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

User's Guide

Version 1.9.2

Harry H. Cheng

Mobile-C User's Guide version 1.9.2 prepared by:

David Ko
Yu-Cheng Chou

October 22, 2007

Major Contributors (in alphabetical order)

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

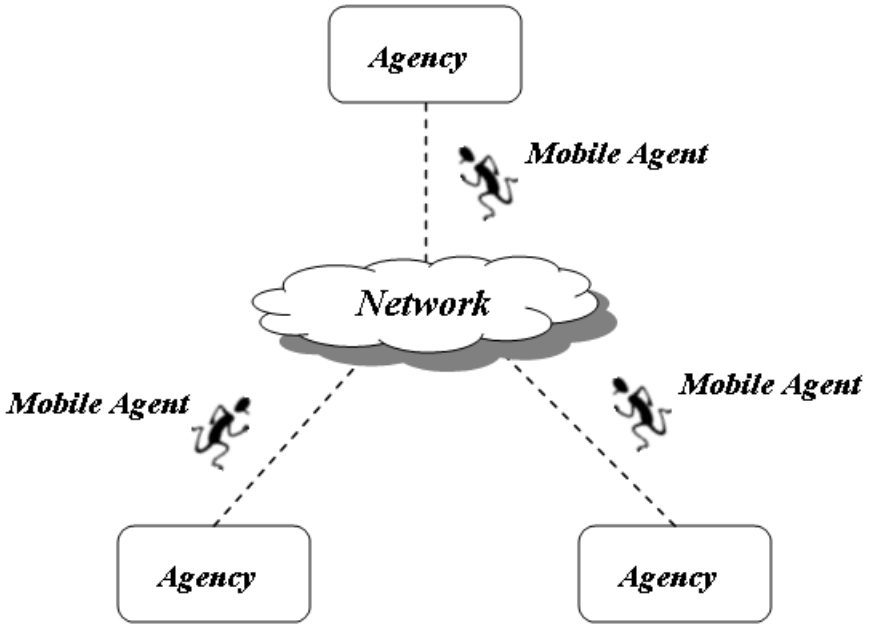
People who helped to make Mobile-C the real thing (if you noticed that some names are missing, please mail to mobilec@iel.ucdavis.edu)

Name	Company (during 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, Yu-Cheng cycchou@ucdavis.edu	Univ. of California, Davis	Design and implementation of the Mobile-C library
Honda, Jason jhonda@sandia.gov	Sandia National Laboratories	
Ko, David, dko@ucdavis.edu	Univ. of California, Davis	Design and implementation of the Mobile-C library
Linz, David ddlinz@gmail.com	Univ. of California, Davis	Design and implementation of Mobile-C
Nestinger, Stephen S., thestinger@ucdavis.edu	Univ. of California, Davis	Webmaster of http://www.mobilec.org

Copyright

Abstract

Mobile-C is an IEEE FIPA (Foundation for Intelligent Physical Agents) standard compliant multi-agent platform for supporting C/C++ mobile agents in networked intelligent mechatronic and embedded systems. Although it is a general-purpose multi-agent platform, Mobile-C is specifically designed for real-time and resource constrained applications with interface to hardware. Mobile agents are software components that are able to move between different execution environments. Mobile agents in a multi-agent system communicate and work collaboratively with other agents to achieve a global goal. It allows a mechatronic or embedded system to adapt to a dynamically changing environment.



Contents

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 Requirements

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

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 Building the Mobile-C Library

The following steps are suggested to build the Mobile-C library.

1. Open your development environment. Currently, only Visual Studio .NET 2003 and 2005 are tested and/or supported.
2. Open the Mobile-C project solution. It is located in either the mobilec/src/win32/vcnet2003/ directory or the mobilec/src/win32/vcnet2005/ directory and is named "mc_lib_win32.sln".
3. Ensure that the 'Debug' configuration is selected.
4. Click on "Build–Build Solution" from the menu. This should automatically build Mobile-C and all of it's modules in the correct order. The library produced is named libmc.lib located in the mobilec/src/ directory.

2.4 Installing the Mobile-C Security Module

For instructions on how to install the Mobile-C Security Module, please refer to Chapter 8 on page 41.

Chapter 3

Getting Started

3.1 Compilation on Unix

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

3.2 Compilation on Windows

The following steps are suggested for building the Mobile-C demos.

1. Open your developement environment. Currently, only Visual .NET 2003 and 2005 are supported.
2. Open the Demo Project solution. It is named "mobilec_demos.sln" and is located in either the mobilec/demos/win32/vcnet2003 or the mobilec/demos/win32/vcnet2005 directory.
3. If you are using VC Net 2005, the 'Release' configuration option may provide better results. The issue with the 'Debug' configuration is to be resolved.
4. To build all of the demos, click on "Build–Build Solution" from the menu.

3.3 Overview of Sample Application Programs

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

int main(int argc, char *argv[])
{
  MCAgency_t agency;
  int local_port = 5051;

  agency = MC_Initialize(local_port, NULL);

  printf("Mobile-C Started\n");
  MC_Wait(agency);
  return 0;
}
```

Program 1: A sample Mobile-C server. (demos/hello_world/server.c)

Program 1 on the preceding page starts an agency that is capable of receiving mobile agents and executing mobile agent code.
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.
The variable $agency$, of type **MCAgency_t**, is a handle that contains information of an agency. The second line initializes a local variable that will hold the port number we wish the agency to bind to.
MC_Initialize() takes an integer and the address of an **MCAgencyOptions_t** variable 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 instead of an MCAgencyOptions_t variable to start an agency with default settings. A local agency will be initialized to listen on port **5051** specified by the variable $local\_port$.
The agency waits indefinitely for a mobile agent by the function **MC_Wait()** .

Program 2 starts an agency that sends a mobile agent to a remote agency. Examining Programs 1 and 2, we see that there are only two new API function calls:

```
MC_SendAgentMigrationMessageFile(
            agency,
            "test1.xml",
            "localhost",
            5051);
```

and

```
MC_End(agency);
```

In Mobile-C, a mobile agent message is an Agent Communication Language (ACL) message in Extensible Markup Language (XML) format. **MC_SendAgentMigrationMessageFile()** 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_SendAgentMigrationMessageFile() sends the mobile agent message saved as **test1.xml** in current directory to the remote agency running on host **localhost** and listening on port **5051**.

After the agent is sent, a call to function **MC_End()** is made. This function tells all of the Mobile-C internal modules to gracefully finish whatever they are doing and exit. This function call is important after calling **MC_SendAgentMigrationMessageFile()** to ensure that the agent is fully processed and sent before terminating the agency. Failure to call **MC_End()** here may result in the agent not being properly sent to the receiving agency.

Also note that any valid hostname may be used in place of “localhost”. The communicating agencies need not be on the same physical machine; in fact, in most cases they will be on seperate machines. Any IPv4 string, i.e. “169.237.104.199”, or qualified hostname, i.e. “machine.ucdavis.edu”, may be used. For instance, the code

```
MC_SendAgentMigrationMessageFile(
            agency,
            "test1.xml",
            "169.237.104.199",
            5055);
```

will send an agent to the server at address “169.237.104.199” listening on port 5055. Or,

```
MC_SendAgentMigrationMessageFile(
```

```
agency,
"test1.xml",
"machine.ucdavis.edu",
5031);
```

will send the agent to an agency at “machine.ucdavis.edu” listening on port 5031.

3.4 Execution of Sample Applications

In general, each of the demos is designed to have very similar execution procedures. For each demo, there are one or more “servers”, which are simply vanilla Mobile-C agencies. To run the demo, start all of the servers (there is only one server for most of the demos), and start the “client” program. Generally, the client program also starts a Mobile-C agency, but it typically sends an agent to a destination as part of its startup process as well.

For example, to run the Mobile-C “Hello World” example, run the following commands from a text terminal on the server machine to start an agency listening on port **5051**.

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

Next, run the following commands from a text terminal on the client machine to start an agency listening on port **5050** and send the mobile agent message **test1.xml**, shown as Program 3 on the next page, to the remote agency listening on port **5051**.

```
cd <MCPACKAGE>/demos/hello_world
./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. Note that in this example, both the server and client are running on the same machine, but this is not a requirement. The field “localhost” may be replaced with any qualified domain name or IP address.

3.5 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.

3.5.1 Architecture of the Mobile-C Library

Figure 3.1 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.

```
#include <stdio.h>
#include <stdlib.h>
#include <libmc.h>

int main(int argc, char *argv[])
{
    MCAgency_t agency;
    agency = MC_Initialize(5050, NULL);

    printf("Mobile-C Started\n");
    MC_SendAgentMigrationMessageFile(agency,
                "test1.xml",
                "localhost",
                5051);
    MC_End(agency);
    exit(0);
}
```

Program 2: A sample Mobile-C client program. The sole purpose of this program is to send a Mobile-C agent to another agency. (demos/hello_world/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>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()
{
    printf("Hello World!\n");
    printf("This is mobagent1 from the agency at port 5050.\n");
    printf("I am performing the task on the agency at port 5051 now.\n");
    return 0;
}
      ]]>
     </AGENT_CODE>
    </TASK>
   </AGENT_DATA>
  </MOBILE_AGENT>
 </MESSAGE>
</GAF_MESSAGE>
```

Program 3: A simple Mobile-C agent. (demos/hello_world/test1.xml)

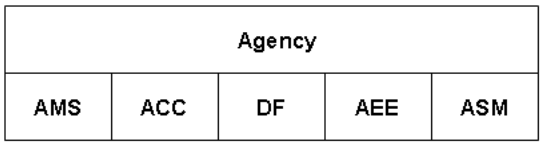


Figure 3.1: Architecture of the Mobile-C library.

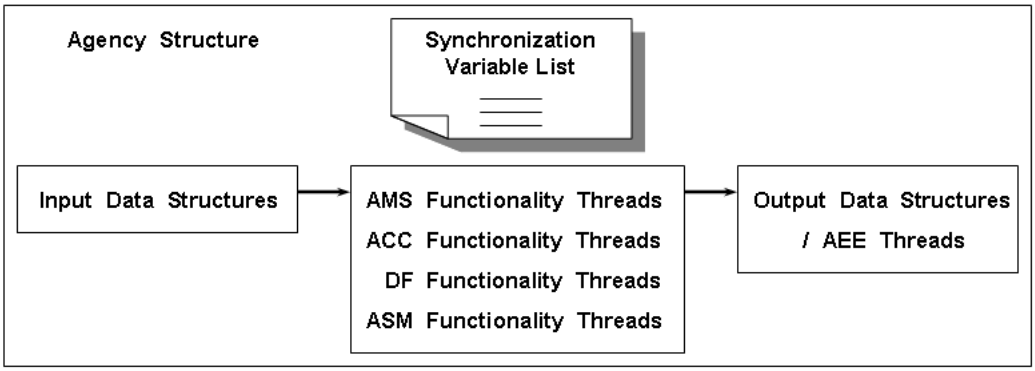


Figure 3.2: Implementation overview of the Mobile-C library.

As a IEEE 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 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.

3.5.2 Implementation of the Mobile-C Library

Figure 3.2 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 500 KB and 390 KB, respectively.

The header file *libmc.h* contains definitions of all the structures and functions of the Mobile-C library. Table A.3 on page 46 lists the currently implemented functions for the binary space.

Chapter 4

Mobile-C Agent Migration Message Format

4.1 General Message Format

The message format for an agent migration message is designed such that multiple tasks and multiple code blocks can be migrated from agency to agency. The message is an XML message with encapsulated C code. An example of a rudimentary agent can be seen in Program 4 on page 12. Following is a brief description of each XML tag.

- MESSAGE: This tag indicates to Mobile-C that the following data is a Mobile-C message. The message type is included in the attribute "message".
- MOBILE_AGENT: This tag indicates that the contained data is a Mobile-C agent.
- AGENT_DATA: This tag indicates that the contained data is data pertaining to this particular agent.
- NAME: The name of the agent.
- OWNER: The owner of the agent.
- HOME: The home of the agent. Any agent that has data to "return" will return it to this address by default.
- TASK: This indicates that the following information pertains to the task or tasks the agent is intended to perform.
- DATA: Each seperate DATA tag indicates a seperate task for the agent to perform. The tasks may be seperate hosts and/or code blocks. In the rudimentary example, there is only one task.
- AGENT_CODE: Each AGENT_CODE block represents a block of code that the agent may execute. Agents with multiple code blocks may decide at run-time which block to execute.

4.2 Multiple Tasks with a Single Code Block

An agent may have an indefinite number of tasks. The agent will perform the tasks in ther order that they are stated in the XML file, completing each one before continuing to the next host. Following is an example of an agent which has multiple tasks to perform, executing the same code block at each new host. See Program 5 on page 13 for an example.

```
<?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: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()
{
    printf("Hello World!\n");
    printf("This is mobagent1 from the agency at port 5050.\n");
    printf("I am performing the task on the agency at port 5051 now.\n");
    return 0;
}
      ]]>
     </AGENT_CODE>
    </TASK>
   </AGENT_DATA>
  </MOBILE_AGENT>
 </MESSAGE>
</GAF_MESSAGE>
```

Program 4: A rudimentary agent. (demos/hello_world/test1.xml)

```
<?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="2" num="0">
     <DATA dim="0" name="results_iel2" complete="0" server="localhost:5051">
     </DATA>
     <DATA dim="0" name="results_ch" complete="0" server="localhost:5052">
     </DATA>
     <AGENT_CODE>
      <![CDATA[
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

double results_iel2;
double results_ch;

int main()
{
    FILE * fptr;
    char line[1024];
    double velocity, count = 0, sum = 0;

    printf("\nThis is the mobile agent 3 from the bird1 machine.\n\n");
    printf("My task on the %s is to find the average velocity of ", mc_host_name);
    printf("vehicles passed under the %s detection station.\n\n", mc_host_name);
    if (mc_task_progress == 0) {
        if((fptr = fopen("ChDataFile_iel2", "r")) == NULL)
        {
            printf("Error: could not open file 'ChDataFile_iel2'.\n");
            exit(EXIT_FAILURE);
        }
    } else {
        if((fptr = fopen("ChDataFile_ch", "r")) == NULL)
```

Program 5: An example agent containing two tasks and a single code block. Note that variables "mc_host_name" and "mc_task_progress" are special built-in variables described in Table B.3 on page 106. (demos/multi_task_example/test_single_code_block.xml)

```
        {
            printf("Error: could not open file 'ChDataFile_ch'.\n");
            exit(EXIT_FAILURE);
        }
    }

    fgets(line, sizeof(line), fptr);
    while(!feof(fptr))
    {
        velocity = atof(strrchr(line, ',') + 1);
        sum += velocity;
        count++;
        fgets(line, sizeof(line), fptr);
    }
    if(count != 0)
    {
        if (mc_task_progress == 0) {
            results_iel2 = sum/count;
        } else {
            results_ch = sum/count;
        }

        printf("The average velocity under the detection station is %f.\n\n", sum/count);
    }
    else
    {
        results_iel2 = 0;
        results_ch = 0;
        printf("There is no vehicle passed under the detection station.\n\n");
    }

    fclose(fptr);
    printf("I am leaving to go to the next host.\n");

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

Program 5: An example agent containing two tasks and a single code block. (Continued) Note that variables "mc_host_name" and "mc_task_progress" are special built-in variables described in Table B.3 on page 106.

4.3 Multiple Tasks with Multiple Code Blocks

See Program 6 on page 16 for a more complicated example of agent code including multiple tasks and multiple code blocks. Note that each code block has an associated ID which is refered to in the respective “DATA” tags. Also note that more than one “DATA” tag may refer to the same code block. Thus, an agent may have more “DATA” tags than code blocks.

```
<?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="2" num="0">
     <DATA dim="0" name="results_iel2" complete="0" server="localhost:5051" code_id="1"/>
     <DATA dim="0" name="results_ch" complete="0" server="localhost:5052" code_id="2"/>
     <AGENT_CODE id="1">
      <![CDATA[
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

double results_iel2;
int main()
{
    FILE * fptr;
    char line[1024];
    double velocity, count = 0, sum = 0;

    printf("\nThis is the mobile agent 3 from the bird1 machine.\n\n");
    printf("My task on the %s is to find the average velocity of ", mc_host_name);
    printf("vehicles passed under the %s detection station.\n\n", mc_host_name);
    if((fptr = fopen("ChDataFile_iel2", "r")) == NULL)
    {
      printf("Error: could not open file 'ChDataFile_iel2'.\n");
      exit(EXIT_FAILURE);
    }

    fgets(line, sizeof(line), fptr);
    while(!feof(fptr))
    {
      velocity = atof(strrchr(line, ',') + 1);
      sum += velocity;
      count++;
      fgets(line, sizeof(line), fptr);
    }
    if(count != 0)
    {
      results_iel2 = sum/count;

      printf("The average velocity under the detection station is %f.\n\n", sum/count);
    }
    else
    {
      results_iel2 = 0;
      printf("There is no vehicle passed under the detection station.\n\n");
    }
    fclose(fptr);
    printf("I am leaving to go to the next host.\n");
```

Program 6: An example agent containing two tasks and two code blocks. (demos/multi_task_example/test_multi_code_block.xml)

```
    return 0;
}
      ]]>
     </AGENT_CODE>
     <AGENT_CODE id="2">
      <![CDATA[
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

double results_ch;

int main()
{
    FILE * fptr;
    char line[1024];
    double velocity, count = 0, sum = 0;

    printf("\nThis is the mobile agent 3 from the bird1 machine.\n\n");
    printf("My task on the %s is to find the average velocity of ", mc_host_name);
    printf("vehicles passed under the %s detection station.\n\n", mc_host_name);
    if((fptr = fopen("ChDataFile_ch", "r")) == NULL)
    {
      printf("Error: could not open file 'ChDataFile_ch'.\n");
      exit(EXIT_FAILURE);
    }

    fgets(line, sizeof(line), fptr);
    while(!feof(fptr))
    {
        velocity = atof(strrchr(line, ',') + 1);
        sum += velocity;
        count++;
        fgets(line, sizeof(line), fptr);
    }
    if(count != 0)
    {
      results_ch = sum/count;
      printf("The average velocity under the detection station is %f.\n\n", sum/count);
    }
    else
    {
      results_ch = 0;
      printf("There is no vehicle passed under the detection station.\n\n");
    }

    fclose(fptr);
    printf("I am leaving to go to the next host.\n");

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

Program 6: An example agent containing two tasks and two code blocks. (Continued)

Chapter 5

Interface between Binary and Mobile Agent Spaces

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 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 7 on the next page 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 *localhost* at port *5051* through the function *MC_SendAgentMigrationMessageFile()*. 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 shown in Program 8 on page 20. The name, owner, source machine of the mobile agent are *mobagent1*, *IEL*, and *localhost:5050*, respectively. The mobile agent is persistent since the flag *persistent* is set to 1 in Program 8. 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 9 on page 21, the server program starts a Mobile-C agency listening on port *5051* by the function *MC_Initialize()*, and waits for a mobile agent. The mobile agent named *mobagentl* 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

```
#include <libmc.h>

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

int main(int argc, char *argv[])
{
    /* Init the agency */
    MCAgency_t agency;
    agency = MC_Initialize(
            5050,
            NULL);

    MC_SendAgentMigrationMessageFile(
            agency,
            "test1.xml",
            "localhost",
            5051);
    MC_End(agency);

    return 0;
}
```

Program 7: A program which sends a persistent mobile agent. (demos/persistent_example/client.c)

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 9. 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 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.

The other method of executing the function is through the Mobile-C api function **MC_CallAgentFunc()** . This method is seen in the example program, Program 9.

Figure 5.1 on page 21 shows the output when the binary file *host* compiled from Program 9 was executed. The string generated and the value returned from the function *hello()* were printed to the screen after the Enter key was pressed once the mobile agent had arrived.

```
<?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:5050</HOME>
    <TASK task="1" num="0">
     <DATA
          persistent="1"
          number_of_elements="0"
          name="no-return"
          complete="0"
          server="localhost:5051">
     </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 8: A persistent mobile agent. Agents marked “persistent” are not flushed from the agency after they terminate. (demos/persistent_example/test1.xml)

```
#include <libmc.h>
#include <embedch.h>
#include <stdio.h>

struct arg_struct{
    int a;
    int b;
}arg;

int main(int argc, char *argv[])
{
    MCAgent_t agent;
    ChInterp_t interp;
    int retval;

    /* Init the agency */
    MCAgency_t agency;
    agency = MC_Initialize(
            5051,
            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);
    }
    /* The following execututions of code may be performed two different
       ways: The first way, which is commented out in this example,
       involves retrieving the agent's interpreter with
       MC_GetAgentExecEngine() and using the Embedded Ch api to call
       the function. The second method involves using the Mobile-C
       api to call the function. Both of these methods used here produce
       identical results. */
    arg.a = 50;
    arg.b = 51;
    /*interp = MC_GetAgentExecEngine(agent);
    Ch_CallFuncByName(interp, "hello", &retval, arg); */
    MC_CallAgentFunc(
            agent,
            "hello",
            &retval,
            &arg);
    printf("Value of %d was returned.\n", retval);
    return 0;
}
```

Program 9: A Mobile-C agency. (demos/persistent_example/host.c)

Figure 5.1: Output from the binary server program.

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 on page 24 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].

The prototypes of Mobile-C functions used in the mobile agent space are declared in *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 *agent.c* by two Embedded Ch functions, *Ch_DeclareVar()* and *Ch_DeclareTypedef()* [12]. Table B.5 on page 108 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 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 program used in this example is the same as Program 1 on page 5. The client program is the same as Program 7 on page 19 except that it calls the function *MC_SendAgentMigrationMessageFile()* twice to send out two mobile agents. The first mobile agent sent to the remote agency is *test2.xml* shown in Program 10 on page 25. The execution of the mobile agent code will repeatedly print a string *Hello* to the screen every second. The second mobile agent sent to the remote agency is *test3.xml* shown in Program 11 on page 26. 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 *mobagent1* by the function *mc_FindAgentByName()* and terminates the execution of *mobagent1* by the function *mc_TerminateAgent()* .

6.2 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.

The server program used in this example is the same as Program 1 on page 5. The client program is the same as Program 7 on page 19 except that it calls the function *MC_SendAgentMigrationMessageFile()* three times to send out three mobile agents. The first mobile agent sent to the remote agency is shown in Program 12 on page 27. 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 12 on page 27. The system variable *mc_current_agent* is declared in *agent.c* using the function *Ch_DeclareVar()* to hold the current mobile agent. An array of pointer to char 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 is similar to the first and also registers two services, *multiplication* and *modulus*, which provides multiplication and modulo operation of two integers. This mobile agent can be seen in Program 13 on page 28.

The third mobile agent sent to the remote agency is shown in Program 14 on page 29. 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 char 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 char 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 *service_provider_1*. The function *addition()* defined in *service_provider_1* 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 addition 44 + 45 is 89.* will be printed to the screen at the end.

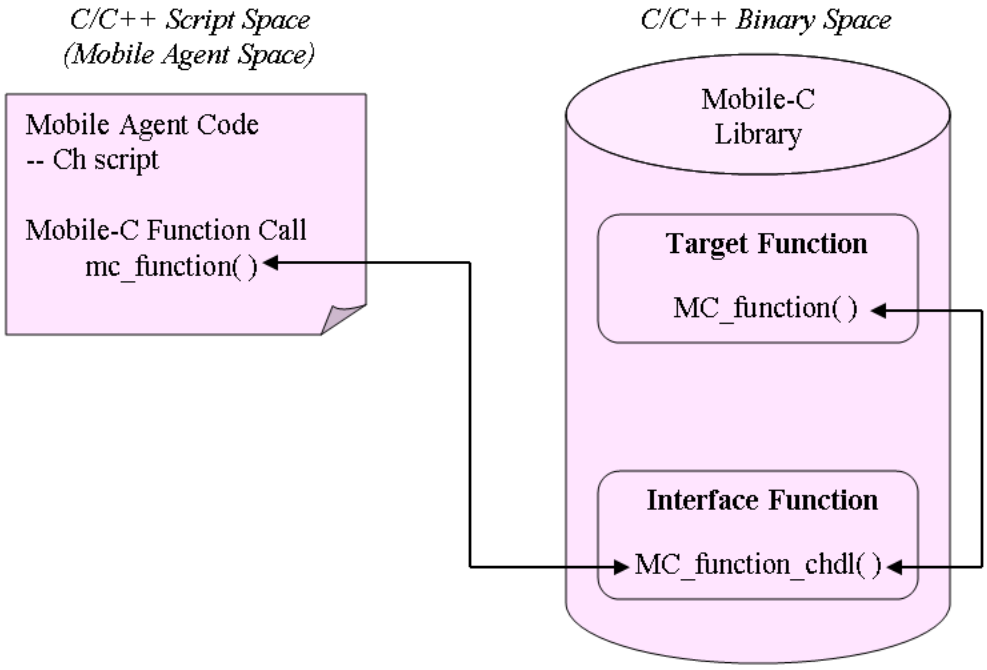


Figure 6.1: Interface of mobile agent code with the Mobile-C library.

```
<?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: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>
#include <unistd.h>
int main()
{
    while(1) {
        printf("Hello\n");
        /* Sleep for 1 second */
        usleep(1000000);
    }
    return 0;
}
      ]]>
     </AGENT_CODE>
    </TASK>
   </AGENT_DATA>
  </MOBILE_AGENT>
 </MESSAGE>
</GAF_MESSAGE>
```

Program 10: A mobile agent which enters an infinite loop and does not terminate. (demos/persistent_example/test2.xml)

```
<?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:5051">
     </DATA>
     <AGENT_CODE>
      <![CDATA[
#include <stdio.h>
int main()
{
    MCAgent_t tmp;
    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 11: This agent terminates the execution of the agent in Program 10. (demos/persistent_example/test3.xml)

```
<?xml version="1.0"?>
<!DOCTYPE myMessage SYSTEM "gafmessage.dtd">
<GAF_MESSAGE>
 <MESSAGE message="MOBILE_AGENT">
  <MOBILE_AGENT>
   <AGENT_DATA>
    <NAME>service_provider_1</NAME>
    <OWNER>IEL</OWNER>
    <HOME>localhost:5050</HOME>
    <TASK task="1" num="0">
     <DATA persistent="1" number_of_elements="0"
           name="no-return" complete="0" server="localhost:5051">
     </DATA>
     <AGENT_CODE>
      <![CDATA[
#include <stdio.h>

struct arg_struct {
    int a;
    int b;
};

int main() {
    char **services;
    int i;
    services = malloc(sizeof(char*)*2);
    for(i = 0; i < 2; i++) {
        services[i] = malloc(40);
    }
    strcpy(services[0], "addition");
    strcpy(services[1], "subtraction");
    printf("Service provider 1 has arrived.\n");
    printf("I provide addition and subtraction service.\n");
    mc_RegisterService( mc_current_agent, services, 2);
    return 0;
}

int addition(struct arg_struct* arg) {
    printf("Adding %d and %d...\n", arg->a, arg->b);
    return arg->a + arg->b;
}

int subtraction(struct arg_struct* arg) {
    printf("Subtracting %d - %d...\n", arg->a, arg->b);
    return arg->a - arg->b;
}
      ]]>
     </AGENT_CODE>
    </TASK>
   </AGENT_DATA>
  </MOBILE_AGENT>
 </MESSAGE>
</GAF_MESSAGE>
```

Program 12: Sample agent containing ’addition’ and ’subtraction’ services. Note that the variable “mc_current_agent” is a special built-in variable described in Table B.3 on page 106. (demos/mc_df_service_test/service_provider_1.xml)

```
<?xml version="1.0"?>
<!DOCTYPE myMessage SYSTEM "gafmessage.dtd">
<GAF_MESSAGE>
 <MESSAGE message="MOBILE_AGENT">
  <MOBILE_AGENT>
   <AGENT_DATA>
    <NAME>service_provider_2</NAME>
    <OWNER>IEL</OWNER>
    <HOME>localhost:5050</HOME>
    <TASK task="1" num="0">
     <DATA persistent="1" number_of_elements="0"
           name="no-return" complete="0" server="localhost:5051">
     </DATA>
     <AGENT_CODE>
      <![CDATA[
#include <stdio.h>

struct arg_struct {
    int a;
    int b;
};

int main() {
    char **services;
    int i;
    services = malloc(sizeof(char*)*2);
    for(i = 0; i < 2; i++) {
        services[i] = malloc(40);
    }
    strcpy(services[0], "multiplication");
    strcpy(services[1], "modulus");
    printf("Service provider 2 has arrived.\n");
    printf("I provide multiplication and modulus service.\n");
    mc_RegisterService( mc_current_agent, services, 2);
    return 0;
}

int multiplication(struct arg_struct* arg) {
    printf("Multiplying %d and %d...\n", arg->a, arg->b);
    return arg->a * arg->b;
}

int modulus(struct arg_struct* arg) {
    printf("Modulo %d % %d...\n", arg->a, arg->b);
    return arg->a % arg->b;
}
      ]]>
     </AGENT_CODE>
    </TASK>
   </AGENT_DATA>
  </MOBILE_AGENT>
 </MESSAGE>
</GAF_MESSAGE>
```

Program 13: Sample agent containing ’multiplication’ and ’modulus’ services. Note that the variable “mc_current_agent” is a special built-in variable described in Table B.3 on page 106. (demos/mc_df_service_test/service_provider_2.xml)

```
<?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:5050</HOME>
    <TASK task="1" num="0">
     <DATA number_of_elements="0" name="no-return"
           complete="0" server="localhost:5051">
     </DATA>
     <AGENT_CODE>
      <![CDATA[
#include <stdio.h>

struct arg_struct {
    int a;
    int b;
};

int main() {
    MCAgent_t agent;
    int retval;
    /* Search Return Variables */
    char** agentNames;
    char** serviceNames;
    int *agentIDs;
    int numResults;

    /* Argument Struct */
    struct arg_struct arg;

    /* Search for addition service */
    printf("\n\n\nSearching for addition service.\n");
    mc_SearchForService(
            "addition",
            &agentNames,
            &serviceNames,
            &agentIDs,
            &numResults);
    printf("Done searching.\n");
    if (numResults < 1) {
        printf("No agents with service 'addition' found.\n");
        exit(0);
    }

    /* Just get the first hit */
    printf("Using agent %s for addition.\n", agentNames[0]);
    agent = mc_FindAgentByID(agentIDs[0]);

    arg.a = 44;
    arg.b = 45;
    mc_CallAgentFunc(agent, "addition", &retval, &arg);
    printf("Result of addition %d + %d is %d.\n", arg.a, arg.b, retval);
    return 0;
}
      ]]>
     </AGENT_CODE>
    </TASK>
   </AGENT_DATA>
  </MOBILE_AGENT>
 </MESSAGE>
</GAF_MESSAGE>
```

Program 14: Sample agent that searches for and invokes agent services. (demos/mc_df_service_test/test2.xml)

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.

A Mobile-C synchronization variable is an abstract variable, initialized by the function *MC_SyncInit()* . Once it has been initialized, it may be used as a mutex, condition variable, or semaphore. No further function calls are necessary to change a generic synchronization variable to one of the types. However, once a synchronization variable is used as a mutex, condition variable, or semaphore, it should not be used again as a different type. For instance, if calls to

```
MC_SyncInit(500);
MC_MutexLock(500);
```

are made, initializing a synchronization variable with ID “500”, and locking it as a mutex, it should not be then used with any of the condition variable or semaphore functions.

The application of the Mobile-C synchronization mechanism is illustrated by the example below.

7.1 Synchronization in Mobile Agent Space

The Mobile-C library allows synrchonization among agents via mutexes, condition variables, and semaphores. Each type of synchronization variable is used for different features. Perhaps the most common and basic of these variables is the mutex.

The client program shown in Program 15 on the next page starts a Mobile-C agency listening on port *5050* and subsequently sends two mobile agents to the remote agency running on host *localhost* at port *5051*. The mobile agents are shown in Program 16 on page 32 and Program 17 on page 33. These mobile agents will use a mutex to guard an operation that may not be performed by two agents simultaneously.

```
#include <stdio.h>
#include <libmc.h>
#ifdef _WIN32
#include <windows.h>
#endif
#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);
#ifndef _WIN32
    sleep(WAIT_TIME);
#else
    Sleep(WAIT_TIME * 1000);
#endif
    printf("Sending wake-up agent...\n");
    MC_SendAgentMigrationMessageFile(
            agency,
            "wake.xml",
            "localhost",
            5051);
    MC_End(agency);
    return 0;
}
```

Program 15: A program used to send a mobile agent. (demos/agent_mutex_example/mc_client.c)

This example demonstrates the ability of a Mobile-C mutex to protect a resource that may be shared between two agents. Any real or imaginary resource that should not be accessed simultaneously by more than one entity at a time should be guarded by a mutex. The resource may be a shared variable, or something more abstract such as control of a robot arm. If there is only one robot arm, then only one entity, an agent in this case, should be able to control it at a time.

In our particular example, the tasks our agents are going to perform are imaginary. Each task is represented instead by the "sleep()" function and the printing of a message, which causes execution of that particular agent to pause for a time, as if it were performing a task. For our example, we will intentionally cause our agents to collide execution times to demonstrate that our mutex works. Examining our client program, Program 15, we see that we set a two second interval between sending the agents. However, the task that each agent tries to perform will be five seconds long. This means that the second agent will arrive while the first agent is in the middle of performing its simulated task. The execution output will demonstrate that the second agent will not begin its task until the first agent is finished.

Semaphores are also used to guard resources in which a limited number of entities may access at a time. Since the behaviour and usage of semaphores are similar to that of a mutex, an example is not provided here. Please see the demo in directory *demos/agent_semaphore_example/* for an example.

Condition variables are also useful in multi-threaded applications in order for threads to sleep and wait

```
<?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 16: An agent which uses a mutex while accessing a shared resource. (demos/agent_mutex_example/sleep.xml)

for a signal. Program 18 on page 34 illustrates an agent that will sleep on a condition variable immediately after arriving at an agency. Program 19 on page 35 shows an agent that will send a signal to the condition variable the first agent in Program 18 is waiting on, thereby causing the first agent to wake up and continue execution.

```
<?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");
    sleep(5);
    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 17: An agent which uses a mutex while accessing a shared resource. (demos/agent_mutex_example/wake.xml)

7.2 Synchronization Between Binary and Agent Spaces

The synchronization variables initialized using MC_SyncInit() are accessible in both agent space and binary space, enabling agents to synchronize with binary threads. Again, all three Mobile-C synchronization variable types: mutexes, condition variables, and semaphores, may be used in both binary and agent space.

Referring the example server code in Program 20 on page 36, we show a piece of code where a binary program containing a Mobile-C agency must perform a subroutine involving a shared resource, protecting it with a mutex. The shared resource will be accessible from both the main() binary thread as well as any agents which are residing in the agency. As such, the server code initializes and uses a mutex to protect the shared resource. In our example agent shown in Program 21 on page 37, we see that this agent needs to access the

```
<?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>

#define SYNC_ID 55
int main()
{
    int cond_id;
    printf("Sleep agent has arrived.\n");
    cond_id = mc_SyncInit(SYNC_ID);
    if (cond_id != SYNC_ID) {
        printf("Possible error. Aborting...\n");
        exit(1);
    }
    printf("This is the sleep agent.\n");
    printf("I am going to sleep now...\n");
    mc_CondWait(cond_id);
    printf("This is the sleep agent: I am awake now. Continuing...\n");
    mc_SyncDelete(cond_id);

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

Program 18: A sample agent which will immediately sleep on a condition variable after arriving at an agency. (demos/agent_cond_example/sleep.xml)

same shared resource, and so it must first lock the mutex before doing so. This example demonstrates that the mutex will prevent both the agent and binary thread from accessing the resource simultaneously

Referring now to Program 22 on page 38 and Program 23 on page 39, we demonstrate the use of Mobile-C condition variables to synchronize an agent with a binary thread. The binary space thread program shown in Program 22 simply waits on a condition variable. The agent shown in Program 23 signals the binary space thread with a call to *mc_CondSignal()*, causing the binary space thread to run once.

```
<?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>

#define SYNC_ID 55
int main()
{
    int cond_id;
    cond_id = SYNC_ID;
    printf("This is the wake agent.\n");
    mc_CondSignal(cond_id);

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

Program 19: A sample agent which will signal a condition variable after arriving at an agency. (demos/agent_cond_example/wake.xml)

7.3 Mobile-C Execution with Multiple Agencies

Using the Mobile-C library, multiple agencies may be initialized within the same program. This is useful in cases where the agencies may have different AEE configuration properties, privileges, etc. Ch is the chosen AEE of Mobile-C. Functions such as MC_CopyAgent() and MC_AddAgent() become useful in such cases.

In the example shown in Program 24 on page 40, we demonstrate a program with two agencies, listening on ports 5051 and 5052, respectively. In our simple example, the server simple duplicates every agent arriving to the agency on port 5051 and adds a copy to the agency on port 5052.

Note that the MC_CopyAgent() function is necessary here because Mobile-C functions which retrieve agents from agencies only retrieve references to the agents, not copies of the agents. The MC_CopyAgent() function performs a deep copy on the agent structure so that it may be used in another agency. Also note that setting the copied agent's status to "MC_WAIT_CH" ensures that it will execute again upon entering the second agency.

```
#include <libmc.h>
#include <stdio.h>
#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif
#define MUTEX_ID 55
int main(int argc, char *argv[])
{
    MCAgency_t agency;
    MCAgencyOptions_t options;
    int i;
    MC_InitializeAgencyOptions(&options);
    /* We want _all_ the threads on: EXCEPT, the command prompt thread */
    for (i = 0; i < MC_THREAD_ALL; i++) {
        MC_SetThreadOn(&options, i);
    }
    MC_SetThreadOff(&options, MC_THREAD_CP);

    agency = MC_Initialize(
            5051,
            &options);

    MC_SyncInit(agency, MUTEX_ID);
    /* Now, lets perform a simulated task which accesses a shared resource
     * 20 times. */
    for(i = 0; i < 20; i++) {
        printf("C Space: Attempting to lock mutex...\n");
        MC_MutexLock(agency, MUTEX_ID);
        printf("C Space: Mutex Locked. Performing task.\n");
#ifndef _WIN32
        sleep(1);
#else
        Sleep(1000);
#endif
        printf("C Space: Unlocking Mutex...\n");
        MC_MutexUnlock(agency, MUTEX_ID);
        printf("C Space: Mutex Unlocked.\n");
    }

    MC_SyncDelete(agency, MUTEX_ID);
    MC_End(agency);
    return 0;
}
```

Program 20: A sample program with an embedded Mobile-C agency demonstrating the use of a Mobile-C mutex to protect a shared resource. (demos/cspace_mutex_example/mc_server.c)

```
<?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;
    int i;
    printf("This is agent 1.\n");
    for(i = 0; i < 10; i++) {
        printf("Agent: Attempting to lock mutex...\n");
        mc_MutexLock(55);
        printf("Agent: Mutex Locked. Performing protected operations...\n");
        sleep(1);
        printf("Agent: Attempting to unlock mutex...\n");
        mc_MutexUnlock(55);
        printf("Agent: Mutex Unlocked.\n");
    }

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

Program 21: A sample Mobile-C agent which must perform an action on a shared resource guarded by a Mobile-C mutex. (demos/cspace_mutex_example/agent.xml)

```
#include <libmc.h>
#include <stdio.h>
#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif
#define COND_ID 55

int main(int argc, char *argv[])
{
    MCAgency_t agency;
    MCAgencyOptions_t options;
    int i;
    MC_InitializeAgencyOptions(&options);
    /* We want _all_ the threads on: EXCEPT, the command prompt thread */
    for (i = 0; i < MC_THREAD_ALL; i++) {
        MC_SetThreadOn(&options, i);
    }
    MC_SetThreadOff(&options, MC_THREAD_CP);

    agency = MC_Initialize(
            5051,
            &options);

    MC_SyncInit(agency, COND_ID);
    /* Let us wait on a condition variable. Every time an agent signals that
     * variable, we will perform some task. */
    while(1) {
        MC_CondWait(agency, COND_ID);
        printf("C space: I am awake! Performing some task.\n");
        MC_CondReset(agency, COND_ID);
    }

    MC_Wait(agency);
    return 0;
}
```

Program 22: An example server containing a thread which will run once each time it is signalled by another thread or by an agent. (demos/cspace_cond_example/mc_server.c)

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

<!DOCTYPE myMessage SYSTEM "gafmessage.dtd">

<GAF_MESSAGE>
 <MESSAGE message="MOBILE_AGENT">
  <MOBILE_AGENT>
   <AGENT_DATA>
    <NAME>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>
#define COND_ID 55
int main()
{
    int i;
    printf("This is agent 1.\n");
    for(i = 0; i < 5; i++) {
        printf("Agent: Perform some task here.\n");
        sleep(2);
        printf("Agent: signal C space for followup action.\n");
        mc_CondSignal(COND_ID);
        sleep(1);
    }

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

Program 23: A sample agent which signals a condition variable. (demos/cspace_cond_example/agent.xml)

```
#include <libmc.h>
#include <stdio.h>
#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif
int main(int argc, char *argv[])
{
    MCAgency_t agency1;
    MCAgency_t agency2;
    MCAgencyOptions_t options;
    int i;

    MCAgent_t agent;
    MCAgent_t agent_copy;

    MC_InitializeAgencyOptions(&options);
    /* We want _all_ the threads on: EXCEPT, the command prompt thread */
    for (i = 0; i < MC_THREAD_ALL; i++) {
        MC_SetThreadOn(&options, i);
    }
    MC_SetThreadOff(&options, MC_THREAD_CP);

    agency1 = MC_Initialize(
        5051,
        &options);
    agency2 = MC_Initialize(
        5052,
        &options);

    while(1) {
      agent = MC_WaitRetrieveAgent(agency1);
      MC_CopyAgent(&agent_copy, agent);
      MC_SetAgentStatus(agent_copy, MC_WAIT_CH);
      MC_AddAgent(agency2, agent_copy);
      MC_ResetSignal(agency1);
    }

    return 0;
}
```

Program 24: An example program containing two Mobile-C agencies. The program copies agents arriving at the agency on port 5051 to the agency at port 5052.(demos/multiple_agency_example/mc_server.c)

Chapter 8

Mobile-C Security Module

The latest Mobile-C package includes an experimental security module. The security module is intended to provide a method of sending encrypted agents to agencies. The encryption helps guard against man-in-the-middle attacks and provides a small measure of agent authentication as well.

8.1 Security Module Architecture and Overview

The Mobile-C security module uses the Diffie-Hellman key exchange cryptographic protocol to encrypt agents. When a security-enabled agency attempts to contact another agency for the first time, the agencies trade public keys. Every other communication between the agencies will be encrypted using the public keys, and thus may only be decrypted by the target agency's private key. Since the private and public keys may be randomly generated, this makes it difficult/impossible for a third party to decrypt an intercepted encrypted agent, since they do not own the private keys.

8.2 Enabling the Security Module

default. Several configuration options need to be changed in order for the module to be built and used.

8.2.1 Enabling the Security Module in Unix

In a unix environment, a configuration option needs to be stated during the configuration process. The new configuration step will be the command

```
./configure --enable-security
```

instead of the old

```
./configure
```

8.2.2 Enabling the Security Module in Windows

In windows, a line needs to be added to the file "src/winconfig.h". The line to be added is

```
#define MC_SECURITY 1
```

8.2.3 Further Instructions

Furthermore, the option needs to be turned on in the options parameter which is passed into **MC_Initialize()**. The following C code snippet will start a security-enabled agency listening on port 5050.

```
MCAgency_t agency;
MC_AgencyOptions_t options;
MC_InitializeAgencyOptions(&options);
options.enable_security = 1;
agency = MC_Initialize(5050, &options);
```

See more about the MC_AgencyOptions_t type at the description of the MC_Initialize() function in Appendix A on page 73.

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://www.mobilec.org.

[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
enum MC_ThreadIndex_e	
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.
enum MC_Signal_e	
MC_RECV_CONNECTION	Signal activated after an agency accepts a connection.
MC_RECV_MESSAGE	Signal activated after an agency receives an ACL message.
MC_RECV_AGENT	Signal activated after an agency receives a mobile agent.
MC_EXEC_AGENT	Signal activated after a mobile agent is executed.
MC_ALL_SIGNALS	Signal activated after any of the above four events occurs.
enum MC_AgentType_e	
MC_REMOTE_AGENT	Identifier for a remote mobile agent.
MC_LOCAL_AGENT	Identifier for a local mobile agent.
MC_RETURN_AGENT	Identifier for a return mobile agent.
enum MC_AgentStatus_e	
MC_WAIT_CH	Value indicating a mobile agent is waiting to be executed.
MC_WAIT_MESSGSEND	Value indicating a mobile agent is waiting to be exported to another agency.
MC_AGENT_ACTIVE	Value indicating a mobile agent is being executed.
MC_AGENT_NEUTRAL	Value indicating a mobile agent is waiting for an unspecified reason.
MC_AGENT_SUSPENDED	Value indicating a mobile agent is being suspended.
MC_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_Barrier()	Block until all agents in an agency have called this function.
MC_BarrierDelete()	Delete a Mobile-C barrier.
MC_BarrierInit()	Initialize a Mobile-C barrier.
MC_CallAgentFunc()	Call a function defined in an agent.
MC_ChInitializeOptions()	Set the initialization options for a Ch to be used as one AEE in an agency.
MC_CondBroadcast()	Wake up all agents/threads waiting on a condition variable.
MC_CondReset()	Reset a Mobile-C condition variable.
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_CopyAgent()	Perform a deep copy to copy an agent.
MC_DeleteAgent()	Stop and remove an agent from an agency.
MC_DeregisterService()	Deregister a service with the Directory Facilitator.
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_GetAgentArrivalTime()	Get the time when an agent arrives an agency.
MC_GetAgentExecEngine()	Get the AEE associated with a mobile agent in an agency.
MC_GetAgentID()	Get the ID of an agent.
MC_GetAgentName()	Get the name of an agent.
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_GetAllAgents()	Obtain all the agents in an agency.
MC_HaltAgency()	Halt an agency's operation.
MC_Initialize()	Start a Mobile-C agency and return a handle of the launched agency.
MC_InitializeAgencyOptions()	Initialize Mobile-C options.
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_RegisterService()	Register a new service with the Directory Facilitator.

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

Function	Description
MC_ResetSignal()	Reset the Mobile-C signalling system.
MC_ResumeAgency()	Resume an agency's operation.
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_SearchForService()	Search the Directory Facilitator for a service.
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_SendSteerCommand()	Send a command to control a steerable binary space function.
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_Steer()	Set up a steerable binary space function.
MC_SteerControl()	Retrieve a steering command from the mobile agent space.
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_WaitSignal()	Block until one of the signals in the second argument is signalled.

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

```
#include <libmc.h>
#include <stdio.h>
#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif
int main(int argc, char *argv[])
{
    MCAgency_t agency1;
    MCAgency_t agency2;
    MCAgencyOptions_t options;
    int i;

    MCAgent_t agent;
    MCAgent_t agent_copy;

    MC_InitializeAgencyOptions(&options);
    /* We want _all_ the threads on: EXCEPT, the command prompt thread */
    for (i = 0; i < MC_THREAD_ALL; i++) {
        MC_SetThreadOn(&options, i);
    }
    MC_SetThreadOff(&options, MC_THREAD_CP);

    agency1 = MC_Initialize(
        5051,
        &options);
    agency2 = MC_Initialize(
        5052,
        &options);

    while(1) {
```

```
      agent = MC_WaitRetrieveAgent(agency1);
      MC_CopyAgent(&agent_copy, agent);
      MC_SetAgentStatus(agent_copy, MC_WAIT_CH);
      MC_AddAgent(agency2, agent_copy);
      MC_ResetSignal(agency1);
    }

    return 0;
}
```

See Also

MC_CallAgentFunc()

Synopsis
#include <**libmc.h**>
int MC_CallAgentFunc(MCAgent_t $agent$, **const char*** $funcName$, **void*** $returnVal$, **void*** $varg$);

Purpose
This function is used to call a function that is defined in an agent.

Return Value
This function returns 0 on success, or a non-zero error code on failure.

Parameters

$agent$	The agent in which to call a function.
$funcName$	The function to call.
$returnVal$	(Output) The return value of the agent function.
$varg$	An argument to pass to the function.

Description
This function enables a program to treat agents as libraries of functions. Thus, an agent may provide a library of functions that may be called from binary space with this function, or from another agent by the agent-space version of this function.

Example

```
#include <libmc.h>
#include <embedch.h>
#include <stdio.h>

struct arg_struct{
    int a;
    int b;
}arg;

int main(int argc, char *argv[])
{
    MCAgent_t agent;
    ChInterp_t interp;
    int retval;

    /* Init the agency */
    MCAgency_t agency;
    agency = MC_Initialize(
            5051,
            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);
```

```
    }
    /* The following executution of code may be performed two different
       ways: The first way, which is commented out in this example,
       involves retrieving the agent's interpreter with
       MC_GetAgentExecEngine() and using the Embedded Ch api to call
       the function. The second method involves using the Mobile-C
       api to call the function. Both of these methods used here produce
       identical results. */
    arg.a = 50;
    arg.b = 51;
    /*interp = MC_GetAgentExecEngine(agent);
    Ch_CallFuncByName(interp, "hello", &retval, arg); */
    MC_CallAgentFunc(
            agent,
            "hello",
            &retval,
            &arg);
    printf("Value of %d was returned.\n", retval);
    return 0;
}
```

See Also

mc_CallAgentFunc()

MC_ChInitializeOptions()

Synopsis
#include <**libmc.h**>
int MC_ChInitializeOptions(**MCAgency_t** $agency$, **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
$agency$ A Mobile-C Agency.
$options$ 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
    char *chhome;    // Embedded Ch home directory
} 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

```
#include <libmc.h>
#include <string.h>

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

    /*************************************************************
     * A typical home directory of Embedded Ch on Windows would be   *
     * like "C:/Program Files/Company Name/program/embedch". We used *
     * "C:/Ch/toolkit/embedch" for testing purposes.                 *
     *************************************************************/
    char embedchhome[] = "C:/Ch/toolkit/embedch";
    ChOptions_t ch_options;

    ch_options.chhome = malloc(strlen(embedchhome)+1);
    strcpy(ch_options.chhome, embedchhome);

    agency = MC_Initialize(local_port, NULL);
    MC_ChInitializeOptions(agency, &ch_options);
```

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

    return 0;
}
```

See Also

MC_CondReset()

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

Purpose
Reset's a Mobile-C condition variable so that it may be used with MC_CondWait() again.

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

Parameters

$agency$	A Mobile-C agency.
id	The id of the condition variable to signal.

Description
This function reset's a Mobile-C condition variable, setting it back to unsignalled status.
Example
Please see Program 22 on page 38 in Chapter 7.

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

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
Please see Program 17 on page 33 and Program 21 on page 37 in Chapter 7.
See Also
MC_CondDelete(), MC_CondInit(), MC_CondSignal().

MC_CondWait()

Synopsis
#include <**libmc.h**>
int MC_CondWait(**MCAgency_t** $agency$, **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

$agency$	A Mobile-C agency.
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 if the same condition variable is to be used more than once, the function MC_CondReset() must be called on the condition variable.

Example
Please see Program 22 on page 38 in Chapter 7.

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

MC_CopyAgent()

Synopsis
#include <**libmc.h**>
int MC_CopyAgent(MCAgent_t $agent\_out$, **MCAgent_t*** $agent\_in$);

Purpose
Copies an agent.

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

Parameters
$agent\_out$ A copied agent.
$agent\_in$ The agent to copy.

Description
This function is used to perform a deep copy on an Mobile-C agent. It is useful in conjunction with functions that retrieve agents from agencies, since those functions only retrieve a reference to the agent: Not a full copy.
Example

```
#include <libmc.h>
#include <stdio.h>
#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif
int main(int argc, char *argv[])
{
    MCAgency_t agency1;
    MCAgency_t agency2;
    MCAgencyOptions_t options;
    int i;

    MCAgent_t agent;
    MCAgent_t agent_copy;

    MC_InitializeAgencyOptions(&options);
    /* We want _all_ the threads on: EXCEPT, the command prompt thread */
    for (i = 0; i < MC_THREAD_ALL; i++) {
        MC_SetThreadOn(&options, i);
    }
    MC_SetThreadOff(&options, MC_THREAD_CP);

    agency1 = MC_Initialize(
        5051,
        &options);
    agency2 = MC_Initialize(
        5052,
        &options);
```

```
    while(1) {
      agent = MC_WaitRetrieveAgent(agency1);
      MC_CopyAgent(&agent_copy, agent);
      MC_SetAgentStatus(agent_copy, MC_WAIT_CH);
      MC_AddAgent(agency2, agent_copy);
      MC_ResetSignal(agency1);
    }

    return 0;
}
```

See Also

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

```
#include <stdio.h>
#include <stdlib.h>
#include <libmc.h>

int main(int argc, char *argv[])
{
    MCAgency_t agency;
    agency = MC_Initialize(5050, NULL);

    printf("Mobile-C Started\n");
    MC_SendAgentMigrationMessageFile(agency,
                "test1.xml",
                "localhost",
                5051);
    MC_End(agency);
    exit(0);
}
```

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
This function is equivalent to the agent-space version. Please see the example for mc_FindAgentByID() listed on page B on page 116.
See Also
MC_FindAgentByName()

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

```
#include <libmc.h>
#include <embedch.h>
#include <stdio.h>

struct arg_struct{
    int a;
    int b;
}arg;

int main(int argc, char *argv[])
{
    MCAgent_t agent;
    ChInterp_t interp;
    int retval;

    /* Init the agency */
    MCAgency_t agency;
    agency = MC_Initialize(
            5051,
            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);
    }
    /* The following executution of code may be performed two different
       ways: The first way, which is commented out in this example,
       involves retrieving the agent's interpreter with
```

```
       MC_GetAgentExecEngine() and using the Embedded Ch api to call
       the function. The second method involves using the Mobile-C
       api to call the function. Both of these methods used here produce
       identical results. */
    arg.a = 50;
    arg.b = 51;
    /*interp = MC_GetAgentExecEngine(agent);
    Ch_CallFuncByName(interp, "hello", &retval, arg); */
    MC_CallAgentFunc(
            agent,
            "hello",
            &retval,
            &arg);
    printf("Value of %d was returned.\n", retval);
    return 0;
}
```

See Also

MC_FindAgentByID()

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

```
#include <libmc.h>
#include <embedch.h>
#include <stdio.h>

struct arg_struct{
    int a;
    int b;
}arg;

int main(int argc, char *argv[])
{
    MCAgent_t agent;
    ChInterp_t interp;
    int retval;

    /* Init the agency */
    MCAgency_t agency;
    agency = MC_Initialize(
            5051,
            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);
    }
    /* The following executution of code may be performed two different
       ways: The first way, which is commented out in this example,
```

```
       involves retrieving the agent's interpreter with
       MC_GetAgentExecEngine() and using the Embedded Ch api to call
       the function. The second method involves using the Mobile-C
       api to call the function. Both of these methods used here produce
       identical results. */
    arg.a = 50;
    arg.b = 51;
    /*interp = MC_GetAgentExecEngine(agent);
    Ch_CallFuncByName(interp, "hello", &retval, arg); */
    MC_CallAgentFunc(
            agent,
            "hello",
            &retval,
            &arg);
    printf("Value of %d was returned.\n", retval);
    return 0;
}
```

See Also

MC_CallAgentFunc()

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
The return value is of an enumerated type, "enum MC_AgentStatus_e". The enum may be seen in Table A.2 on page 45. The values are

0 , MC_WAIT_CH :	Mobile agent is currently waiting to be executed.
1 , MC_WAIT_MESSGSEND :	Mobile agent is currently waiting to be exported to another agency.
2 , MC_AGENT_ACTIVE :	Mobile agent is currently being executed.
3 , MC_AGENT_NEUTRAL :	Mobile agent is waiting for an unspecified reason.
4 , MC_AGENT_SUSPENDED :	Mobile agent is currently being suspended.
5 , MC_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

```
#include <libmc.h>
#include <stdio.h>
#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif
int main(int argc, char *argv[])
{
    MCAgency_t agency;
    MCAgencyOptions_t options;
    MCAgent_t agent;
    char *str;
    int i;
    MC_InitializeAgencyOptions(&options);

    for (i = 0; i < MC_THREAD_ALL; i++) {
        MC_SetThreadOn(&options, i);
    }
    MC_SetThreadOff(&options, MC_THREAD_CP);  /* Turn off the command prompt */

    agency = MC_Initialize(
            5051,
```

```
            &options);

    /* Retrieve the first arriving agent */
    /* Note: MC_WaitRetrieveAgent() pauses the agency: We'll need to unpause
     * it later with MC_SignalReset() */
    agent = MC_WaitRetrieveAgent(agency);
    printf("The agent status is: %d\n", MC_GetAgentStatus(agent));
    printf("This agent has %d task(s).\n", MC_GetAgentNumTasks(agent));
    str = MC_GetAgentXMLString(agent);
    printf("Agent XML String:\n%s\n", str);
    free(str);
    str = MC_RetrieveAgentCode(agent);
    printf("Agent Code:\n%s\n", str);
    free(str);
    MC_ResetSignal(agency);
    MC_Wait(agency);

    return 0;
}
```

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

```
#include <libmc.h>
#include <stdio.h>
#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif
int main(int argc, char *argv[])
{
    MCAgency_t agency;
    MCAgencyOptions_t options;
    MCAgent_t agent;
    char *str;
    int i;
    MC_InitializeAgencyOptions(&options);

    for (i = 0; i < MC_THREAD_ALL; i++) {
        MC_SetThreadOn(&options, i);
    }
    MC_SetThreadOff(&options, MC_THREAD_CP);  /* Turn off the command prompt */

    agency = MC_Initialize(
            5051,
            &options);

    /* Retrieve the first arriving agent */
    /* Note: MC_WaitRetrieveAgent() pauses the agency: We'll need to unpause
     * it later with MC_SignalReset() */
    agent = MC_WaitRetrieveAgent(agency);
    printf("The agent status is: %d\n", MC_GetAgentStatus(agent));
    printf("This agent has %d task(s).\n", MC_GetAgentNumTasks(agent));
    str = MC_GetAgentXMLString(agent);
```

```
    printf("Agent XML String:\n%s\n", str);
    free(str);
    str = MC_RetrieveAgentCode(agent);
    printf("Agent Code:\n%s\n", str);
    free(str);
    MC_ResetSignal(agency);
    MC_Wait(agency);

    return 0;
}
```

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;              /*!< Threads to start */
    int default_agent_status; /*!< Default agent status */
    int modified;             /*!< unused */
    int enable_security;      /*!< security enable flag */

    /* Following are some thread stack size options:
     * unix/pthreads only! */
    int stack_size[MC_THREAD_ALL];
} 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

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

int main(int argc, char *argv[])
{
  MCAgency_t agency;
  int local_port = 5051;

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

```
  printf("Mobile-C Started\n");
  MC_Wait(agency);
  return 0;
}
```

See Also
MC_End()

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
Please see Program 20 on page 36 in Chapter 7.

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
Please see Program 20 on page 36 in Chapter 7.

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

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

int main(int argc, char *argv[])
{
    MCAgency_t agency;
    MCAgent_t agent;
    int dim, *extent;
    double *data;
    int i, j, size;
    agency = MC_Initialize(5050, NULL);

    printf("MobileC Started\n");
    MC_SendAgentMigrationMessageFile(agency,
                "test.xml",
                "localhost",
                5051);
    MC_WaitSignal(agency, MC_RECV_AGENT);
    agent = MC_FindAgentByName(agency, "mobagent3");
    if (agent == NULL) {
        fprintf(stderr, "Did not receive correct agent. \n");
        exit(1);
    }
    printf("%d tasks.\n", MC_GetAgentNumTasks(agent) );
    for (i = 0; i < MC_GetAgentNumTasks(agent); i++) {
        MC_GetAgentReturnData(
                agent,
                i,
```

```
                (void**)&data,
                &dim,
                &extent );
            printf("Task: %d\n", i);
            size = 1;
            printf("dim is %d\n", dim);
            for (j = 0; j < dim; j++) {
                size *= extent[j];
            }
            printf("Size: %d\n", size);
            printf("Data elements: ");
            for (j = 0; j < size; j++) {
                printf("%f ", data[j]);
            }
            printf("\n\n");
            free(data);
            free(extent);
        }
        MC_ResetSignal(agency);
        MC_End(agency);
        return 0;
}
```

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 MC_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

```
#include <libmc.h>
#include <stdio.h>
#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif
int main(int argc, char *argv[])
{
    MCAgency_t agency;
    MCAgencyOptions_t options;
    MCAgent_t agent;
    char *str;
    int i;
    MC_InitializeAgencyOptions(&options);

    for (i = 0; i < MC_THREAD_ALL; i++) {
        MC_SetThreadOn(&options, i);
    }
    MC_SetThreadOff(&options, MC_THREAD_CP);  /* Turn off the command prompt */

    agency = MC_Initialize(
            5051,
            &options);

    /* Retrieve the first arriving agent */
    /* Note: MC_WaitRetrieveAgent() pauses the agency: We'll need to unpause
     * it later with MC_SignalReset() */
    agent = MC_WaitRetrieveAgent(agency);
    printf("The agent status is: %d\n", MC_GetAgentStatus(agent));
    printf("This agent has %d task(s).\n", MC_GetAgentNumTasks(agent));
    str = MC_GetAgentXMLString(agent);
```

```
    printf("Agent XML String:\n%s\n", str);
    free(str);
    str = MC_RetrieveAgentCode(agent);
    printf("Agent Code:\n%s\n", str);
    free(str);
    MC_ResetSignal(agency);
    MC_Wait(agency);

    return 0;
}
```

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
The MC_SemaphorePost() function usage is very similar to the other binary space synchronization functions. Please see Chapter 7 on page 30 and the demo at "demos/agent_semaphore_example/" for more information.

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
The MC_SemaphorePost() function usage is very similar to the other binary space synchronization functions. Please see Chapter 7 on page 30 for more information.

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., 169.237.104.199 or machine.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., 169.237.104.199 or machine.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

```
#include <stdio.h>
#include <stdlib.h>
#include <libmc.h>

int main(int argc, char *argv[])
{
    MCAgency_t agency;
    agency = MC_Initialize(5050, NULL);

    printf("Mobile-C Started\n");
    MC_SendAgentMigrationMessageFile(agency,
                "test1.xml",
                "localhost",
                5051);
    MC_End(agency);
    exit(0);
}
```

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 returns an integer of enumerated type enum MC_AgentStatus_e. Details about this enumerated type may be found in Table A.2 on page 45.

Example

```
#include <libmc.h>
#include <stdio.h>
#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif
int main(int argc, char *argv[])
{
    MCAgency_t agency1;
    MCAgency_t agency2;
    MCAgencyOptions_t options;
    int i;

    MCAgent_t agent;
    MCAgent_t agent_copy;

    MC_InitializeAgencyOptions(&options);
    /* We want _all_ the threads on: EXCEPT, the command prompt thread */
    for (i = 0; i < MC_THREAD_ALL; i++) {
        MC_SetThreadOn(&options, i);
    }
    MC_SetThreadOff(&options, MC_THREAD_CP);

    agency1 = MC_Initialize(
        5051,
        &options);
    agency2 = MC_Initialize(
        5052,
        &options);
```

```
    while(1) {
      agent = MC_WaitRetrieveAgent(agency1);
      MC_CopyAgent(&agent_copy, agent);
      MC_SetAgentStatus(agent_copy, MC_WAIT_CH);
      MC_AddAgent(agency2, agent_copy);
      MC_ResetSignal(agency1);
    }

    return 0;
}
```

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 is used to set the default agent status for all incoming agents in an agency. By default, every incoming agent is set to status "MC_WAIT_CH", but that may be changed with this function. The agent status is an enumerated type "enum MC_AgentStatus_e", which may be seen in Table A.2 on page 45.
Example

```
MCAgency_t agency;
agency = MC_Initialize(5050, NULL);
MC_SetDefaultAgentStatus(agency, MC_AGENT_NEUTRAL);

/* etc... */
```

See Also
MC_GetAgentStatus()

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_Steer()

Synopsis
#include <**libmc.h**>
int MC_Steer(**MCAgency_t** attr, **int(*)(void* data)** funcptr, **void*** arg);

Purpose
The MC_Steer function initialized and runs a function containing an algorithm. The function enables the steering functionality of the algorithm so that it may accept command during runtime to change the execution of the algorithm. For more information, please see the example and the demo located in the demos/steer_example/ directory.
Return Value
The function returns 0 on success, or a non-zero error code on failure.

Description
The **MC_Steer** function is designed execute an algorithm in a fashion which enables that algorithm to be steered or modified on-the-fly during runtime. See the demo and the example for more details.

Example

```
#include <stdio.h>
#include <libmc.h>
#ifdef _WIN32
#include <windows.h>
#endif

int algorithm(void* boo);

int main() {
    MCAgency_t agency;
    int local_port = 5050;

    agency = MC_Initialize(local_port, NULL);

    MC_Steer(
        agency,
        &algorithm,
        NULL
        );

    MC_End(agency);
    return 0;
}

int algorithm(void* boo)
{
  int i=0;
  MC_SteerCommand_t command;
  while(1) {
#ifndef _WIN32
    sleep(1);
#else
```

```
    Sleep(1000);
#endif
    printf("%d \n", i);
    i++;
    command = MC_SteerControl();
    if(
        command == MC_RESTART ||
        command == MC_STOP
      )
    {
      return 0;
    }
  }
}
```

See Also

MC_SteerControl()

MC_SteerControl()

Synopsis
#include <**libmc.h**>
int MC_SteerControl(**void**);

Purpose
This function is used to enable Mobile-C as a steerable computational platform. See the example following for more information, as well as the demo provided in the directory demos/steer_example.

Return Value
This function returns the current steer command. The command is of type **enum MC_Steer_Command_e**. This enumerated type contains the following definitions:

MC_RUN	Continue the algorithm.
MC_SUSPEND	Pause the algorithm.
MC_RESTART	Restart the algorithm from the beginning.
MC_STOP	Stop the algorithm.

Description
MC_SteerControl controls the execution of an algorithm in binary space. This function is meant to retrieve the current requested command for the algorithm, but it is up to the algorithm implementation to actually implement these behaviours. See the example and the demo for more details.
Example

```
#include <stdio.h>
#include <libmc.h>
#ifdef _WIN32
#include <windows.h>
#endif

int algorithm(void* boo);

int main() {
    MCAgency_t agency;
    int local_port = 5050;

    agency = MC_Initialize(local_port, NULL);

    MC_Steer(
        agency,
        &algorithm,
        NULL
        );

    MC_End(agency);
    return 0;
}

int algorithm(void* boo)
{
  int i=0;
  MC_SteerCommand_t command;
  while(1) {
```

```
#ifndef _WIN32
    sleep(1);
#else
    Sleep(1000);
#endif
    printf("%d \n", i);
    i++;
    command = MC_SteerControl();
    if(
        command == MC_RESTART ||
        command == MC_STOP
      )
    {
      return 0;
    }
  }
}
```

See Also

MC_Steer()

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
Please see Chapter 7 on synchronization on page 30 for more details about using this function.

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
Please see Chapter 7 on synchronization on page 30 for more details about using this function.

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 MC_AGENT_NEUTRAL mode if the mobile agent is tagged as 'persistent', or is terminated and flushed otherwise.

Example
This function is identical to the agent-space counterpart. Please see the example listed under mc_TerminateAgent() on page 137.

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

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

int main(int argc, char *argv[])
{
  MCAgency_t agency;
  int local_port = 5051;

  agency = MC_Initialize(local_port, NULL);

  printf("Mobile-C Started\n");
  MC_Wait(agency);
  return 0;
}
```

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

```
#include <libmc.h>
#include <stdio.h>
#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif
int main(int argc, char *argv[])
{
    MCAgency_t agency;
    MCAgencyOptions_t options;
    MCAgent_t agent;
    char *str;
    int i;
    MC_InitializeAgencyOptions(&options);

    for (i = 0; i < MC_THREAD_ALL; i++) {
        MC_SetThreadOn(&options, i);
    }
    MC_SetThreadOff(&options, MC_THREAD_CP);  /* Turn off the command prompt */

    agency = MC_Initialize(
            5051,
            &options);

    /* Retrieve the first arriving agent */
    /* Note: MC_WaitRetrieveAgent() pauses the agency: We'll need to unpause
     * it later with MC_SignalReset() */
    agent = MC_WaitRetrieveAgent(agency);
    printf("The agent status is: %d\n", MC_GetAgentStatus(agent));
    printf("This agent has %d task(s).\n", MC_GetAgentNumTasks(agent));
    str = MC_GetAgentXMLString(agent);
```

```
    printf("Agent XML String:\n%s\n", str);
    free(str);
    str = MC_RetrieveAgentCode(agent);
    printf("Agent Code:\n%s\n", str);
    free(str);
    MC_ResetSignal(agency);
    MC_Wait(agency);

    return 0;
}
```

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:

MC_RECV_CONNECTION :	Continue after a connection is initialized.
MC_RECV_MESSAGE :	Continue after a message is received.
MC_RECV_AGENT :	Continue after an agent is received.
MC_EXEC_AGENT :	Continue after an agent is finished executing.
MC_ALL_SIGNALS :	Continue after any one of the above events occurs.

In order to wait on a custom combination of signals, the bitwise '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 **agent.c** Furthermore, a number of enumerations, data types, and special variables are declared in **agent.c** for each agent interpreter by the agency. These enums, data types, special variables, and functions are all considered "built-in" in the mobile agent space as no header file or extra code is needed to access them. They are declared through Embedded Ch functions, **Ch_DeclareFunc()**, **Ch_DeclareVar()** and **Ch_DeclareTypedef()** [12] Note that the C/C++ script space is also referred to the mobile agent space in this user's guide.

All enumerations and special variables nay be found in Tables B.1, B.2 and B.3, respectively. The defined data type and function prototypes are listed in Tables B.4 and B.5, respectively. **agent.c** can be found in directories 'src'.

Table B.1: enum MC_SteerCommand_e : This enumerated type lists commands that may be used with the mc_SendSteerCommand() function.

Data Type	Description
MC_RUN	Start/continue an algorithm.
MC_SUSPEND	Pause an algorithm.
MC_RESTART	Restart an algorithm for initial values.
MC_STOP	Stop an algorithm.

Table B.2: enum mc_AgentStatus_e: This enumerated type defines the current execucion state of a mobile agent.

0 , MC_WAIT_CH :	Mobile agent is currently waiting to be executed.
1 , MC_WAIT_MESSGSEND :	Mobile agent is currently waiting to be exported to another agency.
2 , MC_AGENT_ACTIVE :	Mobile agent is currently being executed.
3 , MC_AGENT_NEUTRAL :	Mobile agent is waiting for an unspecified reason.
4 , MC_AGENT_SUSPENDED :	Mobile agent is currently being suspended.
5 , MC_WAIT_FINISHED :	Mobile agent has finished execution and is waiting for removal.

Table B.3: A table of pre-defined agent-space variables. These are considered 'built-in' in agent space as no additional header file is required to access these variables.

Variable Name	Description
int mc_agent_id	Holds the unique integer id assigned by the Agency to the agent.
char mc_agent_name[]	Holds the agent's name.
void* mc_current_agent	Holds a pointer itself.
char mc_host_name[]	Holds the agency's hostname.
int mc_host_port	Holds the port of the current agency.
int mc_task_progress	Contains the current task number of the agent.
int mc_num_tasks	Contains the total number of tasks an agent has.

Table B.4: Data type for functions in the C/C++ script space.

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

Table B.5: Functions in the C/C++ script space.

Function	Description
mc_AddAgent()	Add a mobile agent into an agency.
mc_Barrier()	Block until all agents in an agency have called this function.
mc_BarrierDelete()	Delete a Mobile-C barrier.
mc_BarrierInit()	Initialize a Mobile-C barrier.
mc_CallAgentFunc()	Call a function defined in an agent.
mc_CondBroadcast()	Wake up all agents/threads waiting on a condition variable.
mc_CondReset()	Reset a Mobile-C condition variable.
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_DeleteAgent()	Stop and remove an agent from an agency.
mc_DeregisterService()	Deregister a service with the Directory Facilitator.
mc_End()	Terminate a Mobile-C agency.
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_GetAgentArrivalTime()	Get the time when an agent arrives an agency.
mc_GetAgentExecEngine()	Get the AEE associated with a mobile agent in an agency.
mc_GetAgentID()	Get the ID of an agent.
mc_GetAgentName()	Get the name of an agent.
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_GetAllAgents()	Obtain all the agents in an agency.
mc_HaltAgency()	Halt an agency's operation.
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_RegisterService()	Register a new service with the Directory Facilitator.

Table B.5: Functions in the C/C++ script space (contd.).

Function	Description
mc_ResumeAgency()	Resume an agency's operation.
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_SearchForService()	Search the Directory Facilitator for a service.
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_SendSteerCommand()	Send a command to control a steerable binary space function.
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_AddAgent()

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

Purpose
Add a mobile agent into an agency.

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

Parameters
$agent$ An initialized mobile agent.

Description
This function adds a mobile agent to an agency.

Example
Please see the example for MC_AddAgent() on page 48.

See Also

mc_CallAgentFunc()

Synopsis
#include <**libmc.h**>
int mc_CallAgentFunc(**MCAgent_t** $agent$, **const char*** $funcName$, **void*** $returnVal$, **void*** $varg$);

Purpose
This function is used to call a function that is defined in an agent.

Return Value
This function returns 0 on success, or a non-zero error code on failure.

Parameters

$agent$	The agent in which to call a function.
$funcName$	The function to call.
$returnVal$	(Output) The return value of the agent function.
$varg$	An argument to pass to the function.

Description
This function enables a program to treat agents as libraries of functions. Thus, an agent may provide a library of functions that may be called from binary space with this function, or from another agent by the agent-space version of this function.

Example

```
<?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 number_of_elements="0" name="no-return" complete="0" server="localhost:5050">
       </DATA>
  <AGENT_CODE>
          <![CDATA[
#include <stdio.h>
struct arg_struct {
    int a;
    int b;
};
int main()
{
    MCAgent_t agent;
    int retval;
    /* Search Return Variables */
    char** agentNames;
```

```
    char** serviceNames;
    int *agentIDs;
    int numResults;

    /* Argument Struct */
    struct arg_struct arg;

    /* Search for addition service */
    printf("\n\n\nSearching for addition service.\n");
    mc_SearchForService(
            "addition",
            &agentNames,
            &serviceNames,
            &agentIDs,
            &numResults );
    printf("Done searching.\n");
    if (numResults < 1) {
        printf("No agents with service 'addition' found.\n");
        exit(0);
    }

    /* Just get the first hit */
    printf("Using agent %s for addition.\n", agentNames[0]);
    agent = mc_FindAgentByID(agentIDs[0]);

    arg.a = 44;
    arg.b = 45;
    mc_CallAgentFunc(agent, "addition", &retval, &arg);
    printf("Result of addition %d + %d is %d.\n", arg.a, arg.b, retval);

    /* Now search for multiplication service */
    printf("\n\n Searching for Multiplication service...\n");
    mc_SearchForService(
            "multiplication",
            &agentNames,
            &serviceNames,
            &agentIDs,
            &numResults );

    if (numResults < 1) {
        printf("No agents with service 'multiplication' found.\n");
        exit(0);
    }

    printf("Using agent %s for multiplication.\n", agentNames[0]);
    agent = mc_FindAgentByID(agentIDs[0]);
    mc_CallAgentFunc(agent, "multiplication", &retval, &arg);
    printf("Result of multiplication %d * %d is %d.\n", arg.a, arg.b, retval);

    /* Now lets try to deregister a service */
    mc_DeregisterService(
            agentIDs[0],
            serviceNames[0]
            );

    return 0;
}
        ]]>
        </AGENT_CODE>
```

```
      </TASK>
      </AGENT_DATA>
    </MOBILE_AGENT>
  </MESSAGE>
</GAF_MESSAGE>
```

See Also

MC_CallAgentFunc()

mc_CondReset()

Synopsis
int mc_CondReset(int id**);**

Purpose
Reset a Mobile-C Condition variable for re-use.

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

Parameters

id	The id of the condition variable to signal.

Description
This function resets a used condition variable, setting it's state back to an unsignalled state. A Mobile-C condition variable will remain in a signalled state indefinitely until this function is called.

Example
See Program 18 on page 34 and Program 19 on page 35 in Chapter 7.

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

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

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 Program 18 on page 34 and Program 19 on page 35 in Chapter 7.
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 Program 18 on page 34 and Program 19 on page 35 in Chapter 7.

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

```
<?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:5050</HOME>
    <TASK task="1" num="0">
     <DATA number_of_elements="0" name="no-return"
           complete="0" server="localhost:5051">
     </DATA>
     <AGENT_CODE>
      <![CDATA[
#include <stdio.h>

struct arg_struct {
    int a;
    int b;
};

int main() {
    MCAgent_t agent;
    int retval;
    /* Search Return Variables */
    char** agentNames;
    char** serviceNames;
    int *agentIDs;
    int numResults;

    /* Argument Struct */
    struct arg_struct arg;
```

```
    /* Search for addition service */
    printf("\n\n\nSearching for addition service.\n");
    mc_SearchForService(
            "addition",
            &agentNames,
            &serviceNames,
            &agentIDs,
            &numResults);
    printf("Done searching.\n");
    if (numResults < 1) {
        printf("No agents with service 'addition' found.\n");
        exit(0);
    }

    /* Just get the first hit */
    printf("Using agent %s for addition.\n", agentNames[0]);
    agent = mc_FindAgentByID(agentIDs[0]);

    arg.a = 44;
    arg.b = 45;
    mc_CallAgentFunc(agent, "addition", &retval, &arg);
    printf("Result of addition %d + %d is %d.\n", arg.a, arg.b, retval);
    return 0;
}
      ]]>
     </AGENT_CODE>
    </TASK>
   </AGENT_DATA>
  </MOBILE_AGENT>
 </MESSAGE>
</GAF_MESSAGE>
```

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

```
<?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:5051">
     </DATA>
     <AGENT_CODE>
      <![CDATA[
#include <stdio.h>
int main()
{
    MCAgent_t tmp;
    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>
```

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. See Table B.2 on page 106.

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
This function is identical to the binary space version, MC_GetAgentStatus(). Please see the documentation for MC_GetAgentStatus on page 68 for an 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
This function has identical behaviour with the its binary-space counterpart, MC_GetAgentXMLString(). Please see the documentation for MC_GetAgentXMLString() on page 71
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
Please see Program 16 on page 32, Program 17 on page 33, and Chapter 7 on page 30 for more details.

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
Please see Program 16 on page 32, Program 17 on page 33, and Chapter 7 on page 30 for more details.

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 MC_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
Please see the example under MC_RetrieveAgentCode() on page 81.

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
The MC_SemaphorePost() function usage is very similar to the other binary space synchronization functions. Please see Chapter 7 on page 30 and the demo at "demos/agent_semaphore_example/" for more information.

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
The MC_SemaphorePost() function usage is very similar to the other binary space synchronization functions. Please see Chapter 7 on page 30 and the demo at "demos/agent_semaphore_example/" for more information.

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., 169.237.104.199 or machine.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., 169.237.104.199 or machine.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
Please see the example for MC_SendAgentMigrationMessageFile() on page 86.
See Also

mc_SendSteerCommand()

Synopsis
#include <**libmc.h**>
int mc_SendSteerCommand(**MCAgency_t** attr, **int(*)(void* data)** funcptr, **void*** arg);

Purpose
The mc_SendSteerCommand function sends a computational steering command to the algorithm at the agent's current agency.
Return Value
The function returns 0 on success, or a non-zero error code on failure.

Description
This function enables mobile agents to send steer commands to steering-enables algorithms running at the agent's local agency. See the demo at demos/steer_example/ for more details.

Example

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

<!DOCTYPE myMessage SYSTEM "gafmessage.dtd">

<GAF_MESSAGE>
 <MESSAGE message="MOBILE_AGENT">
  <MOBILE_AGENT>
   <AGENT_DATA>
    <NAME>resume_agent</NAME>
    <OWNER>IEL</OWNER>
    <HOME>localhost:5051</HOME>
    <TASK task="1" num="0">
     <DATA name="no-return"
           complete="0"
           server="localhost:5050">
     </DATA>
      <AGENT_CODE>
       <![CDATA[
#include <stdio.h>
int main() {
    printf("Resuming Agent...");
    mc_SendSteerCommand(MC_RUN);
    return 0;
}
       ]]>
      </AGENT_CODE>
    </TASK>
   </AGENT_DATA>
  </MOBILE_AGENT>
 </MESSAGE>
</GAF_MESSAGE>
```

See Also
MC_Steer(), MC_SteerControl()

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 returns an integer of enumerated type enum MC_AgentStatus_e. Details about this enumerated type may be found in Table B.2 on page 106.

Example
Please see the example for MC_SetAgentStatus() on page 87.

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 of type enum mc_AgentStatus_e. See Table B.2 on page 106 for a complete listing of the enumerated type.
Example
Please see the example for MC_SetDefaultAgentStatus() on page 89.
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
Please see the example for MC_SyncDelete() on page 96.

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

```
<?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>
```

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 MC_AGENT_NEUTRAL mode if the mobile agent is tagged as 'persistent', or is terminated and flushed otherwise.

Example

```
<?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:5051">
     </DATA>
     <AGENT_CODE>
      <![CDATA[
#include <stdio.h>
int main()
{
    MCAgent_t tmp;
    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>
```

See Also

Index