Mobile-C
– A Multi-Agent Platform for Mobile C/C++ Agents

User's Guide

Version 1.9

Harry H. Cheng

Mobile-C User's Guide version 1.9 prepared by:

Yu-Cheng Chou
David Ko

August 30, 2007

Major Contributors (in alphabetical order)

Mobile-C is developed with idea, vision, and design
by Professor Harry H. Cheng

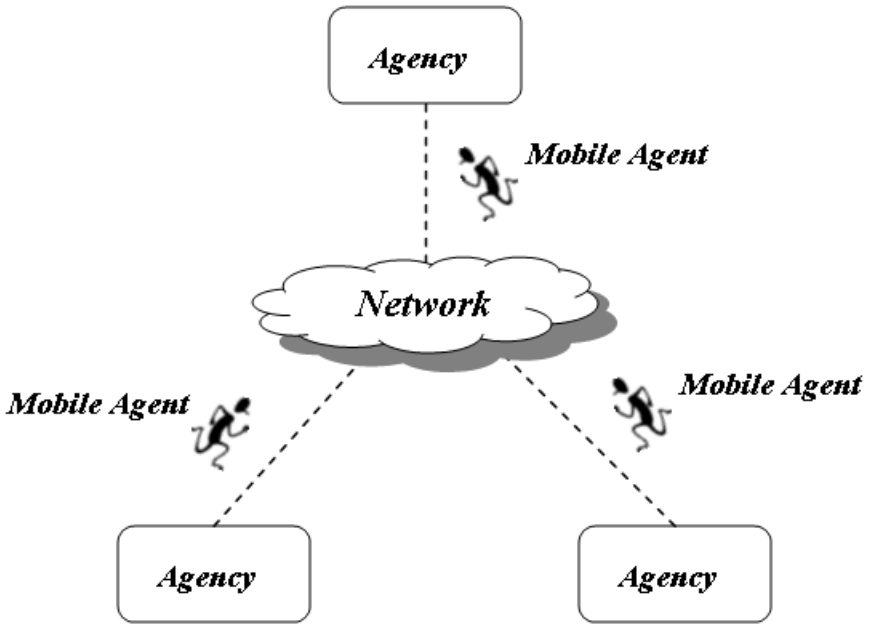
People who helped to make Mobile-C the real thing
(if you noticed that some names are missing,
please mail to mobilec@iel.ucdavis.edu)

Name	Company (by the moment of contribution)	Remarks
Bertocco,Cristian cbertocco@dei.unipd.it	Univ. of California, Davis	Design and implementation of encryption for security in Mobile-C
Chen, Bo bochen@mtu.edu	Univ. of California, Davis	Design and implementation of Mobile-C
Chou, Yucheng cycchou@ucdavis.edu	Univ. of California, Davis	Design and implementation of Mobile-C library
Ko, David, dko@ucdavis.edu	Univ. of California, Davis	Design and implementation of Mobile-C library
Linz, David ddlinz@gmail.com	Univ. of California, Davis	Design and implementation of Mobile-C
Nestinger, Stephan, thestinger@ucdavis.edu	Univ. of California, Davis	Webmaster of http://www.mobilec.org

Copyright

Abstract

Agent technology is emerging as an important concept for the development of distributed complex systems. A number of mobile agent systems have been developed in the last decade. However, most of them were developed to support only Java mobile agents. Furthermore, many of them are standalone platforms. In other words, they were not designed to be embedded in a user application to support code mobility. In order to provide distributed applications with code mobility, this article presents a mobile agent library, the Mobile-C library. The Mobile-C library is supported in various operating systems including Windows, Unix, and real-time OS. It has a small footprint to meet the stringent memory capacity for a variety of mechatronic and embedded systems. This mobile agent library allows Mobile-C, a mobile agent platform, to be embedded in a program to support C/C++ mobile agents. Functions in this library facilitate the development of a multi-agent system that can easily interface with a variety of hardware devices.



Contents

Part I

User's Guide for Mobile-C Library

Chapter 1

Introduction

Parallel and distributed computing [1] [2] are widely used in scientific and engineering fields, especially for time-critical or time-consuming tasks. Parallel computing is typically carried out in dedicated multiprocessors with a central clock and shared memory. On the other hand, distributed computing is decentralized parallel computing, using two or more computers communicating over a network to accomplish a common objective or task. It is similar to computer clustering with the main difference being a wide geographic dispersion of the resources. In addition to the main difference, the types of hardware, programming languages, operating systems and other resources may vary drastically as well in distributed computing.

Although the processing speed of networked computers is typically not as fast as that of a dedicated parallel computer, networked computers are less expensive and more broadly available. Due to the rapid improvement in network hardware and software that makes distributed computing faster, more broadly available, and easier-to-implement than before, there are more and more research investigations nowadays targeting or exploiting this low-end, decentralized parallel computing. Meanwhile, as the scale of distributed applications rapidly expands, there is an increasing demand for the code mobility.

Agent technology can significantly enhance the design and analysis of problem domains under the following three conditions [3]: (1) the problem domain is geographically distributed; (2) the subsystems exist in a dynamic environment; (3) the subsystems need to interact with each other more flexibly. Mobile agents are software components that can travel between different execution environments [4]. Mobile agents can be created dynamically during runtime and dispatched to source systems to perform tasks with the most updated code. Therefore, the mobility of mobile agents provides distributed applications with significant flexibility and adaptability which are both essential to satisfy the dynamically changing requirements and conditions in a distributed environment.

Most of the mobile agent systems were developed to support only Java mobile agents. Furthermore, many of them are standalone platforms. In other words, they were not designed to be embedded in a user application to support code mobility. Mobile-C [5] [6] [7] [8] was originally developed as a standalone, IEEE Foundation for Intelligent Physical Agents (FIPA) compliant mobile agent platform with a primary intention to fit applications where low-level hardware gets involved, such as networked mechatronic and embedded systems. Since most of these systems are written in C/C++, Mobile-C uses C/C++ as the mobile agent language for easy interfacing with control programs and underlying hardware. In addition, Mobile-C uses an embeddable C/C++ interpreter – Ch, originally developed by Cheng [9] [10] [11], to support the execution of C/C++ mobile agent code.

In order to provide distributed applications with code mobility, this user's guide presents a mobile agent library, the Mobile-C library. The Mobile-C library is supported in various operating systems including Windows, Unix, and real-time OS. It has a small footprint to satisfy the small memory requirement for a variety of mechatronic and embedded systems. This mobile agent library allows Mobile-C to be embedded

in a program to support C/C++ mobile agents. The API functions in this library facilitate the development of a multi-agent system that can easily interface with a variety of hardware devices.

Chapter 2

Mobile-C Library Installation

This chapter describes the prerequisites to install the Mobile-C library and the installation steps for both Unix and Windows operating systems.

2.1 Prerequisites

This user's guide assumes all necessary software packages are installed correctly and function. The software packages required to successfully install the Mobile-C library include:
(1) Ch version 6.0.0 or greater: It can be obtained from http://www.softintegration.com
(2) Embedded Ch version 6.0.0 or greater: It can be obtained from http://www.softintegration.com
(3) mxml-2.2.2: It is packaged with the Mobile-C library, but may be obtained from http://www.easysw.com/ mike/mxml/

2.2 Installation on Unix

2.2.1 Install the Mobile-C library

The following commands will install the Mobile-C library in the system directory, which is usually '/usr/local/lib' or '/usr/lib' depending on your system. By default, the Mobile-C library created contains both shared and static versions, which are 'libmc.so.0.0.0' and 'libmc.a', respectively. The header file, libmc.h, used in the C/C++ binary space will be placed in the system directory, which is usually 'usr/local/include' or '/usr/include' depending on your system.

```
cd <MCPACKAGE>/src
./configure
make
make install
```

Note that these commands will automatically build mxml-2.2.2 and xyssl-0.7, both of which are packaged with Mobile-C, but will not install these libraries. The Mobile-C libraries only need these libraries to compile, but does not need them installed in order to run.

Also note that the above commands will automatically compile all the included demos automatically after compiling the Mobile-C library. The demos will run even if the 'make install' step is omitted.

The '–prefix' option can be used to specify the home directory to install the Mobile-C files, as shown in the following commands.

```
cd <MCPACKAGE>/src
./configure --prefix=<MCHOME>
make
make install
```

<MCHOME> is the installation directory for the Mobile-C library and header file.
The library files 'libmc.so.0.0.0' and 'libmc.a' will be installed in <MCHOME>/lib, and the header file 'libmc.h' will be placed in <MCHOME>/include.

2.3 Installation on Windows

2.3.1 Build the mxml-2.2.2 Library

If the mxml-2.2.2 library has not been installed on your system, it can be built from the '<MCPACKAGE>/src/mxml-2.2.2' directory with the following steps in Microsoft Visual Studio .NET 2003.

1. Open the project file loctaed at '<MCPACKAGE>/src/mxml-2.2.2/vcnet/mxml.vcproj'.
2. Click on 'Build Solution' in the 'Build' menu.
3. The library 'mxml.lib' will be created in the '<MCPACKAGE>/src/mxml-2.2.2' directory.

2.3.2 Build the Mobile-C library

The Mobile-C library can be built with the following steps in Microsoft Visual Studio .NET 2003.

1. Open the project file located at '<MCPACKAGE>/src/win32/mc_lib_win32.vcproj'.
2. Click on 'Build Solution' in the 'Build' menu.
3. The library 'libmc.lib' will be created in the '<MCPACKAGE>/src/win32' directory.

Chapter 3

Sample Application Programs

This chapter describes how to compile two sample application programs, 'mc_server.c' and 'mc_client.c', shown as Program 1 and Program 2, on Unix and Windows operating systems, and illustrates these two programs as well.

3.1 Compilation on Unix

All the demo programs are compiled automatically in the Unix version.

3.2 Compilation on Windows

The sample program 'mc_server.c' located in the directory '<MCPACKAGE>/demos/mc_sample_app' can be complied to create the executable 'mc_server.exe' with the following steps in Microsoft Visual Studio .NET 2003.

1. Open the project file loctaed at '<MCPACKAGE>/demos/mc_sample_app/mc_server_win32/mc_server_win32.vcproj'.
2. Click on 'Build Solution' in the 'Build' menu.
3. The executable 'mc_server.exe' will be created in the '<MCPACKAGE>/demos/mc_sample_app' directory.

Similarly, the following steps can be used to compile the other program 'mc_client.c' located in the directory '<MCPACKAGE>/demos/mc_sample_app' to create the executable 'mc_client.exe' in Microsoft Visual Studio .NET 2003.

1. Open the project file loctaed at '<MCPACKAGE>/demos/mc_sample_app/mc_client_win32/mc_client_win32.vcproj'.
2. Click on 'Build Solution' in the 'Build' menu.
3. The executable 'mc_client.exe' will be created in the '<MCPACKAGE>/demos/mc_sample_app' directory.

3.3 Overview of Sample Application Programs

Program 1 starts an agency that is capable of receiving mobile agents and executing mobile agent code.

```
#include <libmc.h>
#include <stdio.h>
int main()
{
```

The header file **libmc.h** is included at the beginning of the program. It defines all the data types, macros and function prototypes for the Mobile-C library.

```
    MCAgency_t agency;
    int local_port = 5130;
```

The variable $agency$, of type **MCAgency_t**, is a handle that contains information of an agency.

```
    agency = MC_Initialize(local_port, NULL);
```

MC_Initialize() takes an integer and the address of an **MCAgencyOptions_t** varibale as its two parameters. An **MCAgencyOptions_t** variable is a structure that contains information about which threads to be activated and the default agent status specified by a user. Here, a **NULL** pointer is passed to **MC_Initialize()** as the second parameter to start an agency with default settings for active threads and the default agent status. A local agency will be initialized to listen on port **5130** specified by the variable $local\_port$.

```
    if(MC_Wait(agency) != 0) {
        MC_End(agency);
        return -1;
    }

    return 0;
}
```

The agency waits indefinitely for a mobile agent by the function **MC_Wait()** on success. Otherwise, the function **MC_End()** will be called to terminate the agency.

Program 2 starts an agency that sends a mobile agent to a remote agency.

```
#include <libmc.h>

int main() {
    MCAgency_t agency;
    int local_port = 5050;
    int server_port = 5130;
    char *file_name = "./test1.xml";
    char *server_name = "localhost";

    agency = MC_Initialize(local_port, NULL);

    MC_SendAgentMigrationMessageFile(agency, file_name, server_name, server_port);

    /* Note: We need to sleep here to ensure that the outgoing message has had
     * enough time to be handled by the ACC. */
    sleep(2);
    printf("Terminating...\n");
    MC_End(agency);

    return 0;
}
```

```
#include <libmc.h>

int main() {
    MCAgency_t agency;
    int local_port = 5130;

    agency = MC_Initialize(local_port, NULL);

    if(MC_Wait(agency) != 0) {
        MC_End(agency);
        return -1;
    }

    return 0;
}
```

Program 1: mc_server.c, located at mobilec/demos/mc_sample_app/mc_server.c.

In Mobile-C, a mobile agent message is an Agent Communication Language (ACL) message in Extensible Markup Language (XML) format. **MC_SendACLMessageFile()** takes an **MCAgency_t** variable, the path to a mobile agent message file, the name of the host on which a remote agency is running, and the port number on which a remote agency is listening as its four parameters. Here, **MC_SendACLMessageFile()** sends the mobile agent message saved as **test1.xml** in current directory to the remote agency running on host **iel2.engr.ucdavis.edu** and listening on port **5130**.

3.4 Execution of Sample Applications

Run the following commands from a text terminal on the server machine, in this case **iel2.engr.ucdavis.edu**, to start an agency listening on port **5130**.

```
cd <MCPACKAGE>/demos/mc_sample_app
./mc_server
```

Next, run the following commands from a text terminal on the client machine, in this case **bird1.engr.ucdavis.edu**, to start an agency listening on port **5050** and send the mobile agent message **test1.xml**, shown as Program 3, to the remote agency running on host **iel2.engr.ucdavis.edu** and listening on port **5130**.

```
cd <MCPACKAGE>/demos/mc_sample_app
./mc_client
```

After the mobile agent message is received and the mobile agent code is executed, the string **Hello World!** should be printed to the text terminal on the server machine.

```
#include <libmc.h>

int main() {
    MCAgency_t agency;
    int local_port = 5050;
    int server_port = 5130;
    char *file_name = "./test1.xml";
    char *server_name = "localhost";

    /* If server is not running on localhost, replace 'localhost' with the
     * fully qualified hostname of your server. i.e., if the server is running
     * on iel2.engr.ucdavis.edu, use */
    /* char *server_name = "iel2.engr.ucdavis.edu"; */

    agency = MC_Initialize(local_port, NULL);

    MC_SendAgentMigrationMessageFile(agency, file_name, server_name, server_port);

    /* Note: We need to sleep here to ensure that the outgoing message has had
     * enough time to be handled by the ACC. */
    sleep(2);
    printf("Terminating...\n");
    MC_End(agency);

    return 0;
}
```

Program 2: mc_client.c, located at mobilec/demos/mc_sample_app/mc_client.c

```
<?xml version="1.0"?>

<!DOCTYPE myMessage SYSTEM "gafmessage.dtd">

<GAF_MESSAGE>
 <MESSAGE message="MOBILE_AGENT">
  <MOBILE_AGENT>
   <AGENT_DATA>
    <NAME>mobagent1</NAME>
    <OWNER>IEL</OWNER>
    <HOME>bird1.engr.ucdavis.edu:5050</HOME>
    <TASK task="1" num="0">
     <DATA name="no-return"
           complete="0"
           server="localhost:5130">
     </DATA>
      <AGENT_CODE>
       <![CDATA[
#include <stdio.h>
int main() {
    printf("Hello World!\n");
    return 0;
}
       ]]>
      </AGENT_CODE>
    </TASK>
   </AGENT_DATA>
  </MOBILE_AGENT>
 </MESSAGE>
</GAF_MESSAGE>
```

Program 3: test1.xml, located at mobilec/demos/mc_sample_app/test1.xml.

Chapter 4

Architecture of the Mobile-C Library

The Mobile-C library allows a Mobile-C agency to be embedded in a program to support C/C++ mobile agents. In addition, the Mobile-C API gives users a full control over a Mobile-C agency embedded in a program. Therefore, the Mobile-C library not only provides a significant code mobility for distributed applications, but also facilitates the development of a multi-agent system that can easily interface with various hardware devices.

4.1 Architecture of the Mobile-C Library

Figure 4.2 illustrates the architecture of the Mobile-C library. The Mobile-C library allows a Mobile-C agency to be embedded in a program to support C/C++ mobile agents. A Mobile-C agency refers to a mobile agent platform within which mobile agents exist and operate. The Mobile-C API gives users a full control over a Mobile-C agency and its different modules.

As an FIPA compliant mobile agent platform, a Mobile-C agency comprises three FIPA normative modules, Agent Management System (AMS), Agent Communication Channel (ACC) and Directory Facilitator (DF). Two additional modules, Agent Execution Engine (AEE) and Agent Security Manager (ASM), are included in a Mobile-C agency as well. These modules provide different functionalities summarized as follows.

Agent Management System (AMS)
An AMS controls the creation, registration, execution, migration, persistence, and termination of a mobile agent. It maintains a directory of Agent Identifiers (AIDs) for registered mobile agents. Each mobile agent

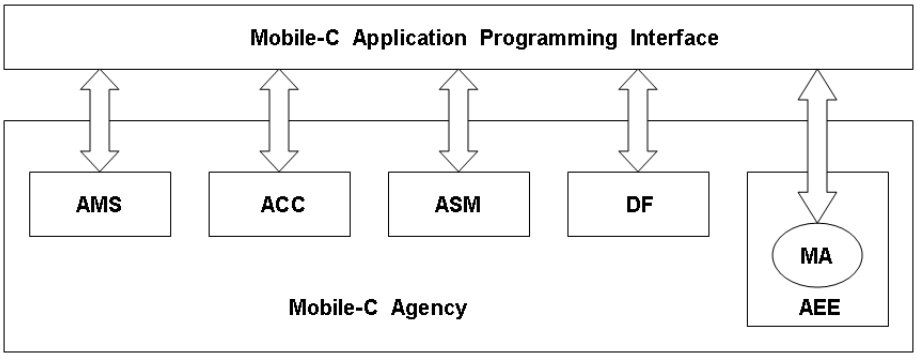


Figure 4.1: Architecture of the Mobile-C library.

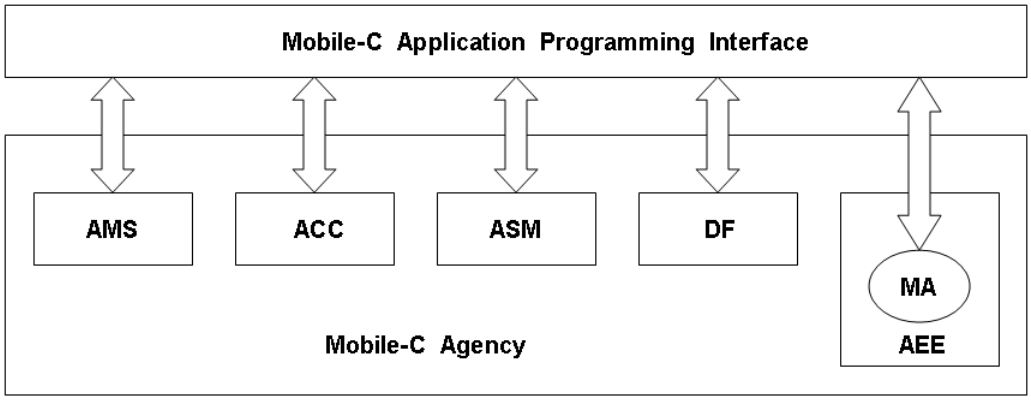


Figure 4.2: Architecture of the Mobile-C library.

must register with an AMS in order to have a valid AID.

Agent Communication Channel (ACC)

An ACC routes messages between local and remote entities. It is responsible for the interactions between distributed components, such as inter-agent communication and inter-platform agent transport. The interactions can be performed through Agent Communication Language (ACL) message exchange.

Directory Facilitator (DF)

A DF serves yellow page services. Mobile agents wishing to advertise their services should register with a DF. Visiting mobile agents can search a DF for mobile agents providing the services they desire.

Agent Execution Engine (AEE)

An AEE serves as the execution environment for mobile agent code. An AEE has to be platform independent in order to support the execution of mobile agents in a heterogeneous environment.

Agent Security Manager (ASM)

An ASM is responsible for maintaining security policies for the host system. Some sample tasks of an ASM include identifying users, protecting host resources, authenticating and authorizing mobile agents, and ensuring the security and integrity of mobile agents.

4.2 Implementation of the Mobile-C Library

Figure 4.2 shows the implementation overview of the Mobile-C library. The functionalities of each module of an agency are implemented as independent threads classified into five categories, that is, the AMS functionality threads, the ACC functionality threads, the DF functionality threads, the ASM functionality threads and the AEE threads. Each AEE thread is launched by one of the AMS functionality threads. The Mobile-C library provides API functions to specify which thread needs to be active or inactive when an agency is initialized. It also provides API functions to access the input and output data structures associated with the functionality threads. A Mobile-C agency maintains a list of synchronization variables that can be used with a group of Mobile-C functions to ensure synchronization among mobile agents and threads. The sizes of the Mobile-C static and shared libraries for Linux are about 340 KB and 290 KB, respectively. The AMS and ACC functionality threads are used as two examples in this section to demonstrate the implementation of the Mobile-C library.

The header file *libmc.h* contains definitions of all the structures and functions of the Mobile-C library. Table A.3 lists the currently implemented functions for the binary space.

Chapter 5

Interface between Binary and Mobile Agent Space

An embeddable C/C++ interpreter Ch was chosen to be the AEE in the Mobile-C library to support C/C++ mobile agents. Therefore, in order to access the variables, functions, and data sets in the mobile agent space from the binary space, Ch must be first embedded in the binary space. The function *MC_GetAgentExecEngine()* in Table A.3 returns the AEE associated with a mobile agent to the binary space. Using the AEE returned by *MC_GetAgentExecEngine()*, all of the Embedded Ch functions [12] can be called in a binary C/C++ program to access the variables, functions, and data sets defined in the mobile agent space. The Embedded Ch toolkit also allows mobile agent code to invoke C/C++ functions defined in a binary C/C++ program.

The Embedded Ch toolkit reduces the complexity of heterogeneous development environment for both embedded scripting and applications. With the consistent C/C++ code base, it can significantly reduce the effort in the software development and maintenance. Moreover, with the Embedded Ch toolkit, C/C++ applications can be extended with all the features of Ch including built-in string type for scripting. The Embedded Ch toolkit has a small footprint. The pointer and time deterministic nature of the C language provide a perfect interface with hardware in real-time systems.

5.1 Example 2: Invoke a Mobile Agent Space Function from Binary Space

This example illustrates how to call a function defined in mobile agent code by using the Mobile-C library and Embedded Ch toolkit. The mobile agent in this example is a persistent agent, which is not removed upon termination of its execution.

The client program shown in Program 4 starts a Mobile-C agency listening on port *5050* by the function *MC_Initialize()*, and sends a mobile agent to the remote agency running on host *iel2.engr.ucdavis.edu* at port *5130* through the function *MC_SendACLMessageFile()*. The filename including the full path of the mobile agent is specified from the standard input.

The mobile agent sent to the remote agency is *mobileagent_ex2.xml* shown in Program 5. The name, owner, source machine of the mobile agent are *mobileagent1*, *IEL*, and *bird1.engr.ucdavis.edu:5050*, respectively. The mobile agent is persistent since the flag *persistent* is set to 1 in Program 5. This flag can be set to 0 or removed by a user for a non-persistent mobile agent. The embedded mobile agent code is a simple but complete C program which defines the function *hello()* to be called in the server program.

As shown in Program 6, the server program starts a Mobile-C agency listening on port *5050* by the function *MC_Initialize()*, and waits for a mobile agent. The mobile agent named *mobileagentl* is found by the function *MC_FindAgentByName()*, and the AEE associated with the mobile agent is obtained by the function *MC_AgentExecEngine()*. The variable returned by *MC_AgentExecEngine()* is a Ch interpreter of

```
/* mc_sample_app.c
 *
 * This sample program uses the Mobile C library to build
 * a simple command-line driven client/server app.
 *
 * 12/15/2006
 * */

#ifndef _WIN32
#include <libmc.h>
#else
#include <libmc.h>
#endif
#include <stdio.h>
#ifndef _WIN32
#include <unistd.h>
#endif

int main(int argc, char *argv[])
{
    /* Init the agency */
    MCAgency_t agency;
    agency = MC_Initialize(
            5051,
            NULL);

    MC_SendAgentMigrationMessageFile(
            agency,
            "test1.xml",
            "localhost",
            5050 );
#ifndef _WIN32
    sleep(1);
#else
    Sleep(1000);
#endif
    MC_SendAgentMigrationMessageFile(
            agency,
            "test2.xml",
            "localhost",
            5050 );

    return 0;
}
```

Program 4: client.c. This file may be found at mobilec/demos/persistent_example/.

data type *ChInterp_t*. This variable is the first parameter for all of the Embedded Ch functions. The function *hello()* defined in the mobile agent code is invoked by the Embedded Ch function *Ch_CallFuncByName()*.

There are several different methods to call functions in mobile agent space from the binary space using the Embedded Ch API. Here we describe the function *Ch_CallFuncByName()* used in Program 6. With *Ch_CallFuncByName()*, a function defined in the mobile agent space can be called by its name. The prototype of *Ch_CallFuncByName()* is shown as follows.

int Ch_CallFuncByName(ChInterp_t interp, char *name, void *retval, ...);

The first argument is an instance of the Ch interpreter. The second argument is a string containing the

```
<?xml version="1.0"?>

<!DOCTYPE myMessage SYSTEM "gafmessage.dtd">

<GAF_MESSAGE>
  <MESSAGE message="MOBILE_AGENT">
    <MOBILE_AGENT>
     <AGENT_DATA>
       <NAME>mobagent1</NAME>
       <OWNER>IEL</OWNER>
       <HOME>localhost:5051</HOME>
       <TASK task="1" num="0">
       <DATA persistent="1" number_of_elements="0" name="no-return" complete="0" server="localhost:50
       </DATA>
     <AGENT_CODE>
          <![CDATA[
#include <stdio.h>
struct arg_struct{
    int a;
    int b;
};
int main()
{
    printf("The sample persistent agent has now arrived.\n");
    return 0;
}

int hello(struct arg_struct* arg)
{
    printf("Hello!!!\n");
    printf("This text is being generated from within the 'hello()' function!\n");
    printf("I received arguments of value %d %d.\n", arg->a, arg->b);
    return 4;
}
        ]]>
        </AGENT_CODE>
      </TASK>
      </AGENT_DATA>
    </MOBILE_AGENT>
  </MESSAGE>
</GAF_MESSAGE>
```

Program 5: test1.xml. This file may be found at mobilec/demos/persistent_example/.

name of the function to be called. The function name is associated with a function defined in mobile agent code. The third argument is a pointer containing the address of the return value of the called function. If the called function takes any arguments, the arguments should be listed after the third argument, *retval*. *Ch_CallFuncByName()* returns zero on successful execution or non-zero on failure.

Figure 5.1 shows the output when the binary file *server* compiled from Program 6 was executed. The string *Hello World!* generated from the function *hello()* was printed to the screen when the Enter key was pressed after the mobile agent had arrived.

```
/* mc_sample_app.c
 *
 * This sample program uses the Mobile C library to build
 * a simple command-line driven client/server app.
 *
 * 12/15/2006
 * */
#ifndef _WIN32
#include <libmc.h>
#else
#include <libmc.h>
#endif
#include <embedch.h>
#include <stdio.h>
#ifndef _WIN32
#include <unistd.h>
#endif

int main(int argc, char *argv[])
{
    MCAgent_t agent;
    ChInterp_t interp;
    int retval;
    int arg[2];
    /* Init the agency */
    MCAgency_t agency;
    agency = MC_Initialize(
            5050,
            NULL);

    printf("Please press 'enter' once the sample agent has arrived.\n");
    getchar();
    agent = MC_FindAgentByName(agency, "mobagent1");
    if (agent == NULL) {
        printf("Could not find agent!\n");
        exit(0);
    }
    /*interp = MC_GetAgentExecEngine(agent);
    Ch_CallFuncByName(interp, "hello", &retval); */
    arg[0] = 50;
    arg[1] = 51;
    MC_CallAgentFunc(
            agent,
            "hello",
            &retval,
            arg);
    printf("Value of %d was returned.\n", retval);
#ifndef _WIN32
sleep(2);
#else
Sleep(2000);
#endif
    return 0;
}
```

Program 6: host.c. This file may be found at mobilec/demos/persistent_example/.

```
$ ./server
Welcome to Mobile-C version 1.8.3 -- University of California, Davis, 2007
Please press 'enter' once the persistent agent has arrived.
The persistent agent has now arrived.

Hello World!
The above text is generated from the function hello().
```

Figure 5.1: Output from the binary server program.

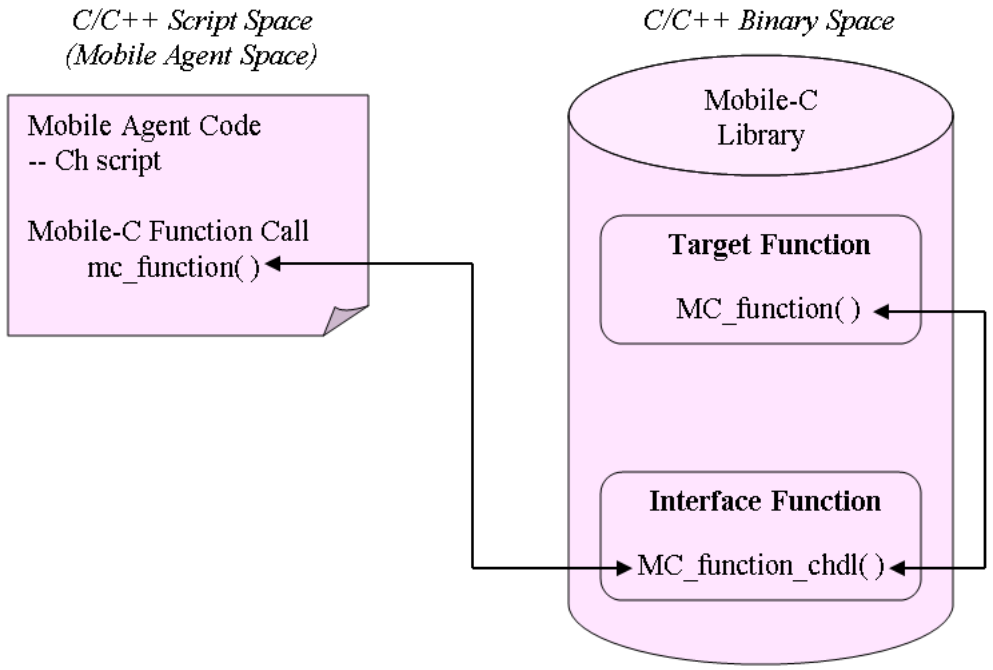


Figure 6.1: Interface of mobile agent code with the Mobile-C library.

Chapter 6

Extend Mobile-C Functionality to Mobile Agent Space

In order to allow mobile agent code to call user defined routines and access data sets defined in the binary space, as well as control other mobile agents defined in the mobile agent space through the Mobile-C API functions, the Mobile-C functionality has to be extended into the mobile agent space. We integrated Ch with the Mobile-C library to provide access to some Mobile-C functionalities.

Figure 6.1 shows how mobile agent code interfaces with the Mobile-C library. When the function *mc_function()* is called in mobile agent code, Ch searches the corresponding interface function *MC_function_chdl()* in the Mobile-C library, and passes arguments to it by calling the function. Subsequently, the interface function *MC_function_chdl()* invokes the target function *MC_function()*, and passes the return value back to the mobile agent space [12].

```
<?xml version="1.0"?>

<!DOCTYPE myMessage SYSTEM "gafmessage.dtd">

<GAF_MESSAGE>
  <MESSAGE message="MOBILE_AGENT">
    <MOBILE_AGENT>
     <AGENT_DATA>
       <NAME>mobagent1</NAME>
       <OWNER>IEL</OWNER>
       <HOME>localhost:5051</HOME>
       <TASK task="1" num="0">
       <DATA dim="0" name="no-return" complete="0" server="localhost:5050">
       </DATA>
     <AGENT_CODE>
          <![CDATA[
#include <stdio.h>
#include <unistd.h>
int main()
{
    printf("The sample persistent agent has now arrived.\n");
    while(1) {
        printf("Hello.\n");
        /* Sleep for 1 second */
        usleep(1000000);
    }
    return 0;
}
        ]]>
        </AGENT_CODE>
      </TASK>
      </AGENT_DATA>
    </MOBILE_AGENT>
  </MESSAGE>
</GAF_MESSAGE>
```

Program 7: test2.xml. This is the agent to be terminated. The code may be found in the mobilec/demos/persistent_example/ directory.

The prototypes of Mobile-C functions used in the mobile agent space are declared in *mob_agent.c* through an Embedded Ch function, *Ch_DeclareFunc()*. The data type, *MCAgent_t*, used as a parameter or return value by certain Mobile-C functions for the mobile agent space is also defined in *mob_agent.c* by two Embedded Ch functions, *Ch_DeclareVar()* and *Ch_DeclareTypedef()* [12]. Table B.2 lists the currently implemented functions for the mobile agent space. Two examples are used to demonstrate the applications and features of the Mobile-C functionality in the mobile agent space.

6.1 Example 3: Terminate Mobile Agent Execution from Mobile Agent Space

This example demonstrates how to send a mobile agent to terminate the execution of another currently running mobile agent. These two mobile agents belong to independent mobile agent spaces.

The server and client programs used in this example are the same as Programs 1 and 4, respectively. The first mobile agent sent to the remote agency is *mobileagent1_ex3.xml* shown in Program 7. The execution of the mobile agent code will repeatedly print a string *Hello World!* to the screen every sec-

```
<?xml version="1.0"?>

<!DOCTYPE myMessage SYSTEM "gafmessage.dtd">

<GAF_MESSAGE>
  <MESSAGE message="MOBILE_AGENT">
    <MOBILE_AGENT>
     <AGENT_DATA>
       <NAME>mobagent3</NAME>
       <OWNER>IEL</OWNER>
       <HOME>localhost:5050</HOME>
       <TASK task="1" num="0">
       <DATA dim="0" name="no-return" complete="0" server="localhost:5050">
       </DATA>
     <AGENT_CODE>
          <![CDATA[
#pragma package <chmobilec>
printf("At the very beginning of code.");
#include <stdio.h>
int main()
{
    MCAgent_t tmp;
    printf("The sample persistent agent has now arrived.\n");
    tmp = mc_FindAgentByName("mobagent1");
    printf("Agent mobagent1 is at address %x\n", tmp);
    if (tmp == NULL) {
        printf("Agent not found. Terminating...\n");
        return 0;
    }
    mc_TerminateAgent(tmp);
    return 0;
}
        ]]>
        </AGENT_CODE>
      </TASK>
      </AGENT_DATA>
    </MOBILE_AGENT>
  </MESSAGE>
</GAF_MESSAGE>
```

Program 8: test3.xml. This is the agent which terminates the execution of the agent in Program 7. The code may be found in the mobilec/demos/persistent_example/ directory.

ond. The second mobile agent sent to the remote agency is *mobileagent2_ex3.xml* shown in Program 8. The function *mc_FindAgentByName()* returns a variable of type *MCAgent_t* as a handle to a mobile agent. The mobile agent code embedded in *mobileagent2_ex3.xml* finds a mobile agent named *mobileagent1* by the function *mc_FindAgentByName()* and terminates the execution of *mobileagent1* by the function *mc_TerminateAgent()*.

6.2 Example 4: Invoke a Registered Service from Mobile Agent Space

This example demonstrates how to send a mobile agent to invoke a service provided by a persistent mobile agent registered with the DF. These two mobile agents belong to independent mobile agent spaces.

The server and client programs used in this example are the same as Programs 1 and 4, respectively.

Program 9: mobileagent1_ex4.xml.

Program 10: mobileagent2_ex4.xml.

The first mobile agent sent to the remote agency is *mobileagent1_ex4.xml* shown in Program 9. The execution of the mobile agent code will register two services with the remote DF through the function mc_RegisterService(). The two services are *addition* and *subtraction* which provide addition and subtraction of two integers, respectively. These services also refer to the functions defined in the mobile agent code. The function *mc_RegisterService()* takes three parameters. An *MCAgent_t* type variable is the first parameter. A system variable of type *MCAgent_t*, *mc_current_agent*, is used as the first parameter when services for the current mobile agent are registered, as illustrated in Program 9. The system variable *mc_current_agent* is declared in *mob_agent.c* of the Mobile-C source code using the function *Ch_DeclareVar()* to hold the current mobile agent. An array of pointer to character and an integer are the second and third parameters, respectively. The array holds the name of the services whereas the integer denotes the number of the services to be registered.

The second mobile agent sent to the remote agency is *mobileagent2_ex4.xml* shown in Program 10. The function *mc_SearchForService()* takes five parameters. The first parameter is the name of the service to be found. The second parameter is the address of an array of ponter to character that holds the names of all the mobile agents with the desired service. Likewise, the third parameter is the address of an array of pointer to charater that holds the desired service name associated with all the found mobile agents. The fourth parameter is the address of a one-dimensional integer array that holds the IDs of all the mobile agents with the desired service. The last parameter is the address of an integer denoting the number of mobile agents that have been found. In this example, once the search for *addition* service is done, the first mobile agent with this service will be returned by the function *mc_FindAgentByID* with a parameter as the first element of array *agentIDs*. In this example, the first found mobile agent is *mobileagent1_ex4.xml*. The function *addition()* defined in *mobileagent1_ex4.xml* will be called through the function *mc_CallAgentFunc()* to perform addition of two integers. Since *mc_CallAgentFunc()* can only pass one argument to the invoked function, the address of a data structure with two integer members is passed to *addition()* in this example. The return value of *addition()* is assigned to the variable *retval*. The string *Result of 49 + 51 is 100.* will be printed to the screen at the end.

Chapter 7

Synchronization Support in the Mobile-C library

In a Mobile-C agency, mobile agents are executed by independent AEEs. A user might also need to design a multi-threaded application where a Mobile-C agency itself is one of the many threads that handle different tasks. The Mobile-C library provides support for synchronization among mobile agents and threads. The synchronization API functions are used to protect shared resources as well as provide a method of deterministically timing the execution of mobile agents and threads.

The internal implementation consists of a linked list of Portable Operating System Interface for UNIX (POSIX) compliant synchronization variables, namely, mutexes, condition variables, and semaphores. Each node in the linked list is a synchronization variable which is assigned or given a unique identification number. The API functions can be called from the binary or mobile agent space to initialize the synchronization variables and access them by their unique identification numbers in the linked list. The application of the Mobile-C synchronization mechanism is illustrated by the example below.

7.1 Example 5: Synchronization Using Mutex in Mobile Agent Space

The server program used in this example is the same as Program 1. The client program shown in Program 11 starts a Mobile-C agency listening on port *5050* and subsequently sends two mobile agents, *mobileagent1_ex5.xml* and *mobileagent2_ex5.xml* to the remote agency running on host *iel2.engr.ucdavis.edu* at port *5130*. The mobile agents are shown in Programs 12 and 13.

As shown in Program 12, *mobileagent1* initializes a mutex with an ID 55 using the function *mc_SyncInit()*. It uses the function *mc_MutexLock()* to lock the mutex. Once the mutex has been locked, the execution waits for 15 seconds. Afterwards, the mutex is unlocked by the function *mc_MutexUnlock()*. After waiting for 2 seconds, *mobileagent1* tries to delete the mutex while the mutex has been locked by *mobileagent2* shown in Program 13. Therefore, *mobileagent1* waits for the mutex to be unlocked by *mobileagent2*. Afterwards, it deletes the mutex by the function *mc_SyncDelete()*.

As shown in Program 13, *mobileagent2* starts with trying to lock a mutex with an ID 55 while the mutex has been locked by *mobileagent1* shown in Program 12. Therefore, *mobileagent2* waits for the mutex to be unlocked by *mobileagent1*. Afterwards, it locks the mutex, waits for 15 seconds, and unlocks the mutex.

```
#include <stdio.h>
#include <libmc.h>
#define WAIT_TIME 2
int main(int argc, char *argv[])
{
    MCAgency_t agency;
    agency = MC_Initialize(5050, NULL);

    printf("MobileC Started\n");
    printf("Sending sleep agent...\n");
    MC_SendAgentMigrationMessageFile(agency,
                "sleep.xml",
                "localhost",
                5051);
    printf("Sleeping for %d seconds.\n", WAIT_TIME);
    sleep(WAIT_TIME);
    printf("Sending wake-up agent...\n");
    MC_SendAgentMigrationMessageFile(agency,
            "wake.xml",
            "localhost",
            5051);
    sleep(2);
    MC_End(agency);
    return 0;
}
```

Program 11: mc_client.c. This file may be found in mobilec/demos/agent_mutex_example/.

```
<?xml version="1.0"?>

<!DOCTYPE myMessage SYSTEM "gafmessage.dtd">

<GAF_MESSAGE>
  <MESSAGE message="MOBILE_AGENT">
    <MOBILE_AGENT>
     <AGENT_DATA>
       <NAME>sleep_agent</NAME>
       <OWNER>IEL</OWNER>
       <HOME>localhost:5050</HOME>
       <TASK task="1" num="0">
          <DATA dim="0" name="no-return" complete="0" server="localhost:5051">
       </DATA>
  <AGENT_CODE>
          <![CDATA[
#include <stdio.h>
int main()
{
    int mutex_id;
    printf("Sleep agent has arrived.\n");
    mutex_id = mc_SyncInit(55);
    if (mutex_id != 55) {
        printf("Possible error. Aborting...\n");
        exit(1);
    }
    printf("This is agent 1.\n");
    printf("Agent 1: I am locking the mutex now.\n");
    mc_MutexLock(mutex_id);
    printf("Agent 1: Mutex locked. Perform protected operations here\n");
    printf("Agent 1: Waiting for 5 seconds...\n");
    sleep(5);
    printf("Agent 1: Unlocking mutex now...\n");
    mc_MutexUnlock(mutex_id);

    return 0;
}
        ]]>
        </AGENT_CODE>
      </TASK>
      </AGENT_DATA>
    </MOBILE_AGENT>
  </MESSAGE>
</GAF_MESSAGE>
```

Program 12: sleep.xml. This file is located in mobilec/demos/agent_mutex_example/

```
<?xml version="1.0"?>

<!DOCTYPE myMessage SYSTEM "gafmessage.dtd">

<GAF_MESSAGE>
  <MESSAGE message="MOBILE_AGENT">
    <MOBILE_AGENT>
     <AGENT_DATA>
       <NAME>wake_agent</NAME>
       <OWNER>IEL</OWNER>
       <HOME>localhost:5050</HOME>
       <TASK task="1" num="0">
          <DATA dim="0" name="no-return" complete="0" server="localhost:5051">
       </DATA>
  <AGENT_CODE>
          <![CDATA[
#include <stdio.h>
int main()
{
    int mutex_id;
    mutex_id = 55;
    printf("Agent 2: Has arrived");
    printf("Agent 2: Attempting to lock the mutex...\n");
    mc_MutexLock(mutex_id);
    printf("Agent 2: Mutex locked.\n");
    printf("Agent 2: Perform protected operations here.\n");
    mc_MutexUnlock(mutex_id);
    printf("Agent 2: Mutex Unlocked\n");
    mc_SyncDelete(mutex_id);

    return 0;
}
        ]]>
        </AGENT_CODE>
      </TASK>
      </AGENT_DATA>
    </MOBILE_AGENT>
  </MESSAGE>
</GAF_MESSAGE>
```

Program 13: wake.xml. This file is located in mobilec/demos/agent_mutex_example/

Bibliography

[1] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Reading, MA: Addison-Wesley, 1994.

[2] U. Manber, *Introduction to Algorithms - A Creative Approach*. Reading, MA: Addison-Wesley, 1989.

[3] J. L. Adler and V. J. Blue, "A Cooperative Multi-Agent Transportation Management and Route Guidance System," *Research Part C - Emerging Technologies*, Vol. 10, No. 5-6, pp. 433–454, 2002.

[4] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding Code Mobility," *IEEE Transactions on Software Engineering*, Vol. 24, No. 5, pp. 342–361, 1998.

[5] B. Chen, "Runtime Support for Code Mobility in Distributed Systems." Department of Mechanical and Aeronautical Engineering, University of California, Davis, Ph.D. dissertation, 2005.

[6] B. Chen and H. H. Cheng, "A Run-Time Support Environment for Mobile Agents," in *Proc. of ASME/IEEE International Conference on Mechatronic and Embedded Systems and Applications*, No. DETC2005-85389, Long Beach, California, 2005.

[7] B. Chen, H. H. Cheng, and J. Palen, "Mobile-C: a Mobile Agent Platform for Mobile C/C++ Agents," *Software-Practice & Experience*, Vol. 36, No. 15, pp. 1711–1733, December 2006.

[8] Mobile-C: A Multi-agent Platform for Mobile C/C++ Code, http://iel.ucdavis.edu/projects/mobilec/.

[9] H. H. Cheng, "Scientific Computing in the Ch Programming Language," *Scientific Programming*, Vol. 2, No. 3, pp. 49–75, Fall 1993.

[10] ——, "Ch: A C/C++ Interpreter for Script Computing," *C/C++ User's Journal*, Vol. 24, No. 1, pp. 6–12, Jan. 2006.

[11] *Ch — an Embeddable C/C++ Interpreter*, http://www.softintegration.com.

[12] *Embedded Ch*, SoftIntegration, Inc., http://www.softintegration.com/products/sdk/embedded_ch/.

Appendix A

Mobile-C API in the C/C++ Binary Space

The header file **libmc.h** defines all the data types, macros and function prototypes for the Mobile-C library. The header file is used in the C/C++ binary space.

Table A.1: Data types defined in **libmc.h**.

Data Type	Description
MCAgency_t	A handle containing information of an agency.
MCAgent_t	A handle containing information of a mobile agent.
MCAgencyOptions_t	A structure containing information about which thread(s) to be activated and the default agent status specified by a user.

Table A.2: Macros defined in **libmc.h**.

Macro	Description
MC_THREAD_AI	Identifier for agent initalizing thread.
MC_THREAD_AM	Identifier for agent managing thread.
MC_THREAD_CL	Identifier for connection listening thread.
MC_THREAD_MR	Identifier for message receiving thread.
MC_THREAD_MS	Identifier for message sending thread.
MC_THREAD_CP	Identifier for command prompt thread.
MC_THREAD_ALL	Identifier for all threads.
RECV_CONNECTION	Signal activated after an agency accepts a connection.
RECV_MESSAGE	Signal activated after an agency receives an ACL message.
RECV_AGENT	Signal activated after an agency receives a mobile agent.
EXEC_AGENT	Signal activated after a mobile agent is executed.
ALL_SIGNALS	Signal activated after any of the above four events occurs.
REMOTE_AGENT	Identifier for a remote mobile agent.
LOCAL_AGENT	Identifier for a local mobile agent.
RETURN_AGENT	Identifier for a return mobile agent.
WAIT_CH	Value indicating a mobile agent is waiting to be executed.
WAIT_MESSGSEND	Value indicating a mobile agent is waiting to be exported to another agency.
AGENT_ACTIVE	Value indicating a mobile agent is being executed.
AGENT_NEUTRAL	Value indicating a mobile agent is waiting for an unspecified reason.
AGENT_SUSPENDED	Value indicating a mobile agent is being suspended.
WAIT_FINISHED	Value indicating a mobile agent has been executed and is waiting to be removed.

Table A.3: Functions in the C/C++ binary space.

Function	Description
MC_AddAgent()	Add a mobile agent into an agency.
MC_ChInitializeOptions()	Set the initialization options for a Ch to be used as one AEE in an agency.
MC_CondSignal()	Signal another agent that is waiting on a condition variable.
MC_CondWait()	Cause the calling agent or thread to wait on a Mobile-C condition variable with the ID specified by the argument.
MC_End()	Terminate a Mobile-C agency.
MC_FindAgentByID()	Find a mobile agent by its ID number in an agency.
MC_FindAgentByName()	Find a mobile agent by its name in an agency.
MC_GetAgentExecEngine()	Get the AEE associated with a mobile agent in an agency.
MC_GetAgentNumTasks()	Get the number of tasks a mobile agent has.
MC_GetAgentReturnData()	Get the return data of a mobile agent.
MC_GetAgentStatus()	Get the status of a mobile agent in an agency.
MC_GetAgentType()	Get the type of a mobile agent.
MC_GetAgentXMLString()	Retrieve a mobile agent message in XML format as a character string.
MC_Initialize()	Start a Mobile-C agency and return a handle of the launched agency.
MC_MutexLock()	Lock a previously initialized Mobile-C synchronization variable as a mutex.
MC_MutexUnlock()	Unlock a locked Mobile-C synchronization variable.
MC_PrintAgentCode()	Print a mobile agent code for inspection.
MC_ResetSignal()	Reset the Mobile-C signalling system.
MC_RetrieveAgent()	Retrieve the first neutral mobile agent from a mobile agent list.
MC_RetrieveAgentCode()	Retrieve a mobile agent code in the form of a character string.
MC_SemaphorePost()	Unlock one resource from a Mobile-C semaphore.
MC_SemaphoreWait()	Allocate one resource from a Mobile-C synchronization semaphore variable.
MC_SendAgentMigrationMessage()	Send an ACL mobile agent message to a remote agency.
MC_SendAgentMigrationMessageFile()	Send an ACL mobile agent message saved as a file to a remote agency.
MC_SetAgentStatus()	Set the status of a mobile agent in an agency.
MC_SetDefaultAgentStatus()	Assign a user defined default status to all incoming mobile agents.
MC_SetThreadOff()	Deactivate a thread in an agency.
MC_SetThreadOn()	Activate a thread in an agency.
MC_SyncDelete()	Delete a previously initialized synchronization variable.
MC_SyncInit()	Initialize a new synchronization variable.
MC_TerminateAgent()	Terminate the execution of a mobile agent in an agency.
MC_Wait()	Cause the calling thread to wait indefinitely on an agency.
MC_WaitAgent()	Cause the calling thread to wait until a mobile agent is received.
MC_WaitRetrieveAgent()	Block the calling thread until a mobile agent arrives, and return the mobile agent instead of executing it.

MC_AddAgent()

Synopsis
#include <**libmc.h**>
int MC_AddAgent(**MCAgency_t** $agency$, **MCAgent_t** $agent$);

Purpose
Add a mobile agent into an agency.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
$agency$ An initialized agency handle to add an agent to.
$agent$ An initialized mobile agent.

Description
This function adds a mobile agent to an already running agency.

Example

See Also

MC_ChInitializeOptions()

Synopsis
#include <**libmc.h**>
int MC_ChInitializeOptions(**MCAgent_t** *agent*, **ChOptions_t** *options*);

Purpose
Set the initialization options for a Ch to be used as one AEE in an agency.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
agent A mobile agent associated with the Ch to be set up.
options Options for setting a Ch to be used as one AEE in an agency. **ChOptions_t** is defined as a structure as the following:

```
typedef struct ChOptions{
    int shelltype;   // shell type
    int chrc;        // TRUE to execute startup file
    char *chrcname;  // user's startup file
    char *chhome;    // Embedded Ch home directory
    char *chmtdir;   // dir for lib chmt1.dl
} ChOptions_t;
```

Description
This function sets up a Ch for executing the mobile agent code. The Ch shell type and the startup file to be used are indicated in the argument *options*. If this function is not called, the default value for ChOptions will be used to start up a Ch for running the mobile agent code.

Example

See Also

MC_CondSignal()

Synopsis
#include <**libmc.h**>
int MC_CondSignal(**int** id);

Purpose
Signal another mobile agent which is waiting on a condition variable.

Return Value
This function returns 0 if the condition variable is successfully found and signalled. It returns non-zero if the condition variable was not found.

Parameters

id	The id of the condition variable to signal.

Description
This function is used to signal another mobile agent or thread that is waiting on a Mobile-C condition variable. The function that calls **MC_CondSignal** must know beforehand the id of the condition variable which a mobile agent might be waiting on.

Example

See Also
MC_CondDelete(), MC_CondInit(), MC_CondSignal().

MC_CondWait()

Synopsis
#include <**libmc.h**>
int MC_CondWait(int id);

Purpose
Cause the calling mobile agent or thread to wait on a Mobile-C condition variable with the id specified by the argument.

Return Value
This function returns 0 upon successful wakeup or non-zero if the condition variable was not found.

Parameters

id	The id of the condition variable to signal.

Description
This function blocks until the condition variable on which it is waiting is signalled. If an invalid id is specified, the function returns 1 and does not block. The function is designed to enable synchronization possibilities between threads and mobile agents without using poll-waiting loops.

Example

See Also
MC_CondDelete(), MC_CondInit(), MC_CondSignal().

MC_End()

Synopsis
#include <**libmc.h**>
int MC_End(**MCAgency_t** $agency$);

Purpose
Terminate a Mobile-C agency.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
$agency$ A handle to a running agency.

Description
This function stops all the running threads in an agency and deallocates all the memories regarding an agency.

Example

See Also

MC_FindAgentByID()

Synopsis
#include <**libmc.h**>
MCAgent_t MC_FindAgentByID(MCAgency_t $agency$, **int** id);

Purpose
Find a mobile agent by its ID number in a given agency.

Return Value
The function returns an **MCAgent_t** object on success or NULL on failure.

Parameters
$agency$ An agency handle.
id An integer representing a mobile agent's ID number.

Description
This function is used to find and retrieve a pointer to an existing running mobile agent in an agency by the mobile agent's ID number.

Example

See Also

MC_FindAgentByName()

Synopsis
#include <**libmc.h**>
MCAgent_t MC_FindAgentByName(MCAgency_t $agency$, **const char ***$name$);

Purpose
Find a mobile agent by its name in an agency.

Return Value
The function returns an **MCAgent_t** object on success or NULL on failure.

Parameters
$agency$ An agency handle.
$name$ A character string containing the mobile agent's name.

Description
This function is used to find and retrieve a pointer to an existing running mobile agent in an agency by the mobile agent's given name.

Example

See Also

MC_GetAgentExecEngine()

Synopsis
#include <**libmc.h**>
ChInterp_t MC_GetAgentExecEngine(**MCAgent_t** $agent$);

Purpose
Get the AEE associated with a mobile agent in an agency.

Return Value
The functions returns a Ch interpreter on success and NULL on failure.

Parameters
$agent$ A valid mobile agent.

Description
This function is used to retrieve a Ch interpreter from a mobile agent. The mobile agent must be a valid mobile agent that has not been terminated at the time of this function call. The Ch interpreter may be used by the Embedded Ch API to execute functions, retrieve data, and other various tasks.

Example

See Also

MC_GetAgentNumTasks()

Synopsis
#include <**libmc.h**>
int MC_GetAgentNumTasks(MCAgent_t $agent$);

Purpose
Return the total number of tasks a mobile agent has.

Return Value
This function returns a non negative integer on success and a negative integer on failure.

Parameters
$agent$ A MobileC agent.

Description
This function returns the total number of tasks that an agent has. It counts all tasks: those that have been completed, those that are in progress, and those that have not yet started.

Example

```
int i;
MCAgent_t agent;

/* More code here */

i = MC_GetAgentNumTasks(agent);
printf("The agent has %d tasks.\n", i);
```

The previous piece of code retrieves the nuber of tasks that an agent has and prints it to standard output.

See Also

MC_GetAgentReturnData()

Synopsis
#include <**libmc.h**>
int MC_GetAgentReturnData(**MCAgent_t** $agent$, **int** $task\_num$, **void**** $data$, **int*** dim, **int**** $extent$);

Purpose
Retrieve the data from a return mobile agent.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

$agent$	A returning agent.
$task\_num$	The task for which the return data is to be retrieved.
$data$	A pointer to hold an array of data.
dim	An integer to hold the dimension of the array.
$extent$	A pointer to hold an array of extents for each dimension of the data array.

Description
This function is used to retrieve the return data of a mobile agent. Mobile agents may return single data values as well as multidimensional arrays of int, float, or double type. The first two arguments, $agent$ and $task\_num$, are input arguments which specify which mobile agent and task for which to retrieve data. The next three arguments are unallocated pointers which are modified by the function. The mobile agent's return data are stored as a single list of values in $data$. The dimension of the array is stored into dim, and the size of each dimension is stored into $extent$.

Example

```
MCAgent_t agent;
MCAgency_t agency;
double *data;
int dim;
int *extent;
int i;
int elem;

/* Agency initialization code here */

agent = MC_FindAgentByName(agency, "ReturnAgent");
MC_GetAgentReturnData(agent, 0, &data, &dim, &extent);
elem = 1;
for(i=0; i<dim; i++) {
    printf("dim %d has %d size.\n", i, extent[i]);
    elem *= extent[i];
}
printf("There are %d total elements in the multidimensional array.\n", elem);
```

The above code prints the dimension and extent of each dimension of the return data held by the agent. It only prints the data of the first task, as indicated by the second argument of function **MC_GetAgentReturnData**(), which is 0 in this example.

See Also

MC_GetAgentStatus()

Synopsis
#include <libmc.h>
int MC_GetAgentStatus(MCAgent_t $agent$);

Purpose
Get the status of a mobile agent in an agency.

Return Value
This function returns an enumerated value representing the current status of a mobile agent. The values are

0 , WAIT_CH :	Mobile agent is currently waiting to be executed.
1 , WAIT_MESSGSEND :	Mobile agent is currently waiting to be exported to another agency.
2 , AGENT_ACTIVE :	Mobile agent is currently being executed.
3 , AGENT_NEUTRAL :	Mobile agent is waiting for an unspecified reason.
4 , AGENT_SUSPENDED :	Mobile agent is currently being suspended.
5 , WAIT_FINISHED :	Mobile agent has finished execution and is waiting for removal.

Parameters

$agent$	The mobile agent from which to retrieve status information.

Description
This function gets a mobile agent's status. The status is used to determine the mobile agent's current state of execution.

Example

See Also

MC_GetAgentType()

Synopsis
#include <libmc.h>
enum MC_AgentType_e MC_GetAgentType(MCAgent_t $agent$);

Purpose
This function blocks until one of a specified number of signals is signalled.

Return Value
This function returns an enumerated value of type MC_AgentType_e.

Parameters
$agency$ A handle associated with a running agency.
$signals$ A combination of signals specified by the enum MC_Signal_e.

Description
This function is used to determine the type of agent that input argument 'agent' is. It is useful for use in determining if the agent is an active agent of type 'MOBILE_AGENT', or a return agent containing return data of type 'RETURN_AGENT'.

Example

```
MCAgent_t agent;
enum MC_AgentType_e type;

/* Code here which assign an agent to variable 'agent' */
type = MC_GetAgentType(agent);
switch(type) {
    case MOBILE_AGENT:
        printf("Received a mobile agent.\n");
        break;
    case RETURN_AGENT:
        printf("Received a return agent.\n");
        break;
    default:
        printf("Received an agent of other type.\n");
        break;
}
```

The above code determines whether a mobile agent is a return agent or a normal agent to be executed, and prints the result to the standard output.

See Also

MC_GetAgentXMLString()

Synopsis
#include <**libmc.h**>
char *MC_GetAgentXMLString(MCAgent_t $agent$);

Purpose
Retrieve a mobile agent message in XML format as a character string.

Return Value
The function returns an allocated character array on success and NULL on failure.

Parameters
$agent$ The mobile agent from which to retrieve the XML formatted message.

Description
This function retrieves a mobile agent message in XML format as a character string. The return pointer is allocated by 'malloc()' and must be freed by the user.

Example

See Also

MC_Initialize()

Synopsis
#include <libmc.h>
MCAgency_t MC_Initialize(int $port$, **MCAgencyOptions_t** $*options$);

Purpose
Start a Mobile-C agency and return a handle of the launched agency.

Return Value
The function returns an **MCAgency_t** on success and NULL on failure.

Parameters

$port$ The port number to listen on for incoming mobile agents.

$options$ The address of a structure of type **MCAgencyOptions_t** for specifying which thread(s) to be activated in an agency and setting the default agent status for incoming mobile agents. **MCAgencyOptions_t** is defined as a structure as the following:

```
typedef struct MCAgencyOptions_s{
    int threads;
    int default_agent_status;
    int modified;
} MCAgencyOptions_t;
```

Description
MC_Initialize() starts a Mobile-C agency and returns a handle of type **MCAgency_t** containing the information about the current agency. The first one specifies the port number on which an agency will listen. The second one can specify which thread(s) to be activated in an agency and the default agent status for incoming mobile agents.

Example

See Also

MC_MutexLock()

Synopsis
#include <**libmc.h**>
int MC_MutexLock(**MCAgency_t** agency, **int** id);

Purpose
This function locks a previously initialized Mobile-C synchronization variable as a mutex. If the mutex is already locked, the function blocks until it is unlocked before locking the mutex and continuing.

Return Value
This function returns 0 on success, or non-zero if the id could not be found.

Parameters

$agency$	The agency in which to find the synchronization variable to lock.
id	The id of the synchronization variable to lock.

Description
This function locks the mutex part of a Mobile-C synchronization variable. While this is primarily used to guard a shared resource, the behaviour is similar to the standard POSIX mutex locking. Note that although a MobileC synchronization variable may assume the role of a mutex, condition variable, or semaphore, once a Mobile-C synchronization variable is used as a mutex, it should not be used as anything else for the rest of its life cycle.

Example

See Also
MC_MutexUnlock(), MC_SyncInit(), MC_SyncDelete().

MC_MutexUnlock()

Synopsis
#include <**libmc.h**>
int MC_MutexUnlock(**MCAgency_t** agency, **int** id);

Purpose
This function unlocks a locked Mobile-C synchronization variable.

Return Value
This function returns 0 on success, or non-zero if the id could not be found.

Parameters

$agency$	The agency in which to find the synchronization variable to lock.
id	The id of the synchronization variable to lock.

Description
This function unlocks a Mobile-C synchronization variable that was previously locked as a mutex. If the mutex is not locked while calling this function, undefined behaviour results. Note that although a Mobile-C may act as a mutex, condition variable, or semaphore, once it has been locked and/or unlocked as a mutex, it should only be used as a mutex for the remainder of it's life cycle or unexpected behaviour may result.

Example
See the example for **MC_MutexLock()**.
See Also
MC_MutexLock(), MC_SyncInit(), MC_SyncDelete().

MC_PrintAgentCode()

Synopsis
#include <**libmc.h**>
int MC_PrintAgentCode(**MCAgent_t** $agent$);

Purpose
Print a mobile agent code for inspection.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
$agent$ The mobile agent from which to print the code.

Description
This function prints the mobile agent code to the standard output.

Example

See Also

MC_ResetSignal()

Synopsis
#include <libmc.h>
int MC_ResetSignal(MCAgency_t $agency$**);**

Purpose
This function is used to reset the Mobile-C signalling system. It is intended to be used after returning from a call to function **MC_WaitSignal()**.

Return Value
This function returns 0 on success and non-zero otherwise.

Parameters

$agency$ A handle to a running agency.

Description
This function is used to reset the Mobile-C signalling system. System signals are triggered by certain events in the Mobile-C library. This includes events such as the arrival of a new message or mobile agent, and the departure of a mobile agent, etc. If function **MC_WaitSignal()** is used to listen for one of these events, function **MC_ResetSignal()** must be called in order to allow Mobile-C to resume with it's operations.

Example

See Also
MC_WaitSignal()

MC_RetrieveAgent()

Synopsis
#include <**libmc.h**>
MCAgent_t MC_RetrieveAgent(**MCAgency_t** *agency*);

Purpose
Retrieve the first neutral mobile agent from a mobile agent list.

Return Value
The function returns an **MCAgent_t** object on success or NULL on failure.

Parameters
agency An agency handle.

Description
This function retrieves the first agent with status AGENT_NEUTRAL from a mobile agent list. If there are no mobile agents with this attribute, the return value is NULL.

Example

See Also

MC_RetrieveAgentCode()

Synopsis
#include <**libmc.h**>
char *MC_RetrieveAgentCode(**MCAgent_t** $agent$);

Purpose
Retrieve a mobile agent code in the form of a character string.

Return Value
The function returns an allocated character array on success and NULL on failure.

Parameters
$agent$ The mobile agent from which to retrieve the code.

Description
This function retrieves a mobile agent code. The return pointer is allocated by 'malloc()' and must be freed by the user.

Example

See Also

MC_SemaphorePost()

Synopsis
#include <**libmc.h**>
int MC_SemaphorePost(**MCAgency_t** agency, **int** id);

Purpose
This function unlocks one resource from a Mobile-C semaphore, increasing its count by one.

Return Value
This function returns 0 on success, or non-zero if the id could not be found or on a semaphore error.

Parameters

$agency$	The agency in which to find the synchronization variable to lock.
id	The id of the synchronization variable to lock.

Description
MC_SemaphorePost unlocks a resourse from a previously allocated and initialized Mobile-C synchronization variable being used as a semaphore. This function may be called multiple times to increase the count of the semaphore up to INT_MAX. Note that although a Mobile-C synchronization variable may be used as a mutex, condition variable, or semaphore, once it is used as a semaphore, it should only be used as a semaphore for the remainder of its life cycle.

Example

See Also
MC_SemaphoreWait(), MC_SyncInit(), MC_SyncDelete().

MC_SemaphoreWait()

Synopsis
#include <**libmc.h**>
int MC_SemaphoreWait(**MCAgency_t** agency, **int** id);

Purpose
This function allocates one resource from a Mobile-C synchronization semaphore variable.

Return Value
This function returns 0 on success, or non-zero if the id could not be found.

Parameters

$agency$	The agency in which to find the synchronization variable to lock.
id	The id of the synchronization variable to lock.

Description
This function allocates one resource from a previously allocated and initialized Mobile-C synchronization semaphore. If the semaphore resource count is non-zero, the resource is immediately allocated. If the semaphore resource count is zero, the function blocks until a resource is freed before allocating a resource and continuing. Note that although a Mobile-C synchronization variable may be used as a mutex, condition variable, or semaphore, once it is used as a semaphore, it should only be used as a semaphore for the remainder of its life cycle.

Example

See Also
MC_SemaphorePost(), MC_SyncInit(), MC_SyncDelete().

MC_SendAgentMigrationMessage()

Synopsis
#include <libmc.h>
int MC_SendAgentMigrationMessage(MCAgency_t *agency*, **char ***$message$, **char ***$hostname$, **int** *port*);

Purpose
Send an ACL mobile agent message to a remote agency.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

agency	A handle associated with an agency from which to send the ACL mobile agent message. A NULL pointer can be used to send the ACL message from an unspecified agency.
$message$	The ACL mobile agent message to be sent.
$hostname$	The hostname of the remote agency. It can be in number-dot format or hostname format, i.e., 10.20.30.40 or iel.ucdavis.edu.
port	The port number on which the remote agency is listening.

Description
This function is used to send an XML based ACL mobile agent message, which is a string, to a remote agency. This function can be used without a running local agency.

Example

See Also

MC_SendAgentMigrationMessageFile()

Synopsis
#include <libmc.h>
int MC_SendAgentMigrationMessageFile(MCAgency_t *agency*, **char ***$filename$, **char ***$hostname$, **int** *port*);

Purpose
Send an ACL mobile agent message saved as a file to a remote agency.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

agency	A handle associated with an agency from which to send the ACL mobile agent message. A NULL pointer can be used to send the ACL message from an unspecified agency.
$filename$	The ACL mobile agent message file to be sent.
$hostname$	The hostname of the remote agency. It can be in number-dot format or hostname format, i.e., 10.20.30.40 or iel.ucdavis.edu.
port	The port number on which the remote agency is listening.

Description
This function is used to send an XML based ACL mobile agent message, which is saved as a file, to a remote agency. This function can be used without a running local agency.

Example

See Also

MC_SetAgentStatus()

Synopsis
#include <**libmc.h**>
int MC_SetAgentStatus(**MCAgent_t** $agent$, **int** $status$);

Purpose
Set the status of a mobile agent in an agency.

Return Value
This function returns 0 on success and non-zero otherwise.

Parameters
$agent$ The mobile agent whose status is to be assigned.
$status$ An integer representing the status to be assinged to a mobile agent.

Description
This function sets the status of a mobile agent by one of the enumerated values listed below.

0 , WAIT_CH :	Mobile agent will wait to be executed.
3 , AGENT_NEUTRAL :	Mobile agent will wait for an unspecified reason.
4 , AGENT_SUSPENDED :	Mobile agent will be suspended.

Example

See Also

MC_SetDefaultAgentStatus()

Synopsis
#include <libmc.h>
int MC_SetDefaultAgentStatus(MCAgency_t $agency$, **int** $status$);

Purpose
Set the default status of any incoming mobile agents.

Return Value
This function returns 0 on success and non-zero otherwise.

Parameters

$agency$	A handle to a running agency.
$status$	An integer representing the status to be assinged to any incoming mobile agents as their default status.

Description
This function sets the default status of any incoming mobile agents by one of the enumerated values listed below.

0 , WAIT_CH :	Mobile agent will wait to be executed.
3 , AGENT_NEUTRAL :	Mobile agent will wait for an unspecified reason.
4 , AGENT_SUSPENDED :	Mobile agent will be suspended.

Example

See Also

MC_SetThreadOff()

Synopsis
#include <**libmc.h**>
int MC_SetThreadOff(**MCAgencyOptions_t** $*options$, **enum threadIndex_e** $thread$);

Purpose
Set a particular thread to not execute upon Mobile-C initialization.

Return Value
This function returns 0 on success and non-zero otherwise.

Parameters
$options$ An allocated MCAgencyOptions_t variable.
$thread$ A thread index.

Description
This function is used to modify the Mobile-C startup options. It is used to disable threads that may otherwise be enabled. The threads which may be modified are

MC_THREAD_AI :	Agent Initializing Thread - Create agent from incoming messages.
MC_THREAD_AM :	Agent Managing Thread - Manage active agents.
MC_THREAD_CL :	Connection Listening Thread - Listen incoming connections.
MC_THREAD_MR :	Message Receiving Thread - Handle incoming connections and recieve agent messages.
MC_THREAD_MS :	Message Sending Thread - Handle outgoing connections and send agent messages.
MC_THREAD_CP :	Command Prompt Thread - Handle an interactive user command prompt.

Example

```
MCAgencyOptions_t options;
MCAgency_t agency;

/* Turn the listen thread off. We will receive our messages
   in another method. */
MC_SetThreadOff(&options, MC_THREAD_AI);

/* Start the agency with no listen thread*/
agency = MC_Initialize(5050, &options);

/* etc ... */
```

See Also
MC_SetThreadOn()

MC_SetThreadOn()

Synopsis
#include <libmc.h>
int MC_SetThreadOn(MCAgencyOptions_t $*options$, **enum threadIndex_e** $thread$);

Purpose
Sets a particular thread to execute upon Mobile C initialization.

Return Value
This function returns 0 on success and non-zero otherwise.

Parameters
$options$ An allocated MCAgencyOptions_t variable.
$thread$ A thread index.

Description
This function is used to modify the Mobile-C startup options. It is used to enable threads that may otherwise be disabled. The threads which may be modified are

MC_THREAD_AI :	Agent Initializing Thread - Create agent from incoming messages.
MC_THREAD_AM :	Agent Managing Thread - Manage active agents.
MC_THREAD_CL :	Connection Listening Thread - Listen incoming connections.
MC_THREAD_MR :	Message Receiving Thread - Handle incoming connections and recieve agent messages.
MC_THREAD_MS :	Message Sending Thread - Handle outgoing connections and send agent messages.
MC_THREAD_CP :	Command Prompt Thread - Handle an interactive user command prompt.

Example

```
MCAgencyOptions_t options;
MCAgency_t agency;

/* Turn the command prompt thread on */
MC_SetThreadOn(&options, MC_THREAD_CP);

/* Start the agency with a command prompt on port 5050 */
agency = MC_Initialize(5050, &options);

/* etc ... */
```

See Also
MC_SetThreadOff()

MC_SyncDelete()

Synopsis
#include <libmc.h>
int MC_SyncDelete(int id);

Purpose
Delete a previously initialized synchronization variable.

Return Value
This function returns 0 on success and nonzero otherwise.

Parameters
id The id of the condition variable to delete.

Description
This function is used to delete and deallocate a previously initialized Mobile-C synchronization variable.

Example

See Also
MC_SyncInit().

MC_SyncInit()

Synopsis
#include <**libmc.h**>
int MC_SyncInit(**MCAgency_t** $agency$, **int** id);

Purpose
Initialize a new synchronization variable.

Return Value
This function returns the allocated id of the synchronization variable.

Parameters

$agency$	The agency in which the new synchronization variable should be initialized.
id	A requested synchronization variable id. A random id will be assigned if the value passed is 0 or if there is a conflicting id.

Description
This function initializes a generic Mobile-C synchonization node for use by agents and the agency. Each node contains a mutex, a condition variable, and a semaphore. Upon initialization, each variable is initialized to default values: The mutex is unlocked and the semaphore has a value of zero. Each node may be used as a mutex, condition variable, or semaphore. Though it is possible to use multiple synchronization variables in a single node, this is discouraged as it may lead to unpredictable results.

Example

See Also
MC_CondSignal(), MC_CondWait(), MC_MutexLock(), MC_MutexUnlock(), MC_SemaphorePost(), MC_SemaphoreWait(), MC_SyncDelete().

MC_TerminateAgent()

Synopsis
#include <libmc.h>
int MC_TerminateAgent(MCAgent_t $agent$);

Purpose
Terminate the execution of a mobile agent in an agency.

Return Value
The function returns 0 on success and an error code on failure.

Parameters
$agent$ A valid mobile agent.

Description
This function halts a running mobile agent. The Ch interpreter is left intact. The mobile agent may still reside in the agency in AGENT_NEUTRAL mode if the mobile agent is tagged as 'persistent', or is terminated and flushed otherwise.

Example

See Also

MC_Wait()

Synopsis
#include <**libmc.h**>
int MC_Wait(**MCAgency_t** $agency$);

Purpose
Cause the calling thread to wait indefinitely on an agency.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
$agency$ A handle associated with a running agency.

Description
This function simply waits for the agency. It must be run on a handle that is attached to an agency that has already been started with the function **MC_Initialize()**.

Example

See Also

MC_WaitAgent()

Synopsis
#include <**libmc.h**>
int MC_WaitAgent(**MCAgency_t** $agency$);

Purpose
Cause the calling thread to wait until a mobile agent is received.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
$agency$ A handle associated with a running agency.

Description
This function waits on an agency and wakes up the addition of a new mobile agent to the agency.

Example

See Also

MC_WaitRetrieveAgent()

Synopsis
#include <libmc.h>
MCAgent_t MC_WaitRetrieveAgent(MCAgency_t $agency$**);**

Purpose
Block the calling thread until a mobile agent arrives, and return the mobile agent instead of executing it.

Return Value
The function returns a mobile agent on success and a NULL on failure.

Parameters
$agency$ A handle associated with a running agency.

Description
This function waits on an agency and wakes up the addition of a new mobile agent to the agency. It will then remove the mobile agent from the agency and return it.

Example

See Also

MC_WaitSignal()

Synopsis
#include <libmc.h>
int MC_WaitSignal(MCAgency_t $agency$**, int** $signals$**);**

Purpose
This function is used to block the execution of a Mobile-C library application until the event of a signal.

Return Value
This function returns 0 on success and non-zero otherwise.

Parameters

$agency$ A handle to a running agency.
$signals$ A bitwise-or combination of signals to wait on.

Description
This function is used to block the execution of an application using the Mobile-C library until a given signal is received as specfied by the parameter $signals$. Currently implemented signals that may be waited on are:

RECV_CONNECTION :	Continue after a connection is initialized.
RECV_MESSAGE :	Continue after a message is received.
RECV_AGENT :	Continue after an agent is received.
EXEC_AGENT :	Continue after an agent is finished executing.
ALL_SIGNALS :	Continue after any one of the above events occurs.

In order to wait on a custom combination of signals, the bitwize 'or operator' may be used to specify combinations of signals.

Example

```
/* More code here. */

/* Now we wait until we receive a message or mobile agent. */
MC_WaitSignal(agency, RECV\_MESSAGE | RECV_AGENT);

/* At this point, a message or mobile agent has been received. */

/* Perform operations on the new message or mobile agent here. */

/* Resume the Mobile-C library */
MC_ResetSignal(agency);

/* More code here. */
```

The above piece of code blocks execution until either a RECV_MESSAGE or a RECV_AGENT event occurs. The function **MC_ResetSignal()** must be invoked at some point after returning from **MC_WaitSignal()**

in order for Mobile-C to resume normal operations.

See Also
MC_ResetSignal()

Appendix B

Mobile-C API in the C/C++ Script Space

The prototypes of Mobile-C functions used in the C/C++ script space are declared in **mob_agent.c** through an Embedded Ch function, **Ch_DeclareFunc()** [12]. Every arriving mobile agent will be appended to one line of code that defines a data type used as a parameter or return value for Mobile-C functions in the C/C++ script space. This line of code can be found in **mob_agent.c**. The function prototypes and defined data type shown in Tables B.1 and B.2 are listed in **agent_space_api.txt** for user's information. **mob_agent.c** and **agent_space_api.txt** can be found in directories '<MCPACKAGE>/src' and '<MCPACKAGE>/src/include', respectively. <MCPACKAGE> is the directory for the downloaded Mobile-C package containing all the source files of the library, examples, and documentation.

Table B.1: Data type for functions in the C/C++ script space.

Data Type	Description
MCAgent_t	A void pointer for a mobile agent.

Table B.2: Functions in the C/C++ script space.

Function	Description
mc_CondSignal()	Signal another agent that is waiting on a condition variable.
mc_CondWait()	Cause the calling agent or thread to wait on a Mobile C c ondition variable with the ID specified by the argument.
mc_FindAgentByID()	Find a mobile agent by its ID in an agency.
mc_FindAgentByName()	Find a mobile agent by its name in an agency.
mc_GetAgentStatus()	Get the status of a mobile agent in an agency.
mc_GetAgentXMLString()	Retrieve a mobile agent message in XML format as a character string.
mc_MutexLock()	Lock a previously initialized Mobile-C synchronization variable as a mutex.
mc_MutexUnlock()	Unlock a locked Mobile-C synchronization variable.
mc_PrintAgentCode()	Print a mobile agent code for inspection.
mc_RetrieveAgent()	Retrieve the first neutral mobile agent from the mobile agent list.
mc_RetrieveAgentCode()	Retrieve a mobile agent code in the form of a character string.
mc_SemaphorePost()	Unlock one resource from a Mobile-C semaphore.
mc_SemaphoreWait()	Allocate one resource from a Mobile-C synchronization semaphore variable.
mc_SendAgentMigrationMessage()	Send an ACL mobile agent message to a remote agency.
mc_SendAgentMigrationMessageFile()	Send an ACL mobile agent message saved as a file to a remote agency.
mc_SetAgentStatus()	Set the status of a mobile agent in an agency.
mc_SetDefaultAgentStatus()	Assign a user defined default status to all incoming mobile agents.
mc_SyncDelete()	Delete a previously initialized synchronization variable.
mc_SyncInit()	Initialize a new synchronization variable.
mc_TerminateAgent()	Terminate the execution of a mobile agent in an agency.

mc_CondSignal()

Synopsis
int mc_CondSignal(**int** id);

Purpose
Signal another mobile agent which is waiting on a condition variable.

Return Value
This function returns 0 if the condition variable is successfully found and signalled. It returns non-zero if the condition variable was not found.

Parameters pace-0.1in

id The id of the condition variable to signal.

Description
This function is used to signal another mobile agent or thread that is waiting on a Mobile-C condition variable. The function that calls **mc_CondSignal()** must know beforehand the id of the condition variable an agent may be waiting on. Note that although a MobileC synchronization variable may act as a mutex, condition variable, or semaphore, once it is used as a condition variable, it should only be used as a condition variable for the remainder of it's life cycle.

Example

See Also
mc_CondDelete(), mc_CondInit(), mc_CondSignal().

mc_CondWait()

Synopsis
int mc_CondWait(int id);

Purpose
Cause the calling mobile agent or thread to wait on a Mobile-C condition variable with the id specified by the argument.

Return Value
This function returns 0 upon successful wakeup or non-zero if the condition variable was not found.

Parameters

id	The id of the condition variable to signal.

Description
This function blocks until the condition variable on which it is waiting is signalled. If an invalid id is specified, the function returns 1 and does not block. The function is designed to enable synchronization possibilities between threads and mobile agents without using poll-waiting loops. Note that although a MobileC synchronization variable may act as a mutex, condition variable, or semaphore, once it is used as a condition variable, it should only be used as a condition variable for the remainder of it's life cycle.

Example

See Also
mc_CondDelete(), mc_CondInit(), mc_CondSignal().

mc_FindAgentByID()

Synopsis
MCAgent_t MC_FindAgentByID(int id);

Purpose
Find a mobile agent by its ID number in a given agency.

Return Value
The function returns an **MCAgent_t** object on success or NULL on failure.

Parameters

id	An integer representing a mobile agent's ID number.

Description
This function is used to find and retrieve a pointer to an existing running mobile agent in an agency by the mobile agent's ID number.

Example

See Also

mc_FindAgentByName()

Synopsis
MCAgent_t mc_FindAgentByName(const char *$name$);

Purpose
Find a mobile agent by its name in an agency.

Return Value
The function returns an **MCAgent_t** object on success or NULL on failure.

Parameters

$name$ A character string containing the mobile agent's name.

Description
This function is used to find and retrieve a pointer to an existing running mobile agent in an agency by the mobile agent's given name.

Example

See Also

mc_GetAgentStatus()

Synopsis
#include <mobilec.h>
int mc_GetAgentStatus(MCAgent_t $agent$);

Purpose
Get the status of a mobile agent in an agency.

Return Value
This function returns an enumerated value representing the current status of a mobile agent. The values are

0 , WAIT_CH :	Mobile agent is currently waiting to be executed.
1 , WAIT_MESSGSEND :	Mobile agent is currently waiting to be exported to another agency.
2 , AGENT_ACTIVE :	Mobile agent is currently being executed.
3 , AGENT_NEUTRAL :	Mobile agent is waiting for an unspecified reason.
4 , AGENT_SUSPENDED :	Mobile agent is currently being suspended.
5 , WAIT_FINISHED :	Mobile agent has finished execution and is waiting for removal.

Parameters

$agent$	The mobile agent from which to retrieve status information.

Description
This function gets a mobile agent's status. The status is used to determine the mobile agent's current state of execution.

Example

See Also

mc_GetAgentXMLString()

Synopsis
char *mc_GetAgentXMLString(MCAgent_t $agent$);

Purpose
Retrieve a mobile agent message in XML format as a character string.

Return Value
The function returns an allocated character array on success and NULL on failure.

Parameters
$agent$ The mobile agent from which to retrieve the XML formatted message.

Description
This function retrieves a mobile agent message in XML format as a character string. The return pointer is allocated by 'malloc()' and must be freed by the user.

Example

See Also

mc_MutexLock()

Synopsis
int mc_MutexLock(int id);

Purpose
This function locks a previously initialized Mobile-C synchronization variable as a mutex. If the mutex is already locked, the function blocks until it is unlocked before locking the mutex and continuing.

Return Value
This function returns 0 on success, or non-zero if the id could not be found.

Parameters

id	The id of the synchronization variable to lock.

Description
This function locks the mutex part of a Mobile-C synchronization variable. While this is primarily used to guard a shared resource, the behaviour is similar to the standard POSIX mutex locking. Note that although a Mobile-C synchronization variable may assume the role of a mutex, condition variable, or semaphore, once a Mobile-C synchronization variable is used as a mutex, it should not be used as anything else for the rest of its life cycle.

Example

```
<?xml version="1.0"?>

<!DOCTYPE myMessage SYSTEM "gafmessage.dtd">

<GAF_MESSAGE>
  <MESSAGE message="MOBILE_AGENT">
    <MOBILE_AGENT>
     <AGENT_DATA>
       <NAME>sleep_agent</NAME>
       <OWNER>IEL</OWNER>
       <HOME>localhost:5050</HOME>
       <TASK task="1" num="0">
          <DATA dim="0" name="no-return" complete="0" server="localhost:5051">
       </DATA>
  <AGENT_CODE>
          <![CDATA[
#include <stdio.h>
int main()
{
    int mutex_id;
    printf("Sleep agent has arrived.\n");
    mutex_id = mc_SyncInit(55);
    if (mutex_id != 55) {
        printf("Possible error. Aborting...\n");
        exit(1);
    }
    printf("This is agent 1.\n");
```

```
    printf("Agent 1: I am locking the mutex now.\n");
    mc_MutexLock(mutex_id);
    printf("Agent 1: Mutex locked. Perform protected operations here\n");
    printf("Agent 1: Waiting for 5 seconds...\n");
    sleep(5);
    printf("Agent 1: Unlocking mutex now...\n");
    mc_MutexUnlock(mutex_id);

    return 0;
}
        ]]>
        </AGENT_CODE>
      </TASK>
      </AGENT_DATA>
    </MOBILE_AGENT>
  </MESSAGE>
</GAF_MESSAGE>
```

The previous example code may be found at mobilec/demos/agent_mutex_example/sleep.xml

See Also

mc_MutexUnlock(), mc_SyncInit(), mc_SyncDelete().

mc_MutexUnlock()

Synopsis
int mc_MutexUnlock(int id);

Purpose
This function unlocks a locked Mobile-C synchronization variable.

Return Value
This function returns 0 on success, or non-zero if the id could not be found.

Parameters

id	The id of the synchronization variable to lock.

Description
This function unlocks a Mobile-C synchronization variable that was previously locked as a mutex. If the mutex is not locked while calling this function, undefined behaviour results. Note that although a Mobile-C may act as a mutex, condition variable, or semaphore, once it has been locked and/or unlocked as a mutex, it should only be used as a mutex for the remainder of it's life cycle or unexpected behaviour may result.

Example

See Also
mc_MutexLock(), mc_SyncInit(), mc_SyncDelete().

mc_PrintAgentCode()

Synopsis
int mc_PrintAgentCode(**MCAgent_t** $agent$);

Purpose
Print a mobile agent code for inspection.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
$agent$ The mobile agent from which to print the code.

Description
This function prints the mobile agent code to the standard output.

Example

See Also

mc_RetrieveAgent()

Synopsis
MCAgent_t mc_RetrieveAgent($void$);

Purpose
Retrieve the first neutral mobile agent from a mobile agent list.

Return Value
The function returns an **MCAgent_t** object on success or NULL on failure.

Parameters

$void$	This function does not take any parameters.

Description
This function retrieves the first agent with status AGENT_NEUTRAL from a mobile agent list. If there are no mobile agents with this attribute, the return value is NULL.

Example

See Also

mc_RetrieveAgentCode()

Synopsis
char *mc_RetrieveAgentCode(**MCAgent_t** $agent$);

Purpose
Retrieve a mobile agent code in the form of a character string.

Return Value
The function returns an allocated character array on success and NULL on failure.

Parameters

$agent$ The mobile agent from which to retrieve the code.

Description
This function retrieves a mobile agent code. The return pointer is allocated by 'malloc()' and must be freed by the user.

Example

See Also

mc_SemaphorePost()

Synopsis
int mc_SemaphorePost(int id);

Purpose
This function unlocks one resource from a Mobile-C semaphore, increasing its count by one.

Return Value
This function returns 0 on success, or non-zero if the id could not be found or on a semaphore error.

Parameters

id	The id of the synchronization variable to lock.

Description
mc_SemaphorePost unlocks a resource from a previously allocated and initialized Mobile-C synchronization variable being used as a semaphore. This function may be called multiple times to increase the count of the semaphore up to INT_MAX. Note that although a Mobile-C synchronization variable may be used as a mutex, condition variable, or semaphore, once it is used as a semaphore, it should only be used as a semaphore for the remainder of its life cycle.

Example

See Also
mc_SemaphoreWait(), mc_SyncInit(), mc_SyncDelete().

mc_SemaphoreWait()

Synopsis
#include <**libmc.h**>
int mc_SemaphoreWait(**int** id);

Purpose
This function allocates one resource from a MobileC synchronization semaphore variable.
Return Value
This function returns 0 on success, or non-zero if the id could not be found.

Parameters

id	The id of the synchronization variable to lock.

Description
This function allocates one resource from a previously allocated and initialized MobileC synchronization semaphore. If the semaphore resource count is non-zero, the resource is immediately allocated. If the semaphore resource count is zero, the function blocks until a resource is freed before allocating a resource and continuing.

Note that although a MobileC synchronization variable may be used as a mutex, condition variable, or semaphore, once it is used as a semaphore, it should only be used as a semaphore for the remainder of its life cycle.
Example

See Also
mc_SemaphorePost(), mc_SyncInit(), mc_SyncDelete().

mc_SendAgentMigrationMessage()

Synopsis
int mc_SendAgentMigrationMessage(**char ***$message$, **char ***$hostname$, **int** $port$);

Purpose
Send an ACL mobile agent message to a remote agency.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

$message$	The ACL mobile agent message to be sent.
$hostname$	The hostname of the remote agency. It can be in number-dot format or hostname format, i.e., 10.20.30.40 or iel.ucdavis.edu.
$port$	The port number on which the remote agency is listening.

Description
This function is used to send an XML based ACL mobile agent message, which is a string, to a remote agency.

Example

See Also

mc_SendAgentMigrationMessageFile()

Synopsis
int mc_SendAgentMigrationMessageFile(const char * $filename$, **const char *** $hostname$, **int** $port$);

Purpose
Send an ACL mobile agent message saved as a file to a remote agency.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

$filename$	The ACL mobile agent message file to be sent.
$hostname$	The hostname of the remote agency. It can be in number-dot format or hostname format, i.e., 10.20.30.40 or iel.ucdavis.edu.
$port$	The port number on which the remote agency is listening.

Description
This function is used to send an XML based ACL mobile agent message, which is saved as a file, to a remote agency.

Example

See Also

mc_SetAgentStatus()

Synopsis
int mc_SetAgentStatus(MCAgent_t $agent$, **int** $status$);

Purpose
Set the status of a mobile agent in an agency.

Return Value
This function returns 0 on success and non-zero otherwise.
Parameters
$agent$ The mobile agent whose status is to be assigned.
$status$ An integer representing the status to be assinged to a mobile agent.

Description
This function sets the status of a mobile agent by one of the enumerated values listed below.

0 , WAIT_CH :	Mobile agent will wait to be executed.
3 , AGENT_NEUTRAL :	Mobile agent will wait for an unspecified reason.
4 , AGENT_SUSPENDED :	Mobile agent will be suspended.

Example

See Also

mc_SetDefaultAgentStatus()

Synopsis
int mc_SetDefaultAgentStatus(int $status$);

Purpose
Set the default status of any incoming mobile agents.

Return Value
This function returns 0 on success and non-zero otherwise.
Parameters
$status$ An integer representing the status to be assinged to any incoming mobile agents as their default status.

Description
This function sets the default status of any incoming mobile agents by one of the enumerated values listed below.

0 , WAIT_CH :	Mobile agent will wait to be executed.
3 , AGENT_NEUTRAL :	Mobile agent will wait for an unspecified reason.
4 , AGENT_SUSPENDED :	Mobile agent will be suspended.

Example

See Also

mc_SyncDelete()

Synopsis
int mc_SyncDelete(int id);

Purpose
Delete a previously initialized synchronization variable.

Return Value
This function returns 0 on success and nonzero otherwise.

Parameters

id	The id of the condition variable to delete.

Description
This function is used to delete and deallocate a previously initialized Mobile-C synchronization variable.

Example

See Also
mc_SyncInit().

mc_SyncInit()

Synopsis
int mc_SyncInit(int id);

Purpose
Initialize a new synchronization variable for agents to wait on.

Return Value
This function returns the allocated id of the synchronization variable.

Parameters

id	A requested synchronization variable id. A random id will be assigned if the value passed is 0 or if there is a conflicting id.

Description
This function initializes and registers a new MobileC synchronization variable. Mobile-C Synchronization variables may be used as a mutex, a condition variable (with an associated mutex), or a semaphore. The purpose of the Mobile-C synchronization variables is to synchronize the execution of agents with each other, as well as the excution of agents with their respective agencies.

Example

See Also
mc_CondSignal(), mc_CondWait(), mc_MutexLock(), mc_MutexUnlock(), mc_SemaphorePost(), mc_SemaphoreWait(), mc_SyncDelete().

mc_TerminateAgent()

Synopsis
int mc_TerminateAgent(MCAgent_t $agent$);

Purpose
Terminate the execution of a mobile agent in an agency.

Return Value
The function returns 0 on success and an error code on failure.

Parameters
$agent$ A valid mobile agent.

Description
This function halts a running mobile agent. The Ch interpreter is left intact. The mobile agent may still reside in the agency in AGENT_NEUTRAL mode if the mobile agent is tagged as 'persistent', or is terminated and flushed otherwise.

Example

See Also

Index