```

```
switch (rqstp->rq_proc) {
case RPCMSG_NULL:
xdr_argument = (xdrproc_t) xdr_void;
xdr_result = (xdrproc_t) xdr_void;
local = (char *(*)(char *, struct svc_req *)) _rpcmsg_null_1;
break;

case RPCMSG_NEW_MSG:
xdr_argument = (xdrproc_t) xdr_RPCMsg;
xdr_result = (xdrproc_t) xdr_void;
local = (char *(*)(char *, struct svc_req *)) _rpcmsg_new_msg_1;
break;

default:
svcerr_noproc(transp);
return;
}
(void) memset((char *)&argument, 0, sizeof (argument));
if (!svc_getargs(transp, xdr_argument, (caddr_t) &argument)) {
svcerr_decode(transp);
return;
}
result = (*local)((char *)&argument, rqstp);
if (result != NULL && !svc_sendreply(transp, xdr_result, result)) {
svcerr_systemerr(transp);
}
if (!svc_freeargs(transp, xdr_argument, (caddr_t) &argument)) {
fprintf(stderr, "unable to free arguments");
exit(1);
}
return;
}
```

7.2.9 rpcmsg_xdr.c

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include <rpc/types.h>
#include <rpc/xdr.h>

#include "rpcmsg.h"

bool_t
xdr_RPCMsg_s(XDR *xdrs, RPCMsg_s *objp)
{

 register long *buf=buf;

 if (!xdr_string(xdrs, &objp->from, ~0)) {
 return (FALSE);
 }
```

```
 if (!xdr_string(xdrs, &objp->text, ~0)) {
 return (FALSE);
 }
return (TRUE);
}

bool_t
xdr_RPCMsg(XDR *xdrs, RPCMsg *objp)
{

 register long *buf=buf;

 if (!xdr_RPCMsg_s(xdrs, objp)) {
 return (FALSE);
 }
return (TRUE);
}
```

7.2.10 rpcmsg.h

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#ifndef _RPCMSG_H_RPCGEN
#define _RPCMSG_H_RPCGEN

#include <rpc/rpc.h>


struct RPCMsg_s {
char *from;
char *text;
};
typedef struct RPCMsg_s RPCMsg_s;
#ifdef __cplusplus
extern "C" bool_t xdr_RPCMsg_s(XDR *, RPCMsg_s*);
#elif __STDC__
extern  bool_t xdr_RPCMsg_s(XDR *, RPCMsg_s*);
#else /* Old Style C */
bool_t xdr_RPCMsg_s();
#endif /* Old Style C */


typedef RPCMsg_s RPCMsg;
#ifdef __cplusplus
extern "C" bool_t xdr_RPCMsg(XDR *, RPCMsg*);
#elif __STDC__
extern  bool_t xdr_RPCMsg(XDR *, RPCMsg*);
#else /* Old Style C */
bool_t xdr_RPCMsg();
#endif /* Old Style C */
```

```
#define RPCMSG ((u_long)400004)
#define RPCMSG_VERS_1 ((u_long)1)

#ifdef __cplusplus
#define RPCMSG_NULL ((u_long)0)
extern "C" void * rpcmsg_null_1(CLIENT *);
extern "C" void * rpcmsg_null_1_svc(struct svc_req *);
#define RPCMSG_NEW_MSG ((u_long)1)
extern "C" void * rpcmsg_new_msg_1(RPCMsg , CLIENT *);
extern "C" void * rpcmsg_new_msg_1_svc(RPCMsg , struct svc_req *);

#elif __STDC__
#define RPCMSG_NULL ((u_long)0)
extern  void * rpcmsg_null_1(CLIENT *);
extern  void * rpcmsg_null_1_svc(struct svc_req *);
#define RPCMSG_NEW_MSG ((u_long)1)
extern  void * rpcmsg_new_msg_1(RPCMsg , CLIENT *);
extern  void * rpcmsg_new_msg_1_svc(RPCMsg , struct svc_req *);

#else /* Old Style C */
#define RPCMSG_NULL ((u_long)0)
extern  void * rpcmsg_null_1();
extern  void * rpcmsg_null_1_svc();
#define RPCMSG_NEW_MSG ((u_long)1)
extern  void * rpcmsg_new_msg_1();
extern  void * rpcmsg_new_msg_1_svc();
#endif /* Old Style C */

#endif /* !_RPCMSG_H_RPCGEN */
```

7.2.11 `Makefile.am`

```
bin_PROGRAMS = msgsrv

msgsrv_SOURCES = gladesrc.c gladesig.c gladesrc.h gladesig.h \
rpcmsg_xdr.c rpcmsg_svc.c rpcmsg_server.c rpcmsg.h

noinst_HEADERS = gladesrc.h gladesig.h rpcmsg.h

BUILT_SOURCES = rpcmsg.h rpcmsg_xdr.c rpcmsg_svc.c

EXTRA_DIST = rpcmsg.x

%_svc.c: %.x %.h
$(RM) $@
rpcgen -C -N -m -o $@ $<

%_xdr.c : %.x
$(RM) $@
rpcgen -C -N -c -o $@ $<

%.h : %.x
$(RM) $@
```

```
rpcgen -C -N -h -o $@ $<

mrproper: clean
$(RM) *~ $(BUILT_SOURCES)
```
