

Communication Toolbox

Claude Gomez

Manual Version 1.0 for Scilab 2.4

This is a beta version of the Communication Toolbox. It lacks security problems when using remote communications (see 2.1). A good error trapping is also under development, so you can “block” Scilab when doing mistakes in the names of the linked applications.

GeCI is an interactive communication manager created in order to manage remote executions of programs and allow exchanges of messages between those programs. It offers the possibility to exploit numerous machines on a network, as a virtual computer, by creating a distributed group of independent softwares.

It has been originally developed in the CalICo Project (LaBRI- Université Bordeaux I, France) by Nadine Rouillon.

In Scilab, GeCI manages communications between Scilab itself and other applications (included Scilab itself). In fact, the `scilab` command is a shell script which at last executes the command:

```
<Scilab directory>/bin/geci -local <Scilab directory>/bin/scilex
```

where `<Scilab directory>` is the main directory of Scilab. We will use `<Scilab directory>` to denote this directory in this manual. This means that Scilab is ready to communicate with others applications. In particular, this is the way Metanet Windows are executed in the Metanet toolbox.

In section 1, we explain how to use the Scilab functions of the Communication Toolbox and in section 2 we describe the way to make an application able to communicate with Scilab.

1 Functions of the Communication Toolbox

There are six functions in the Communication Toolbox to make communications. They are `CreateLink`, `DestroyLink`, `ExecAppli`, `GetMsg`, `SendMsg` and `WaitMsg`. They are described in the on-line manual. We are only going here to describe how to use them for exchanging messages between two Scilab programs. You will find in section 2.2 how to communicate between Scilab and other programs.

After executing Scilab, you can execute another Scilab on the same computer from the first one and create a link to it and from it by issuing the commands:

```
-->h=unix_g("hostname");  
-->ExecAppli(SCI+"/bin/scilex",h,"Scilab2")  
-->CreateLink("SELF","Scilab2")  
-->CreateLink("Scilab2","SELF")
```

Each application linked to GeCI has a name. Two special names are defined, "SELF" always stands for the name of the Scilab program where you are and "XGeCI" stands for the first Scilab program.

Now, another Scilab named `Scilab2` appears and you can send a message to it from the first Scilab:

```
-->SendMsg("Hi!","How are you?");
```

And in `Scilab2`, you can get it:

```
-->[type,msg,apply] = GetMsg()
apply =

XGeCI
msg =

How are you?
type =

Hi!
```

You do not have to tell to which application you want to send the message: you send it to all applications linked to you.

You can also send a message from Scilab2 to first Scilab:

```
-->SendMsg("Answer","Fine and you?");
```

and you get it in Scilab:

```
-->[type,msg,apply] = GetMsg()
apply =

Scilab2
msg =

Fine and you?
type =

Answer
```

You can even open a new Scilab, named Scilab3 for instance, link it to Scilab and/or Scilab2 and exchange messages between them after having created links. Use the `DestroyLink` function to destroy a link.

Note that there is no notion of client and server. The server is the GeCI program and all the applications which communicate are at the same level. That means that the `CreateLink`, `DestroyLink` and `ExecAppli` functions can be used in any Scilab instance.

2 Creating applications to communicate with Scilab

To communicate with Scilab, an application must have been prepared for, by including a communication library in it. The way to do this is described in this section.

Suppose you have a C or fortran program and you want to add in it the functionalities to communicate using GeCI. For this you need to add new C functions in the code. For instance, you can create a new C file which you are going to link with your existing C or fortran program. You also need to link the program with the following library:

```
<Scilab directory>/libs/libcomm.a
```

At the beginning of the C code, you must include the following header files:

```
#include "<Scilab directory>/routines/libcomm/libCalCom.h"
#include "<Scilab directory>/routines/libcomm/libCom.h"
```

Then you define the messages known by the application, usually:

```

static void QuitAppli();
static void EndAppli();
static void ParseMessage();
static void MsgError();

static actions_messages tb_messages[]={
    {ID_GeCI,MSG_QUITTER_APPLI,NBP_QUITTER_APPLI,QuitAppli},
    {ID_GeCI,MSG_FIN_APPLI,NBP_FIN_APPLI,EndAppli},
    {NULL,MSG_DISTRIB_LISTE_ELMNT,NBP_DISTRIB_LISTE_ELMNT,ParseMessage},
    {NULL,NULL,0,MsgError}};

```

All these functions have one argument of type message.

QuitAppli is executed when THIS application terminates.

EndAppli is executed when an application executed by THIS application terminates.

MsgError is executed when there is an error in a message.

ParseMessage is executed to get the received messages. For instance, you can write ParseMessage the following way:

```

static char *TheAppli;
static char *TheType;
static char *TheMsg;

static void ParseMessage(message)
Message message;
{
    int lappli, ltype, lmsg;

    lappli = strlen(message.tableau[0]);
    if ((TheAppli = (char *)malloc((unsigned)sizeof(char)*(lappli + 1)))
        == NULL) {
        return;
    }
    strcpy(TheAppli,message.tableau[0]);

    ltype = strlen(message.tableau[3]);
    if ((TheType = (char *)malloc((unsigned)sizeof(char)*(ltype + 1)))
        == NULL) {
        return;
    }
    strcpy(TheType,message.tableau[3]);

    lmsg = strlen(message.tableau[4]);
    if ((TheMsg = (char *)malloc((unsigned)sizeof(char)*(lmsg + 1)))
        == NULL) {
        return;
    }
    strcpy(TheMsg,message.tableau[4]);
}

```

and you get:

- in TheAppli the name of the application which has sent the message
- in TheType the type of the message (an ASCII string)
- in TheMsg the message (an ASCII string)

Then you must initialize the communications.

For this, the main function of the application must understand the arguments `-pipes <pipe1> <pipe2>` which are automatically given by GeCI when executing it by:

```
geci -local <my application>
```

A simple way to do this is to use the `find` function:

```
static int find(s,n,t)
char *s;
int n;
char **t;
{
    int i;
    for (i=0; i<n; i++)
        if (!strcmp(s,t[i])) return(i);
    return(-1);
}

int main(argc, argv)
unsigned int argc;
char **argv;
{
    int igeci;
    int p1, p2;
    igeci = find("-pipes",argc,argv);
    if (igeci == -1) exit(1);

    p1 = atoi(argv[igeci+1]); p2 = atoi(argv[igeci+2]);
```

And you initialize the communications:

```
init_messages(tb_messages,p1,p2);
```

Then it is possible to use the functions of GeCI. For this, you have to send messages to GeCI.

- You can execute an application from your program:

```
envoyer_message_parametres_var(ID_GeCI,
                                MSG_LANCER_APPLI,
                                <appli>,
                                <host>,
                                <path appli>,
                                INS_ID_PIPES,
                                NULL);
```

where `<appli>` is the name you give to the application you execute, `<host>` is the name of the host where you want to execute the application and `<path appli>` is the path of the program of the application on the host. If you want to execute an application locally on the same host, you give the name of your host. You can also execute the application on a remote host on Internet, see [2.1](#).

Every application has a name. The name of your program has also a name given automatically. When you have to use it (to link applications for instance), you can get it by `identificateur_appli()`.

- To create a (directed) link from the application named `<appli1>` to the application named `<appli2>`:

```

envoyer_message_parametres_var( ID_GeCI,
                                MSG_CREER_LIAISON,
                                <appli1>,
                                <appli2>,
                                NULL);

```

Then you are able to send messages from <appli1> to <appli2>.

Note that the two applications must have been executed first by the preceding message.

- To destroy a link from <appli1> to <appli2>:

```

envoyer_message_parametres_var( ID_GeCI,
                                MSG_DETUIRE_LIAISON,
                                <appli1>,
                                <appli2>,
                                NULL);

```

- To send a message to all linked applications:

```

envoyer_message_parametres_var( ID_GeCI,
                                MSG_POSTER_LISTE_ELMNT,
                                <type>,
                                <msg>,
                                NULL);

```

where <type> is a string corresponding to the type of the message and <msg> is the string corresponding to the message.

Note that before being able to exchange messages, applications must have been linked.

- To get a message in an asynchronous way:

```

scanner_messages();

```

- To wait for a message from <appli> in a synchronous way:

```

attendre_reponse(<appli>,
                 MSG_DISTRIB_LISTE_ELMNT,
                 NBP_DISTRIB_LISTE_ELMNT);

```

2.1 Communication between remote hosts

CAUTION: *With this beta version of the Communication Toolbox, nothing has been done for addressing possibly security holes and problems when using the **gecid** daemon and remote **geci** programs. So use them very carefully.*

With GeCI you can also have communications between programs on remote hosts.

Suppose you are on host h1, have a local application a1 and you want to execute an application a2 on host h2 and open a communication between a1 and a2.

First, you must have a **geci** program on both hosts. Second you must have a daemon, called **gecid**, on the remote host. The C source code of this daemon is given in the **geci** directory of Scilab distribution.

You have to give the good path of **geci** program in the source code of program **gecid** in variable **GECI**.

Then, you first start the daemon **gecid** on host h2. After, you execute application a1, Scilab for instance, on host h1 and use GeCI functions to execute application a2 on host h2: you have to give the complete Internet name of host h2 in these functions.

If application a2 is another Scilab, do not forget to give the good **-display** argument to the **scilex** command.

In fact the **gecid** daemon will wait for a socket connection on port 2001 and then start **geci** on host h2.

2.2 Examples

You will find below two complete C programs as examples.

The first program opens Scilab, wait for messages and print them. We call the file of this program `alpha.c` and we suppose that the main directory of Scilab, `<Scilab directory>` is `/usr/local/lib/scilab-2.4`.

Compile this program with:

```
cc -o alpha alpha.c /usr/local/lib/scilab-2.4/libs/libcomm.a
```

and execute it with (if shell is csh):

```
setenv SCI /usr/local/lib/scilab-2.4
/usr/local/lib/scilab-2.4/bin/geci -local alpha
```

Then a new Scilab is executed and you can execute in it commands of the form `SendMsg("Hi", "How are you?")`. On the console you must see:

```
Message received from Scilab
  type: Hi
  message: How are you?
```

"alpha.c" is given below and in the directory `docs/comm` of Scilab distribution.

```
/* **** */
#include <stdio.h>
#include <string.h>

/* Communications headers */
#include "/usr/local/lib/scilab-2.4/routines/libcomm/libCalCom.h"
#include "/usr/local/lib/scilab-2.4/routines/libcomm/libCom.h"

static void QuitAppli();
static void EndAppli();
static void ParseMessage();
static void MsgError();

/* Known messages */
static actions_messages tb_messages[]={
    {ID_GeCI,MSG_QUITTER_APPLI,NBP_QUITTER_APPLI,QuitAppli},
    {ID_GeCI,MSG_FIN_APPLI,NBP_FIN_APPLI,EndAppli},
    {NULL,MSG_DISTRIB_LISTE_ELMNT,NBP_DISTRIB_LISTE_ELMNT,ParseMessage},
    {NULL,NULL,0,MsgError}};

static void QuitAppli(message)
    Message message;
{
    printf("Quit application\n");
    exit(0);
}

static void EndAppli(message)
    Message message;
{
    printf("End application\n");
}
```

```

static void MsgError(message)
    Message message;
{
    printf("Bad received message\n");
}

static char *TheAppli;
static char *TheType;
static char *TheMsg;

/* ParseMessage is executed when a message is received */
static void ParseMessage(message)
    Message message;
{
    int lappli, ltype, lmsg;

    lappli = strlen(message.tableau[0]);
    if ((TheAppli = (char *)malloc((unsigned)sizeof(char)*(lappli + 1)))
        == NULL) {
        return;
    }
    strcpy(TheAppli,message.tableau[0]);

    ltype = strlen(message.tableau[3]);
    if ((TheType = (char *)malloc((unsigned)sizeof(char)*(ltype + 1)))
        == NULL) {
        return;
    }
    strcpy(TheType,message.tableau[3]);

    lmsg = strlen(message.tableau[4]);
    if ((TheMsg = (char *)malloc((unsigned)sizeof(char)*(lmsg + 1)))
        == NULL) {
        return;
    }
    strcpy(TheMsg,message.tableau[4]);
}

static int find(s,n,t)
    char *s;
    int n;
    char **t;
{
    int i;
    for (i=0; i<n; i++)
        if (!strcmp(s,t[i])) return(i);
    return(-1);
}

int main(argc, argv)
    unsigned int argc;
    char **argv;
{
    int igeci;

```

```

int p1, p2;
char myhost[128];
/* Scilab application to execute */
char *scilex = "/usr/local/lib/scilab-2.4/bin/scilex";

igeci = find("-pipes",argc,argv);
if (igeci == -1) exit(1);

p1 = atoi(argv[igeci+1]); p2 = atoi(argv[igeci+2]);

/* Intialization of communications */
init_messages(tb_messages,p1,p2);

/* Get the name of my computer */
gethostname(myhost,128);

/* Execute Scilab with name "Scilab" on my local host */
envoyer_message_parametres_var(ID_GeCI,
                               MSG_LANCER_APPLI,
                               "Scilab",
                               myhost,
                               scilex,
                               INS_ID_PIPES,
                               NULL);

/* Link THIS application with "Scilab" */
envoyer_message_parametres_var(ID_GeCI,
                               MSG_CREER_LIAISON,
                               identificateur_appli(),
                               "Scilab",NULL);

/* Link "Scilab" with THIS application */
envoyer_message_parametres_var(ID_GeCI,
                               MSG_CREER_LIAISON,
                               "Scilab",
                               identificateur_appli(),NULL);

/* Loop waiting for messages */
while (1) {
    scanner_messages();
    if (TheType != NULL) {
        printf("Message received from %s\n",TheAppli);
        printf("    type: %s\n",TheType);
        printf("    message: %s\n",TheMsg);
        TheAppli = NULL; TheType = NULL; TheMsg = NULL;
    }
}
}
/*****

```

The second program has everything to communicate with another application linked to GeCI, Scilab for instance. It waits for messages and print them. We call the file of this program `beta.c` and we suppose that the main directory of Scilab, <Scilab directory> is `/usr/local/lib/scilab-2.4`.

Compile this program with:

```
cc -o beta beta.c /usr/local/lib/scilab-2.4/libs/libcomm.a
```

Then, being in the directory where beta lies, execute Scilab and issue the following Scilab command:

```
// get host name of my computer
h=unix_g("hostname")
// execute program beta from Scilab and give it "Beta" as a name
ExecAppli("beta",h,"Beta")
// create a link from Scilab to
CreateLink("SELF","Beta")
// send a message to "Beta"
SendMsg("Hi","How are you?")
```

On the console you must seen (written by beta):

```
Message received from Scilab
  type: Hi
  message: How are you?
```

”beta.c” is given below and in the directory docs/comm of Scilab distribution.

```
/* **** */
#include <stdio.h>
#include <string.h>

/* Communications headers */
#include "/usr/local/lib/scilab-2.4/routines/libcomm/libCalCom.h"
#include "/usr/local/lib/scilab-2.4/routines/libcomm/libCom.h"

static void QuitAppli();
static void EndAppli();
static void ParseMessage();
static void MsgError();

/* Known messages */
static actions_messages tb_messages[]={
    {ID_GeCI,MSG_QUITTER_APPLI,NBP_QUITTER_APPLI,QuitAppli},
    {ID_GeCI,MSG_FIN_APPLI,NBP_FIN_APPLI,EndAppli},
    {NULL,MSG_DISTRIB_LISTE_ELMNT,NBP_DISTRIB_LISTE_ELMNT,ParseMessage},
    {NULL,NULL,0,MsgError}};

static void QuitAppli(message)
    Message message;
{
    printf("Quit application\n");
    exit(0);
}

static void EndAppli(message)
    Message message;
{
    printf("End application\n");
}

static void MsgError(message)
    Message message;
```

```

{
    printf("Bad received message\n");
}

static char *TheAppli;
static char *TheType;
static char *TheMsg;

/* ParseMessage is executed when a message is received */
static void ParseMessage(message)
    Message message;
{
    int lappli, ltype, lmsg;

    lappli = strlen(message.tableau[0]);
    if ((TheAppli = (char *)malloc((unsigned)sizeof(char)*(lappli + 1)))
        == NULL) {
        return;
    }
    strcpy(TheAppli,message.tableau[0]);

    ltype = strlen(message.tableau[3]);
    if ((TheType = (char *)malloc((unsigned)sizeof(char)*(ltype + 1)))
        == NULL) {
        return;
    }
    strcpy(TheType,message.tableau[3]);

    lmsg = strlen(message.tableau[4]);
    if ((TheMsg = (char *)malloc((unsigned)sizeof(char)*(lmsg + 1)))
        == NULL) {
        return;
    }
    strcpy(TheMsg,message.tableau[4]);
}

static int find(s,n,t)
    char *s;
    int n;
    char **t;
{
    int i;
    for (i=0; i<n; i++)
        if (!strcmp(s,t[i])) return(i);
    return(-1);
}

int main(argc, argv)
    unsigned int argc;
    char **argv;
{
    int igeci;
    int p1, p2;

```

```

igeci = find("-pipes",argc,argv);
if (igeci == -1) exit(1);

p1 = atoi(argv[igeci+1]); p2 = atoi(argv[igeci+2]);

/* Intialization of communications */
init_messages(tb_messages,p1,p2);

/* Loop waiting for messages */
while (1) {
    scanner_messages();
    if (TheType != NULL) {
        printf("Message received from %s\n",TheAppli);
        printf("    type: %s\n",TheType);
        printf("    message: %s\n",TheMsg);
        TheAppli = NULL; TheType = NULL; TheMsg = NULL;
    }
}
}
/*****

```

Contents

1	Functions of the Communication Toolbox	1
2	Creating applications to communicate with Scilab	2
2.1	Communication between remote hosts	5
2.2	Examples	6