Integrating Application Programs for Bioinformatics Using a Web Browser

Yutaka Ueno **Kiyoshi Asai** **Masanori Arita**
{ueno, asai, arita}@etl.go.jp
Machine Understanding Division, Electrotechnical Laboratory
1-1-4 Umezono, Tsukuba, Ibaraki 305-8568, Japan

Abstract

We have constructed a general framework for integrating application programs with control through a local Web browser. This method is based on a simple inter-process message function from an external process to application programs. Commands to a target program are prepared in a script file, which is parsed by a message dispatcher program. When it is used as a helper application to a Web browser, these messages will be sent from the browser by clicking a hyper-link in a Web document. Our framework also supports pluggable extension-modules for application programs by means of dynamic linking. A prototype system is implemented on our molecular structure-viewer program, MOSBY. It successfully featured a function to load an extension-module required for the docking study of molecular fragments from a Web page. Our simple framework facilitates the concise configuration of Web softwares without complicated knowledge on network computation and security issues. It is also applicable for a wide range of network computations processing private data using a Web browser.

1 Introduction

Recent progress in the genome project and protein science obliges researchers to analyze genomic sequences and protein structures in different methods and from various angles. Today, researchers heavily use the World Wide Web, providing a wide range of public information on bioinformatics through its simple mechanism of Hypertext Markup Language (HTML) documents [15]. On-line accesses to large databases are usually provided by Common Gateway Interface (CGI) in a client-server model [8].

The dependence on the network service, however, makes it difficult to process private data on stand-alone personal computers. For example, when a researcher in a pharmaceutical company tries to analyze a confidential sequence or atomic coordinates, a transaction through the Internet is unfavorable. Also, the security issues of the network make the configuration of a software much complicated in a client-server system. Since advanced hardware technologies have reinforced the performance of personal computers, one solution for processing private data is computation on the client side.

For this purpose, Java programming language provides a portable framework for network-based application softwares. The applet run-time model has been deployed in most Web browsers, and ensures secure code-execution with an restricted access to local resources. However, this method raises a problem in saving a computed result on a local disk [5]. Moreover, in addition to practical disadvantages in speed and portability, its programming technique became too complex for users to adapt. Since a large number of tools were developed in previous programming environments and are widely used in bioinformatics, a simpler way to integrate native programs is expected.

We have enhanced a molecular structure viewer program, MOSBY [13] in respect of private data processing and software integration. This paper describes our simple inter-process message functions added in MOSBY to accept commands from an external process, thereby facilitating advanced software integration. This scheme runs on different platforms, and allows users to add new functions. The software architecture is based on our programming library named ASHLEY (Application Support Hybrid Library for Easy programming), which features a component-based software architecture in a plug-in style [12], and a high-throughput graphics library [13] as reported previously. Message functions introduced in this paper were actually implemented in the library without loss of portability.

The message functions introduced in this paper may be called a software scripting, a protocol which automates programming tasks and integrates application softwares. With these functions, an application program can receive commands from a Web browser through a script file linked from a HTML document. As similar scripting tools, Perl and Tcl play an important role in customizing in-house software systems in life-science research. However, they require detailed security consideration on a Web based integration. For another example, UserTalk script in Frontier, a Web system software (UserLand Software Inc.), provides a protocol for a Web browser to communicate with application programs. However, this object-oriented framework is complicated and requires substantial amount of time for a user to learn.

As above, our work is not the first attempt to let processes to communicate through a network. EyeChem [4] is a molecular structure viewer with networking functions using a Web browser. RasMol [10] is widely used as a helper application for viewing molecular structures on Web pages with its scripting commands. In these cases, however, a Web page can not send a command to the currently running application program. Our novel idea is to prepare a 'steering' button for each application program. It is a kind of graphical user interfaces to integrate local application programs to a Web browser.

2 Method

2.1 Overview

Our method is based on a simple message communication between application programs, passing commands in place of keyboard inputs. We defined an application message protocol and its script file format. A message dispatcher program CAY, which stands for 'Command to ASHLEY', reads messages in a script file in order to dispatch and to send messages to application programs. The message protocol is described as a procedure call with three arguments in our message script:

```
caymessage("name", "command", "parameter")
```

This command in a script file sends a message of `"command"` to an application `"name"` with data `"parameter"` as if it were typed from a keyboard. Usually a Web browser invokes a helper application, when it processes the data of a specific MIME (Multipurpose Internet Mail Extensions) [14] type in a Web page. Our protocol is also in MIME format so that a HTML document on a Web page can start a CAY script to control custom, or in-house, application programs (Fig. 1).

2.2 Inter-Process Message

The style of message-passing follows *a single listener and multiple senders* model. Each listener must be uniquely registered by its ascii name in a local system. Messages are queued in the system, and a sender receives a reply after its listener processes its message. The programming library ASHLEY provides its Application Program Interface (API) in C language as summarized in Appendix. An application program can become a message listener by an API `YoyOnMessage()` with its name and a handler function, which is called when the application receives a message. The message handler takes its command and parameter in character-strings as arguments.

An application can become a message sender by an API `YiyMessage()` with two character-strings that will be passed to the handler function of a listener. Sending a message blocks the execution of the calling process (sender) until the listener actually completes the handler function. The handler

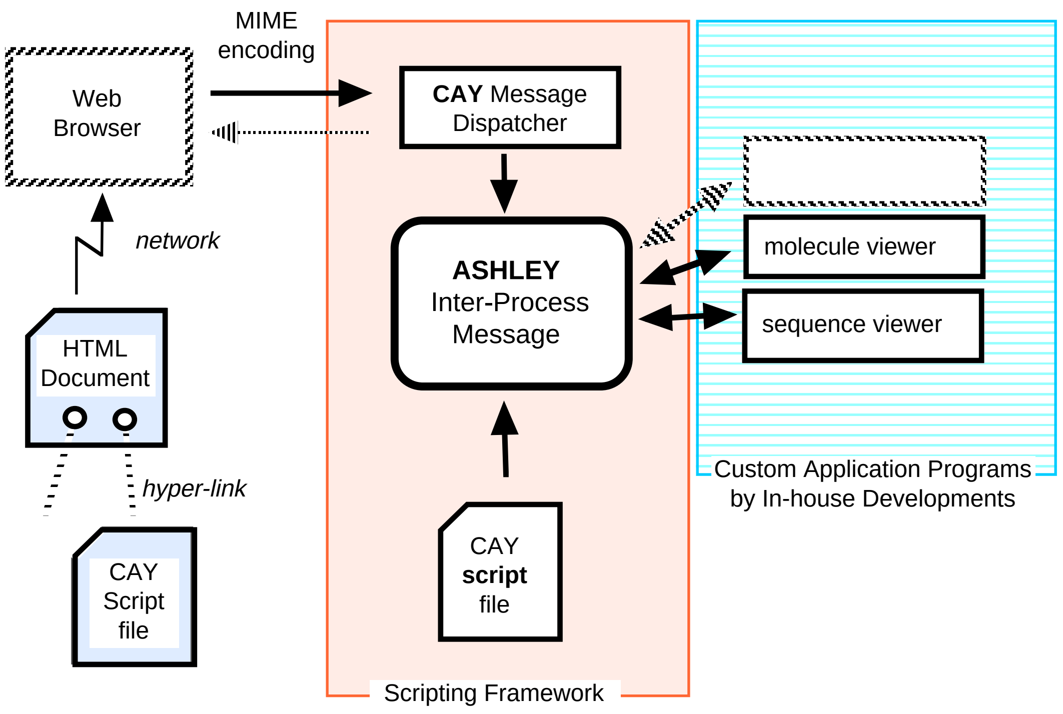


Figure 1: Overview of Software Integration using a Web Browser. Application programs built on ASHLEY programming library are equipped with inter-process message functions. A Web browser can communicate with them through CAY, a message dispatcher program.

function is a kind of call-back procedure responding to a message event. The size of string data in this message is limited, but another API supports a larger storage data.

2.3 Software Component Architecture

We briefly explain the component-based software architecture of ASHLEY, which facilitates a pluggable extension of application softwares [12]. In this framework, a user can easily adapt application programs for scripting.

It is basically a memory table in a program, indexed by their names (ascii strings) as retrieval keys. There is a table for each program module to provide software services (Fig. 2). Each module assigns a name to its command-handler function which also takes two arguments in character-strings. It means that a command is processed in almost the same manner as the inter-process message handling using ASHLEY API (See Appendix for detail.). A command-handler function is chosen according to the given module name, and is executed to tell a program module what to do.

ASHLEY also provides a menu and graphical user interface. Selecting one menu-item issues corresponding command-strings through its API, which mimics keyboard input. The command-handler is nothing but a call-back procedure in this event-driven programming framework.

The pluggable extension module is a dynamically linked code which updates the table of the command-handler when it is loaded. In other words, an application program can equip new commands only by loading those modules.

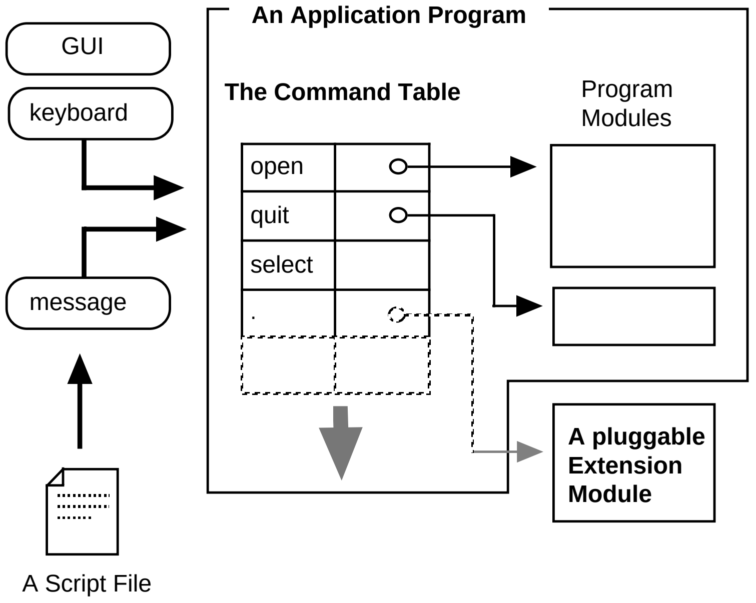


Figure 2: Scripting Framework of An Application Program. ASHLEY programming library provides a command table for associated function entries of the application program. Users can enter command from either a keyboard, GUI or a script file.

2.4 Message Script

There are standard commands for application programs: `open`, `launch` and `quit`. The `launch` command starts an application program if it is registered as a listener in a local system. If the program is already running, it just brings it to the foreground. To enclose a data file in a script, the data file is simply attached after the following CAY script.

```
caymessage("mosby","open",caystrip(".pdb"))
                 $endinput
                   .....
```

The function, `caystrip()`, creates a temporary file of the attached data after a keyword `$endinput` until the end of file. Then the message is sent to the specified application program with the created file name.

An argument to the function `caystrip()` is an extension to the temporary file so that application program can easily recognize it. Optional parameters for the program can also be specified in its command-strings. In the next example, the command is `launch` which ends at a space character and the rest will be passed to a program `clastalw` as a command-line argument.

```
caymessage("clastalw","launch -ktuple=2 -matrix=pam100",caystrip(".seq"))
```

Since our framework is independent of a console and the standard-input, programs which read data from the standard-input need an appropriate redirection from a file to be engaged in this method.

2.5 PAX Format for a Binary Executable File

For sending binary data or an executable code, we embed the CAY script messages in PAX format. It is an archive of data indexed with ascii labels organized from the end of the file (Fig. 3). An archive

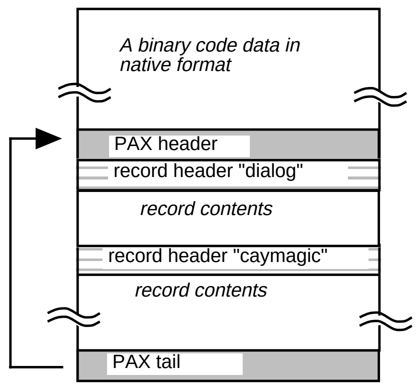


Figure 3: PAX File Format for Binary Files. The PAX format has a special tail data at the end of file which describes its native format of the binary code and the PAX header position. Multi-part data are stored after a record header containing its data size sequentially from the PAX header. Note that the native binary data must be structured so that loaders of the operating systems ignore attached PAX record.

record indexed as "caymagic" corresponds to a CAY script file. The next example is a program loader named `"launch"`, which executes the code in the given file.

```
caymessage("launch","open",paxcode())
```

Function `paxcode()` creates a temporary file of the codes in PAX format. The PAX archive may also contain multiple binary executables, so that we can extract and select an appropriate code for each local operating system. The use of the function `paxcode()` always spawns a dialog box to ensure the execution of the code, because a foreign code potentially violates the security of a local system.

The next example sends a dynamically loadable code to a currently running application `mosby`, which recognizes the command `"plugin"` to execute the code as a pluggable extension module.

```
caymessage("mosby","plugin",paxcode())
```

An application program has to guarantee the security of the execution of a foreign code in a local system. The function `paxcode()` can be implemented with some security checks for viral or malicious codes or with an appropriate castration. The PAX format also allows multi-part data in a single file.

3 Implementation

3.1 A Message Dispatcher Program CAY

A message dispatcher program, CAY was implemented with three indispensable scripting primitives: `caymessage()`, `caystrip()`, and `paxcode()`. When reading a script message, it can override messages to program loader named `"launch"` in order to act as an installer for a binary program code in PAX format, with an appropriate prompt to a user.

The code was developed on a workstation (Silicon Graphics Inc., Indy R4400SC, IRIX 6.4 operating system) and a personal computer (Dell Computer Corp., GXi5200, Pentium 200 MHz, Linux 2.0.27 operating system and gcc 2.7.2). ASHLEY inter-process message function was implemented using the window property function in X-Window system (Release 6, XFree86-3.2). A message listener creates a property of the message name on the root window to wait a message by a Xlib function `XSetSelectionOwner()`. A sender creates an window with message context and send them using

XConvertSelection() so that it can receive a XPropertyChangedEvent from the listener as a reply or error code if a listener is absent.

3.2 Cooperation with a Web Browser

Netscape Navigator 4.05 (Netscape Communications Corp.) was used for a Web browser. A message dispatcher program CAY was installed as a helper application in the MIME type application/x-cay for files with extension .cay. When it handles a CAY script linked from a HTML document, the URL address should be specified to allow further improvement in security. It is provided by %u to the command line arguments to the helper application as well as %s for its cache file. After this setting, local CAY script files with an extension .cay can be executed. It is simple to build and maintain a Web page with scripts: once this MIME type is registered in a http server, those pages are ready to serve on a network.

On the other hand, if an application program wants to send a message to a Web browser, the messaging function may differ according to the browser and the operating system [11, 16]. This process is delegated by another wrapper program which takes an ASHLEY message "www" and forward the message to the corresponding function of browsers. This is a portable solution to communicate with a selected browser on various operating systems. For example, command "openURL" request a Web browser to show the specified Uniform Resource Locator (URL) as its parameter.

4 Result

4.1 Application to MOSBY

Since a program MOSBY is written in ASHLEY in a component-oriented manner, it is straightforward to offer some scripting command. In addition to simple forwarding messages to internal command name, a command "plugin" is also offered to load a pluggable extension code. A demo Web page is available at http://www.etl.go.jp/~ueno/demo99/ (Fig. 4). After installing the CAY dispatcher program, the page can demonstrate MOSBY program with interactive messages by the following links:

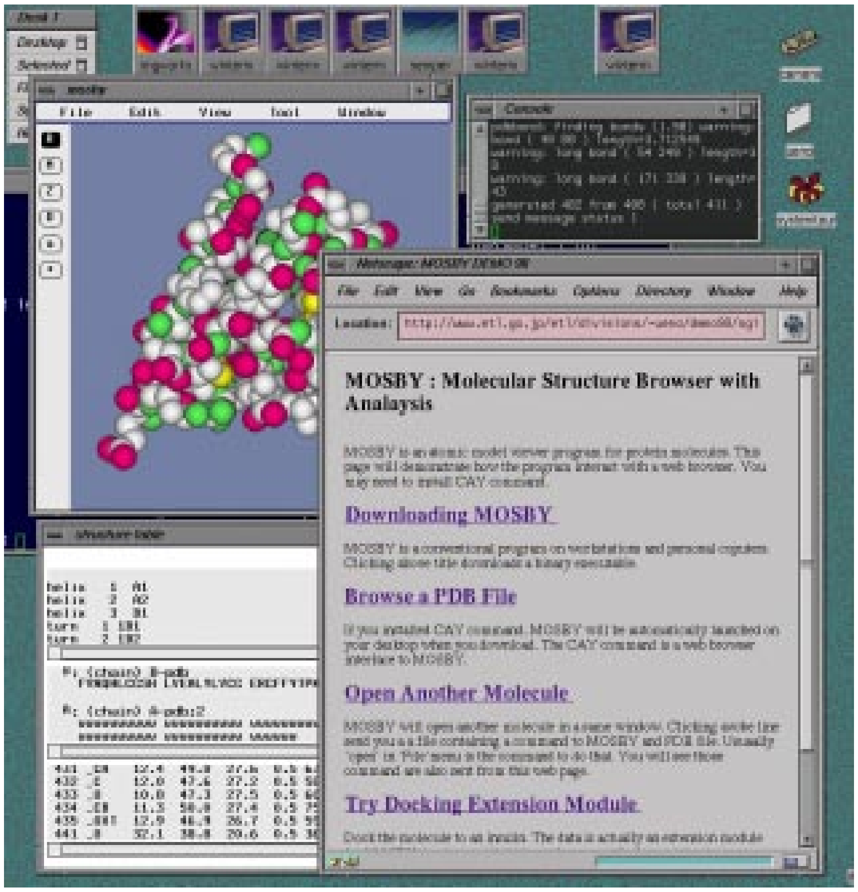
1. Downloading MOSBY: This link sends a binary executable of MOSBY, which will be launched automatically. This can be used for an alternative to a Java applet.

2. Browsing a PDB File: This link sends a CAY script file including a PDB file, which MOSBY will display.

3. Open Another Molecule: This link also sends a CAY file and appends an another molecule to the current window of MOSBY.

4. Loading Docking Study Extension Module: This link provides a pluggable application extension to MOSBY, which will be installed automatically. The second fragment loaded by the previous link can be rotated or moved by the switching mode of mouse operation.

Although this demo skips the calculation of collisions and of molecular force field energy for a docking study, different algorithms and energy calculations in pluggable extension modules were tested. Our work is the first to achieve the binding command sequences and code fragments for a molecular structure viewer with a Web page through a network. It is notable that the size of ASHLEY pluggable extension module is small, about 33 Kbytes in this example. The distribution our code of MOSBY and ASHLEY will be released at http://www.etl.go.jp/~ueno/bioinfo.


mosby
File Edit View Tool Window
Console
Netscape: MOSBY DEMO 98
File Edit View Go Bookmarks Options Directory Window Help
Location:
structure table

```
cobbond: finding bonds (1.98) warning:
bond ( 40 80 ) length=1.12548
warning: long bond ( 54 240 ) length=
3
warning: long bond ( 171 338 ) length=
43
generated 402 from 408 ( total 411 )
send message status )
```

```
helix   1  A1
helix   2  A2
helix   3  B1
turn    1  LB1
turn    2  LB2
```

MOSBY : Molecular Structure Browser with Analaysis

MOSBY is an atomic model viewer program for protein molecules. This page will demonstrate how the program interact with a web browser. You may need to install CAY command.

Downloading MOSBY

MOSBY is a conventional program on workstations and personal computers. Clicking above title downloads a binary executable.

Browse a PDB File

If you installed CAY command, MOSBY will be automatically launched on your desktop when you download. The CAY command is a web browser interface to MOSBY.

Open Another Molecule

MOSBY will open another molecule in a same window. Clicking above line send you a a file containing a command to MOSBY and PDB file. Usually 'open' in 'File' menu is the command to do that. You will see those command are also sent from this web page.

Try Docking Extension Module

Dock the molecule to an insulin. The data is actually an extension module

Figure 4: A Snapshot of our MOSBY Demonstration. An Insulin monomer was depicted through a script file in the HTML document.

5 Discussions

5.1 MIME type and Helper Application

In a standard integration with Web browsers, a helper application like RasMol [10] can not interactively receive a message from a Web page. Our framework solved this problem by inter-process message passing. The support of one MIME type for each application program [4] is another shortcoming of the standard helper application. Because increasing numbers of MIME types makes their file extensions confusing, Web browsers of end-uses easily fail to synchronize. Since our CAY script uses only one MIME type, no more MIME types are required for extended programs in a customized software system.

Multi-media contents are often handled by a plug-in module of a browser, in which the content is embedded in its window using an `OBJECT` tag. In this case, the information context is regulated by the Web browser, and will disappear on closing or switching the window. In contrast, our framework let a helper application run regardless of the existence of a Web page. A Web page is merely a control panel with steering buttons, sending commands to the application. This is a more favorable scheme to integrate several applications through a browser.

5.2 Security Considerations

When a HTML document drives a local application program, two issues must be considered for security purpose: protection of a local resource and information leakage. Instead of grappling with their fundamental problems, we entrust these issues to an individual application program. For this reason, if an application program handles a command to delete files, it is subject to attack by a viral script. Fortunately, most bioinformatics computations do not need those risky command.

Limiting privileges for a general binary executable and pluggable application extension code for security reasons often sacrifices useful functions for an application program. The digital signature [5] used in Java for authentication could be employed, but may result in endless patch-ups for security holes. In our framework, the code execution is application-specific and it is also possible to equip special defense modules. It is easy to update, and the chance of a intrusion is much less. Unlike a class library in Java, a foreign code does not interfere with other programs.

5.3 Comparisons with Related Works

Windows operating system (Microsoft Corp.) has a mechanism called ActiveX to embed a code in a HTML document to be executed in a local Web browser [2]. Our method can also send a code fragments through a browser, but differs in a point that the execution is subject to the application program. The application program is responsible for whether taking a full advantage of computation with local resources or running a risk of executing code from the Internet.

There are also related works in terms of local code execution: WebApp [9] focused on the installation of programs, and Flypaper [7] uses AppleScript (Apple Computer Inc.). However, they lack portability for UNIX operating systems, where most programs for bioinformatics have been developed. Since ASHLEY programming library is already ported to Windows and MacOS, it is also possible to implement the described method for those platform. EyeChem [4] is also a molecular structure viewer which can communicate with a Web browser. But its software architecture depends on a specific visualization tool [1] which lacks portability for practical use.

5.4 Scriptable Application

Scriptable application is a terminology of Apple computer for a graphical user interface program, which accepts commands from a script file. In an object oriented framework like CORBA (Object Management Group) [2], the interface to a software component has to be pre-defined, which becomes sometimes inadequate during a software-development cycle. In addition, due to the considerable amount of knowledge required, it has not successfully been caught up by the real world bioinformatics community.

In contrast, our basic concept is to develop a framework alternative to a keyboard. This led us to a more comprehensive method for a scriptable application. Our method stays in a conventional programming model, where a single thread process waits a command-input from a terminal. In terms of scripting languages, we are going to embed LUA language [6] to our message dispatcher program for finer control of messages using variables, conditionals, and loops.

5.5 An Application to Gene Finding

Our method is a general framework for scripting and networking application programs. In addition to MOSBY, it is currently applied to our development of a genomic sequence viewer for a gene finding system [3]. Since the gene finding system requires different levels of data set and knowledge for a target sequence, a computational server with a single algorithm does not always give the best criteria for scientists. Viewing multiple results together with other related information in a local disk would be important. While server side computing has the limit in this respect, private computing can maximize the ability of programs to support human inference on undocumented knowledge in biology.

6 Conclusion

We have introduced our simple scripting framework for application programs using inter process message implemented in our programming library ASHLEY. Through a message dispatcher tool as a helper application with a MIME type, we were able to control local application programs from a Web browser by selecting links to its script data. The result was successful and proved that this simple method is useful for a wide range of application systems in bioscience with private data processing.

Acknowledgments

We would thank to Dr. Katsutoshi Takahashi, Dr. Yutaka Akiyama for useful discussions, and Dr. Nobuyuki Otsu for continuous support and suggestions.

Appendix: Introduction to ASHLEY API

ASHLEY programming library provides APIs in C language for building an application program with described features. Our command table is implemented as an *order book* mechanism, where a module will *take order* and a caller *gives order* for an specific task.

```
void    YosTakeOrder(module, entry)
void    YosGiveOrder(module, command, parameter)
long    YbsOrderBead(module)
                     char module[],command[],parameter[];
                     void (*entry)();
```

The function `YosTakeOrder()` registers specified function, entry as a command handler for the module. The function `entry()` takes two arguments of character-strings superseding previous function. `YosGiveOrder()` resolves the function for the given module name and call the registered function with two arguments of character-strings. If the handler function is in current scope of the source code, it is equivalent to calling the specified function. But it allows dynamic linking module to install new names and functions. The context of the function is stored in our memory database. `YbsOrderBead()` returns its memory handle which is used in other APIs for our memory database.

```
void    YosOnMessage(name,entry)
int     YisMessage(name, command, parameter)
void    YosMessageReply(status, immediate)
                        char name[];
                        int status,immediate;
```

These are basic functions for ASHLEY inter-client message. The function `YosOnMessage()` declares the name of message listener in the local system overriding existing listener. The message handler function `entry()` takes two arguments of character-strings as well as the command handler function. `YosMessageSend()` may return error status for the absence of the listener or a timeout. It blocks execution until listener complete process of the handler function. Optionally, listener can return a user-defined status by `YosMessageReply()` in the handler function. The status is immediately returned for the sender if `immediate = true`.

References

[1] Abram, G. and Treinish, L., An extended data-flow architecture for data analysis and visualization, *Computer Graphics*, 29(2):17–21, 1995.

[2] Adler, R.M., Emerging standards for component software, *IEEE Computer*, 3:68–77, 1995.

[3] Asai, K., Itou, K., and Ueno, Y., Recognition of human genes by stochastic parsing, *Proceedings of the Pacific Symposium on Biocomputing (Altman et al., eds.)*, World Scientific Press, 228–239, 1998.

[4] Casher, O. and Rzepa, H.S., A chemical collaboratory using explorer EyeChem and the common client interface, *Computer Graphics*, 29:52–54, 1995.

[5] Gong, L., New security architectural directions for Java (extended abstract), *Proceedings of IEEE COMPCON, San Jose*, 97–102, 1997.
`http://java.sun.com/people/gong/papers/ieee-compcon97.ps.gz`

[6] Ierusalimschy, R., de Figueiredo, L.H., and Filho, W.C., LUA–an extensible extension language, *Software: Practice & Experience*, 26(6): 635–652, 1996.

[7] Iversion, E., Flypaper (shareware),
`http://www.zeeland.k12.mi.us/ mcoleman/spanish/flypaper.html`

[8] National Center for Supercomputing Applications, The Common Gateway Interface, 1995.
`http://hoohoo.ncsa.uiuc.edu/webapp`

[9] WebApp. *The Institute for Academic Technology, The Univ of North Carolina at Chapel Hill, Durham.* `http://www.iat.unc.edu/technology/software/webapp/index.html`

[10] Sayle., R.A. and Milner-White, E.J., RasMol: biomolecular graphics for all, *Trends in Biochemical Science*, 20:374–376, 1995.

[11] Thompson, D., NCSA Mosaic Common Client Interface, *NCSA*.
`http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/cci-spec.html`

[12] Ueno, Y. and Asai, K., A new plug-in software architecture applied for a portable molecular structure browser, *Proceedings of Intelligent Systems for Molecular Biology (Gaasterland et al., eds.)*, AAAI Press, 329–332, 1997.

[13] Ueno, Y. and Asai, K., A high-throughput graphics library designed for a portable molecular structure viewer, *Proceedings of the Pacific Symposium on Biocomputing (Altman et al., eds.)*, World Scientific Press, 201–212, 1998.

[14] Vaudreuil, G., MIME: multi-media, multi-lingual extensions for RFC 822 based electronic mail, CNRI, *ConneXions*, 36–39, (September) 1992.

[15] World Wide Web Consortium, Hypertext Markup Language (HTML), 1996.
`http://www.w3.org/pub/WWW/MarkUp`

[16] Zawinski, Z. and Netscape Communications Corp., Remote Control of Unix Netscape, Netscape Support Documentation. `http://home.netscape.com/newsref/std/x-remote.html`