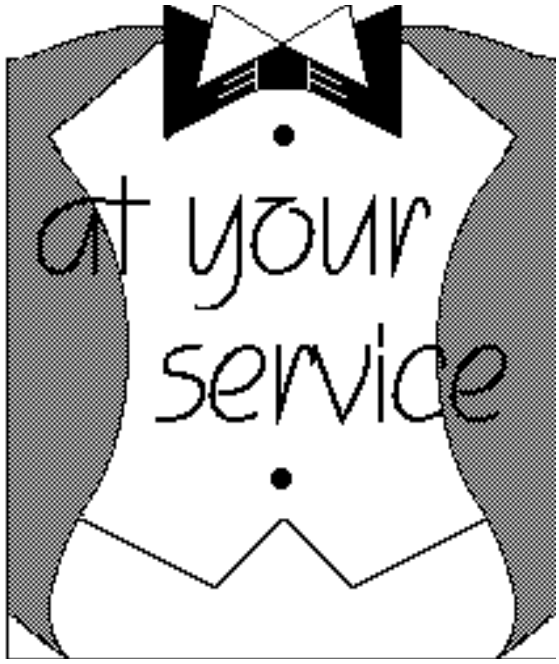



# AYS

At-Your-Service
HyperCard
Communications Package

Version 2.0

1st March 1994

**Contents**

***NB:*** **this manual was originally prepared using Times 12 point font set up for A4 paper.**

# About AYS


## Abstract

*At Your Service* is a HyperCard communications package that provides a series of unique stack based communication facilities.

The package is designed around the concept of a “service”, which is defined as any process running on a local or remote computer, eg. an Online Public Access Catalogue (OPAC) or a Campus Wide Information System (CWIS). A service is uniquely defined via a series of 16 parameters that describe attributes such as the name, service type, the method used to connect to the service, the login and logout scripts, etc.

The AYS Installer stack represents the hub of the package since it is responsible for maintaining service definitions by providing facilities for creation, storage and testing. As the name implies the stack is also responsible for installing/removing definitions into/from other stacks, therefore enabling the creation of custom built communication stacks or the addition of communications facilities to existing stacks.

An installed service definition may be used to either initiate a service connection or ‘listen’ for an incoming service connection. This later 'listening' mode of operation permits the creation of simple HyperCard 'server' facilities.

Service definitions also allow for multiple, simultaneous sessions to be active at any one time. A typical session might therefore include a serial connection to your local OPAC in one window, whilst another window displays information from a CWIS, and a third displays a remote network connection to a second OPAC.

The package utilises the power of HyperCard’s scripting language, HyperTalk to provide a powerful communication scripting facility. It achieves this by defining a series of communication specific language extensions that enable scripts to communicate directly with services.

AYS is based upon Apple’s Communications Toolbox technology, consequently connectivity is only limited by the range of Connection and Terminal Emulation tools that are available on your machine.

Although originally intended as a tool for librarians to connect to remote OPAC’s it can as easily be used for other purposes including production control, development and administration.

Two example stacks are included with the package. Both stacks are stand-alone communications “applications” that are intended both as an example of what can be achieved as well as useful, ready-to-use utilities.


## What AYS can be used for.

AYS can be used within any stack that requires text based, scriptable communication capabilities. In addition to the two example stacks, AYS has been used as the

communications vehicle in at least two production, library application stacks.

The first, uses AYS to automatically connect to, and navigate though a predefined list of library inquiry systems to check for certain categories of newly arrived material. The information retrieved is then used to e-mail a list of interested ‘customers’.

AYS has also been used successfully within an inter-library loans stack. AYS is used within the stack to automatically down-loan loan requests from a remote inter-library loans e-mail system. The loan requests are then parsed, records created and written to a loans database. The stack also facilitates the creation of new, outgoing loan requests that are automatically sent to the remote e-mail system.

## Registering your Copy

AYS is a shareware product, it is not free.

The distributed versions of the AYS Installer and example stacks will only launch communication sessions with a time duration of 5 minutes, after which they will be automatically closed down. To remove these restrictions you are obligated to pay the registration fee. Upon registration, you will be sent a password that may be used to nullify these restrictions.

## Contact

Please direct all communications to: postal: Tim Barlow.
PO Box 607
Sandy Bay, TASMANIA 7005
AUSTRALIA

e-mail: tim.barlow@lib.utas.edu.au

# Installing AYS

## What the AYS Package Includes

The AYS Installer stack.

Three example stacks.

Communications ToolBox tools, including the TGE TCP tool.

Several TeachText "Read Me" documents including an abstract and a "Getting Started" document.

The user manual.

## What you need to use AYS

AYS requires HyperCard version 2.1 or later. In order to use the full range of facilities provided by the AYS installer you will need to be able to set the user level of HyperCard to 5 (scripting). To use the TGE TCP tool you require MacTCP version 1.1 or later to be installed.

## Installing the Communications Tool Box Tools

The installation of AYS involves dropping the included communications tools and fonts onto the system folder.

Open the folder "Comms Tools" in which the tools are stored. Select all of the tools and fonts and drag them and drop over the system folder. System 7.x will then store them in the correct place.

## Increasing the HyperCard Memory Partition

It is recommended that you set the memory partition of HyperCard to a minimum of 1300 kbytes. To do so, firstly choose "Find" from the finder File menu and locate the HyperCard application. Once the application is highlighted, choose "Get Info" from the File menu. Locate the current size box and increase it to at least 1300kbytes (later versions allow for a preferred and minimum memory setting, both of which should be set to at least 1300 kbytes). The default for HyperCard is 1000kbytes so it is likely that this will need to be changed. Increase it again if you get low memory warnings.

# The AYS Installer Stack

## Introduction

The purpose of the Installer stack is to -

- Provide facilities for the creation and modification of service definitions from either manually entered data or via a text file scanning facility.
- Provide facilities for the storage and indexing of service definitions.
- Allow the testing (activation) of service definitions.
- Provide facilities for installing service definitions into target stacks, either individually or in batches.
- Provide facilities for removing existing service definitions from target stacks, either individually or in batches.
- Provide facilities for importing service definitions from other stacks either individually or in batches.
- Provide facilities for the creation of one or more text edit windows.

## Service Definitions

A service definition is a set of parameters that together define the external attributes of a service. Each service is defined by a set of 16 parameters, all of which are held on a single HyperCard card. Typical parameters are, the service name, the service type, the connection tool used to connect to the service, the login and logout scripts, etc.

Service definitions may be used to either initiate a connection to a service or be set into 'listening' mode to await the arrival of a complementary, initiating connection. This later capability permits the creation of simple HyperCard 'server' facilities.

Service definitions fall into one of two major categories, services that have associated terminal emulation windows ("windowed" services) and services that don't ("windowless" services).

A "windowed" service displays all connection data into a terminal emulation window that appears, on the desktop as any other Macintosh window. After the successful launching of the first service, a new menu, the "Windows" menu will appear in the menubar and subsequent service launches will update this menu. The menu provides options for the control of service windows.

A "windowless" service has no associated window nor menu. This type of service is intended to be used by stacks that implement communication front-ends that wish to hide the detail of the connection. Since these services are neither visible to the user nor appear in any menu they may only be written to and read from via HyperTalk scripts.

The installer may install/remove/import service definitions in one of two forms, either as HyperCard 'cards' or as Macintosh resources. The first AYS example stack, the "Personal Networker" is an example of a stack constructed from card definitions. The second AYS example stack, the "Archie Client" is an example of a stack constructed from resource

definitions. The use of resource based definitions is particularly useful for constructing stacks where it is desired to hide as much of the service detail as possible.

## Registering

Each time the installer stack is launched a dialog box is displayed that requests the input of a password; this password is obtained by registering your copy of the AYS package. Until this is done the example stacks, the AYS Installer stack, and any stack derived from it will only launch services for a duration of 5 minutes, after which they are automatically terminated. Once you have registered your copy and entered the password the AYS Installer will beep three times and a further dialog will be displayed. If this does not happen it is likely that the password has been entered incorrectly. Successful entry of the code will remove the time restriction from the installer stack but not from the example stacks or any other stack derived from the installer. To remove the time restrictions from these stacks you must either install a new service, or reinstall an existing service into each stack WHILST THE OPTION KEY IS DEPRESSED. A dialog box is displayed to indicate successful removal of the time restrictions from each stack.

**Navigating in the Installer**

The introductory card of the stack contains a scrolling list of the services available with a line of rectangular buttons beneath them. These buttons are “Import”, “Install”, “Delete”, “Defaults” and “Prefs”. Each of these buttons has an associated card. They give you access to all the major operations performed by the Installer apart from creating a new service. The function of these buttons will be described later under individual headings.

To modify an existing service you click on its name in the scrolling list. This will take you to the definition card for that service.

A new service is created when you select “New Card” from the Edit menu and define the associated parameters such as the Connection and Terminal Settings, and any scripts that might be required.


The stack has a special menu, the “Function” menu that is only highlighted when a service definition card is displayed. The menu provides a series of service definition manipulation facilities such as installing and removing definitions into/from other stacks.

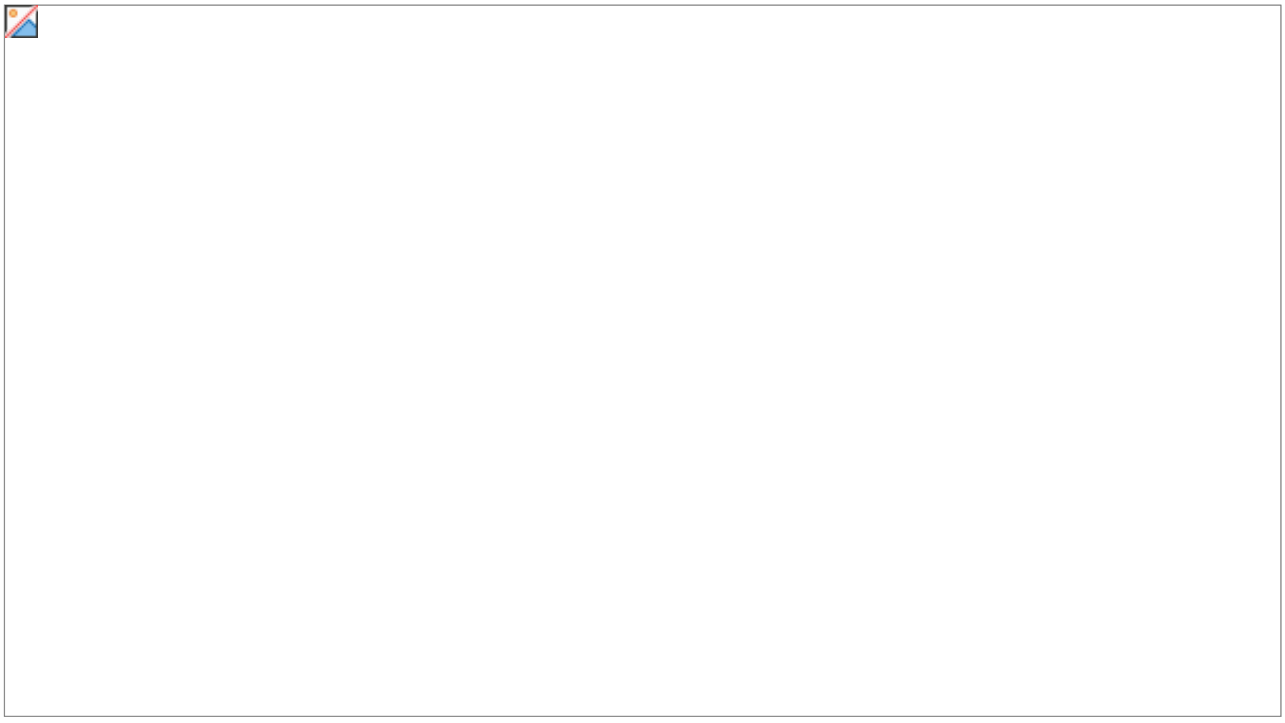
The last option on the menu provides for the creation of a text edit window that may be used to capture, edit and save information from/to service windows. The contents of text edit windows can be saved to text files that may then, for example be imported into word processors at a later date.

**Creating a New Service**

The following step by step guide will enable you to create a service definition. Start by defining a default service, and then create a new definition card and complete the details. Test the new definition and when you are happy with its performance, install it into a target stack.

**Service Definition Parameters**

Each service definition is constructed from a series of parameters that are held on a single card, for example -



The parameters are as follows -

1. The unique name of the service eg. “NewsReader”. This field may be used to sort the cards via the “Preferences” card. The length of the name is restricted to 30 characters and **may not contain numeric characters such as - + * / or #.**
2. The geographic location of the service, this field may be used to sort the cards via the “Preferences” card.
3. The type of service provided, this field may be used to sort the cards via the “Preferences” card.
4. Notes and general information about the service.
5. The charge in cents per minute associated with the service. Setting this value to 0 will turn this feature off.
6. The number of minutes of inactivity before the service will “timeout” and terminate the connection. Setting this value to 0 will turn this feature off.
7. The number of screen pages to be scrolled by the Terminal Emulation tool being used.
8. The local password to protect the service.
9. The communications toolbox Connection tool to be used to connect to the target computer upon which the service is available eg. “TGE TCP Tool”.
   The tools available for selection are those Connection tools that have been

installed on your system. Every time a selection is made the operator is presented with a tool specific interface (dialog box) by which the various communications parameters for the tool may be specified. For instance, the TGE TCP tool dialog allows the network address, terminal type and many other parameters to be specified. Each tool is different.

When the tool parameters have been selected and the dialog box closed the selected parameters will appear as a string of 'tokens' in the scrolling field to the right of the 'Term. Tokens' button.

10. Whether or not any menu associated with Connection tool is to be added to the service menubar.

11. Whether or not the break key is to be activated for the service
12. The Communications toolbox Terminal Emulation tool that will be used to display the output generated from the service (eg. “VT102 Tool”.) or “None” to indicate that the service is a “windlowless” service.
    The tools available for selection are those Terminal Emulation tools that have been installed on your system. Every time a selection is made the operator is presented with a tool specific interface (dialog box) by which the various emulation parameters for the tool may be specified. For instance, the VT102 tool dialog allows the character set, cursor type, local echo and many other parameters to be specified. Each tool is different.
    When the tool parameters have been selected and the dialog box closed the selected parameters will appear as a string of ‘tokens’ in the scrolling field to the right of the ‘Conn. Tokens’ button.
13. Whether or not any menu associated with Terminal Emulation tool is to be added to the service menubar.
14. Up to 5, comma separated “termination” character strings that will, upon detection signal the closure of the service. That is, AYS will check all output from the service against these strings. If a match is found then the stack will terminate the service by executing the logout script and closing the connection, etc. Each string is limited to 255 characters in length and may contain character pairs that together define a control character. That is, the character “^” (shift 6) is interpreted as defining the following character to be a control character. The character is converted to its numeric control value by subtracting 64 from its character value, eg. ^A = control A and ^[ = escape. The matching of strings against the service output is case sensitive ie. the case of each character in each string is significant.
15. Up to 5, comma separated “notify” character strings that will, upon detection trigger the execution of a HyperTalk handler called “StringNotify”. That is, AYS will check all output from the service against these strings. If a match is found then the stack will attempt to execute a HyperTalk handler named “StringNotify” Each service may have it’s own “StringNotify” handler associated with it, see item 16 below. Each string is limited to 255 characters in length and may contain character pairs that together define a control character. That is, the character “^” (shift 6) is interpreted as defining the following character to be a control character. The character is converted to its numeric control value by subtracting 64 from its character value, eg. ^A = control A and ^[ = escape. The matching of strings against the service output is case sensitive ie. the case of each character in each string is significant.
16. The HyperTalk script handlers that are to be associated with the service.

Defining the Default Parameters

Since many of the features of a service will remain constant for your environment you

should firstly set your service defaults to minimise the amount of work required to create new services. On the introductory card click the "Defaults" button to take you to the "Service definition defaults" card.

Setting the default communications tool will depend on the type of network access you have, many users will have tcp/ip access using MacTCP. In this case you should set the TGE TCP tool as the default tool.

The Terminal tool most people will use is the VT102 Tool. If this is the case you should set this as your default terminal tool.

Complete all of the other fields as necessary, the on-line help should provide sufficient information to complete this task.

Create the New Service

Choose "New Card" from the Edit menu. A new service card will be created with the name "New Service" and it will contain the default values previously defined. Fill in the remaining fields such as the name and modify any of the defaults that do not apply to this particular service. The on-line help should provide sufficient information to complete this task.

Select the script window by clicking on the "Service Scripts" button and create any HyperTalk scripts required by the service.

**Scripting**

A service definition may have any number of HyperTalk script handlers, (not functions) associated with it. The names of the associated handlers will appear in the "Scripts" pop-up menu that appears below the service window bar when the mouse is clicked in the bar, and the command key is depressed.

The script handlers are written in HyperTalk, HyperCard's scripting language. AYS extends HyperTalk with several commands, functions and properties to enable it to talk directly to service and text edit windows. These extensions are described in detail in appendix C. Examples of the use of most of these extensions appears in appendix F.

AYS maintains a series of HyperTalk globals that together reflect the status of a stack's active service and text edit windows. You can query these globals within your scripts simply by including a "Global" statement at the head of the script. A full description of all the globals maintained by AYS appears in Appendix D.

To create scripts, click the "Service Scripts" button on the service definition card. This will place you in the HyperCard script editor with which you may start entering scripts. The scripts are stored in the "Card Script Area" of the card. It must be remembered, of the scripts entered, and thus associated with the service, only the names of the handlers will appear in the pop-up "Scripts" menu.

Two special handlers are recognised. The first is called "LoginToService". If a script handler of this name is associated with the service then it will be executed immediately following the successful launching of the service. If the handler fails then the service will be closed down. The second special script handler is called "LogoutFromService". If a script handler of this name is associated with the service then it will be executed immediately prior to closing the service.

Script handlers associated with a service may be executed in a number of ways. Firstly, the special handlers "LoginToService" and "LogoutFromService" are executed automatically by AYS at service start-up and shut-down time. The second way that a service handler may be executed is via the use of the pop-up "Scripts" menu that appears when the mouse is depressed in the service window's drag bar whilst the command key is depressed. Selecting a handler name from the menu will result in the execution of that handler. The remaining methods of script execution depend on the method used to install the service into the target stack.

If the service is installed as a HyperCard 'card' then a service script may also be executed in the normal fashion by either calling it from another script or from the message box. A service script handler may also be executed via the use of the special function "ExecuteServiceScript", which is one of the AYS HyperTalk extensions described in appendix C. The function may be called from any HyperTalk script

(either handler of function) or from the message box and takes as one of its arguments the name of the service handler to be executed.

If the service is installed in a target stack as a resource then a service handler may ONLY be executed from another script (or message box) via the use of the "ExecuteServiceScript", function (the reasons for this are given in section "Technical Considerations").

NB. During the execution of a script, AYS ignores the occurence of "notification" strings but will react to the presence of "termination" strings, (resulting in the failure of the script).

**Testing**

The whole process may now be tested by choosing either the "Open Service" or "Listen for Service" item from the "Function" menu. The action taken depends on the menu item chosen.

If the "Open Service" item is chosen then an attempt is made to connect to the required service. If the connection attempt is successful, and there is a script handler called "LoginToService" associated with the service, then this will be executed. This script may be aborted by typing command-period thus allowing for the debugging of login scripts, (NB. this abort option is not available once the service has been installed into a target stack). If the service is a "windowed" service then following the successful launching of the first service, a new menu, the "Windows" menu will appear in the menubar. This menu provides options for the manipulation and management of the new window. Once connected you can move to another service card and connect to that service.

If the "Listen for Service" item is chosen then the service is placed in 'listen' mode, (dependant upon the configured Connection tool). The service remains in this mode until a connection attempt is made by a complementary process. If the attempt succeeds that the 'listening' service becomes a fully operation service, otherwise an error message is displayed. Regardless of the outcome, the stack is informed of the event and will attempt to execute a HyperTalk handler named "ListenNotify" that may or may not be associated with the service.

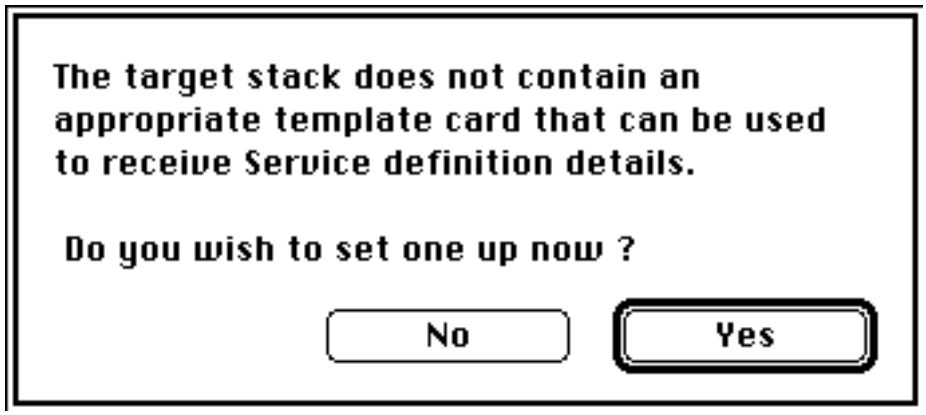
If the service is a "windowed" service then it may be closed by selecting the "Close Window" option from the "Windows" menu or by clicking in the "Go Away" box on the service window. If the service is a "windlowless" service then it may only be closed by executing the "CloseService" script from the message box, see appendix C. When the service is closed, if there is a script handler called "LogoutFromService" associated with the service, then this will be executed first before the connection is closed. Like the login script, this script may be aborted by typing command-period. This allows for the debugging of logout scripts, (NB. this abort option is not available once the service has been installed into a target stack). If this is the last "windowed" service to be closed then the "Windows" menu will be removed from the menubar.

**Installing**

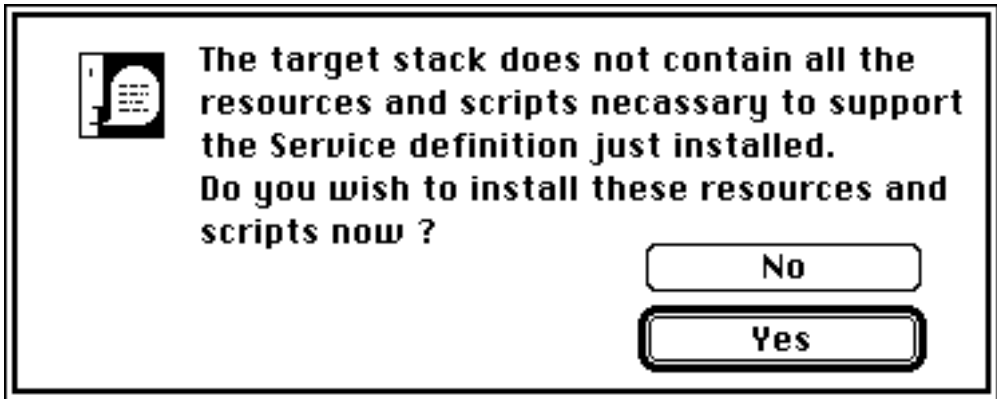
Once the service is tested the next step is to install it into a target stack. You must firstly define the stack into which you wish to place this new service. Chose "Define Target" from the "Function" menu and select the stack into which the service is to be installed. You will be asked if you wish to install service definitions as HyperCard cards or as Macintosh resources. Select whichever is appropriate to the type of stack that you are constructing.

Once the target has been defined you install it into the stack by choosing the "Install into Target" option from the "Function" menu. The first time that you install a service you will be prompted by a further one or two dialog boxes.

If you are installing service card definitions and the stack into which you are installing does not contain a card of the type required by AYS then the following dialog is shown. If you click the "No" button the process will be aborted.


The target stack does not contain an appropriate template card that can be used to receive Service definition details.
Do you wish to set one up now ?
No
Yes

Regardless of the type of service definition being installed the following dialog will be displayed. The dialog box will be displayed each time a service is installed into the stack until the "Yes" button is clicked. For details of the resources and scripts that are copied to the target stack see the section "Technical Considerations" below.




**Aids to Defining a Service**

To assist in the creating and archiving of service definitions the Installer stack provides the following two features.

Archiving a Service Definition to a Text File

A service definition may be archived to a disc file via the use of the "Create Parameter File" option in the "Function" menu. This option will encode the service definition parameters of the current card and write them to file in a format that is understood by the following option.

Creating a Service Definition from a Text File

To assist in the creation of service definitions the "Load from Text File" option of the "Function" menu may be used to scan any text file for a definable list of keywords that may describe AYS service parameters.
When you select this menu option the first action is to replace the Connection and Terminal names and configuration parameters (tokens) of the "New Service" card with those defined on the "Keyword Default" card. This is necessary to ensure that a known set of parameters are present in the "New Service" definition prior to scanning the text file. The text file is then scanned by interpreting the keyword table contained on the "Keyword definition" card. The rules for scanning files for keywords is described in appendix G.

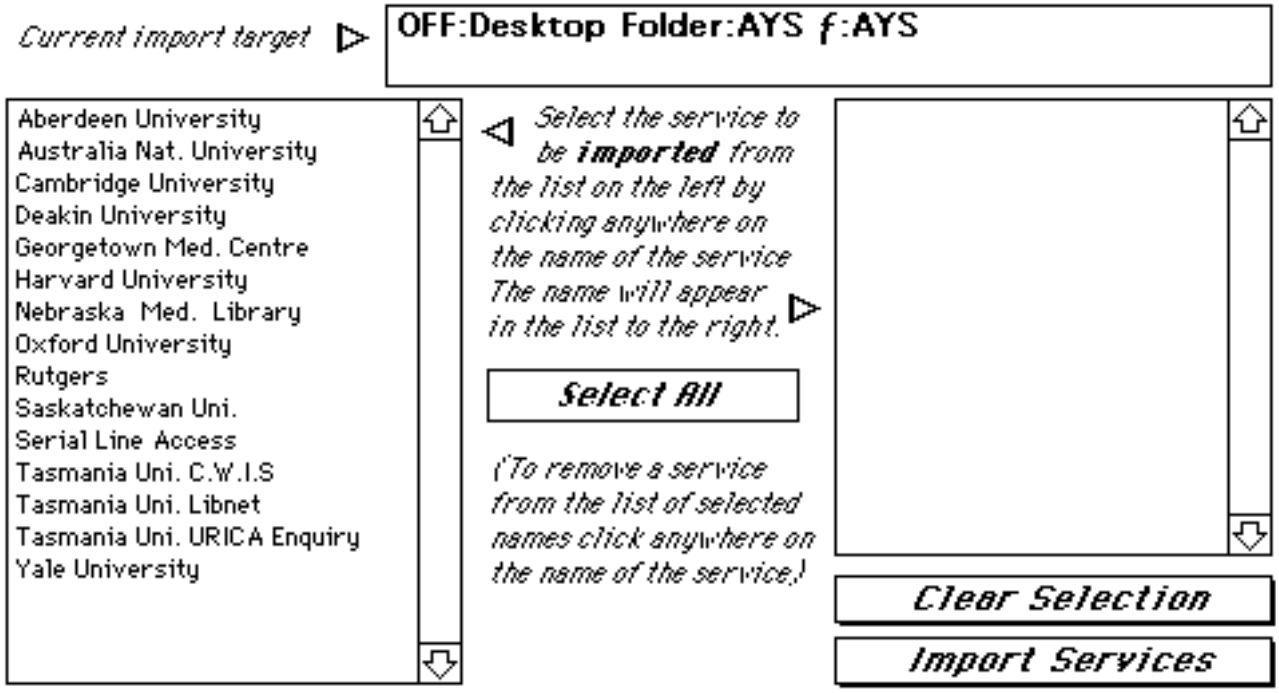
The intension of this feature is two fold. Firstly it allows the creation of service definitions from files created by the previous feature ie. the reinstalling of

previously archived definitions. Secondly, it allows the creation of definitions from text files such as Hytelnet files or Telnet configuration files.

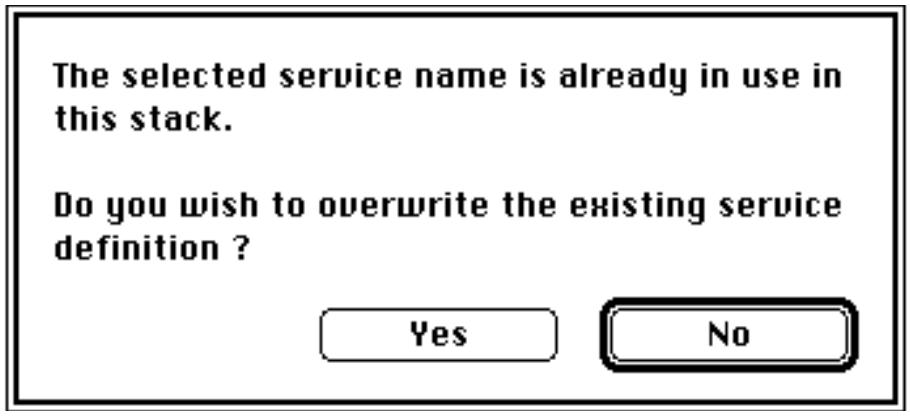
Import

You may import service definitions from any AYS compatible stack. If you click the "Import" button a standard open file dialog box will enable you to choose a stack from which to import service definitions.
Once a stack has been chosen you will be asked if you wish to import service definition cards or resources. If the selected stack does not contain any of the chosen type of definition then an error message will be displayed.

The following card will then be shown with the service definitions that are available for importation appearing into the left hand field.




These may be chosen singularly by clicking on each service name that you wish to add to the Installer, or all at once by clicking the "Select All" button. Click the "Import Services" button at the bottom of the card, and the selected definitions will be added to the installer. If the installer stack already contains a service with the same name as one of the services being imported then the following warning message is output.



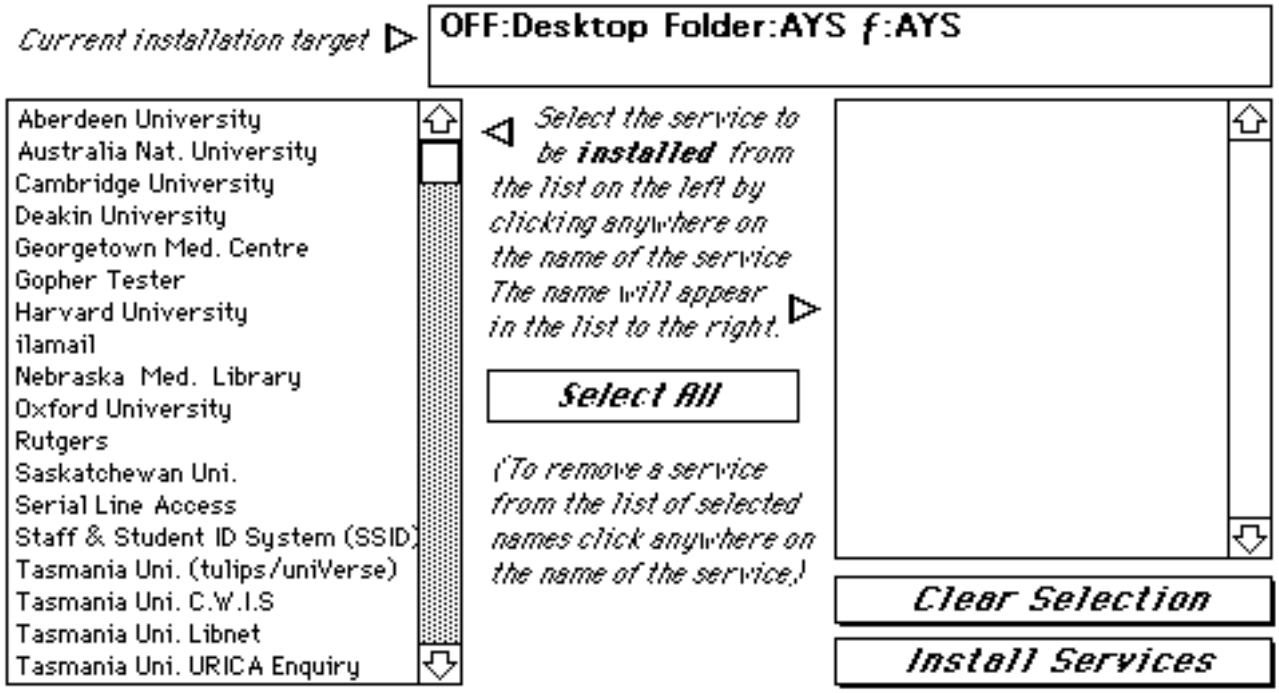

An alternative to this "batch" importation feature is to import a single service at a time, This may be achieved by going to the card associated with the service to be imported, choosing "Define Target" to set the stack from which you wish to import/recover and then choosing the "Function" menu option "Import from Target".

Install

You may install service definitions into any stack. If you click the “Install” button a standard open file dialog box will enable you to choose a stack into which to install service definitions.

Once a stack has been chosen you will be asked if you wish to install service definition cards or resources.

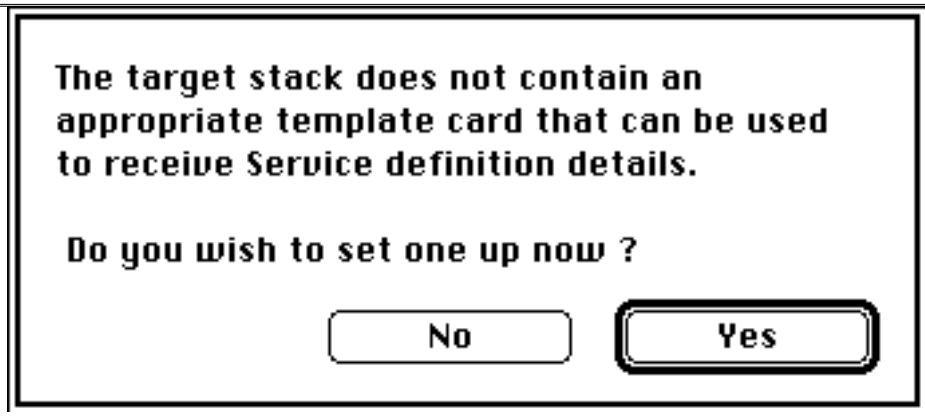
The following card will then be shown with the service definitions that are available for installation appearing in the left hand field.



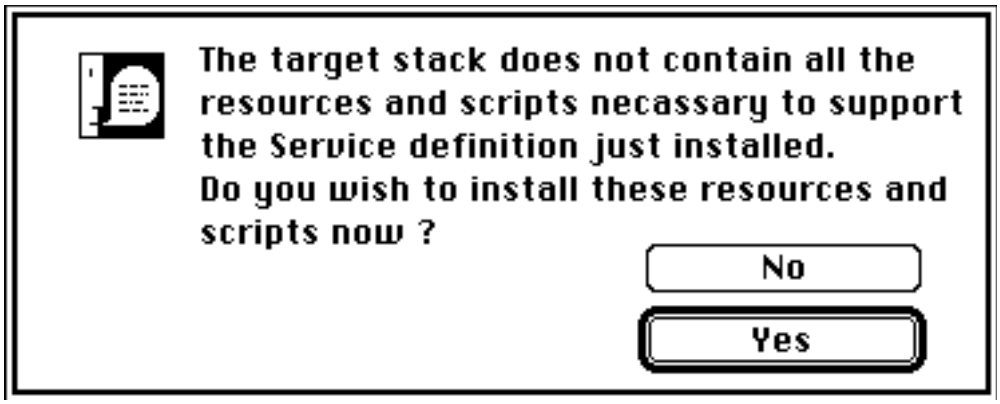

These may be chosen singularly by clicking on each service name that you wish to install, or all at once by clicking the "Select All" button. Click the "Install Services" button at the bottom of the card, and the selected definitions will be installed into the target stack. If the stack being installed into already contains a service with the same name then the following warning message is output.


The first time that you install a service you will be prompted by a further one or two dialog boxes.

If you are installing service card definitions and the stack into which you are installing does not contain a card of the type required by AYS then the following dialog is shown. If you click the "No" button the process will be aborted.



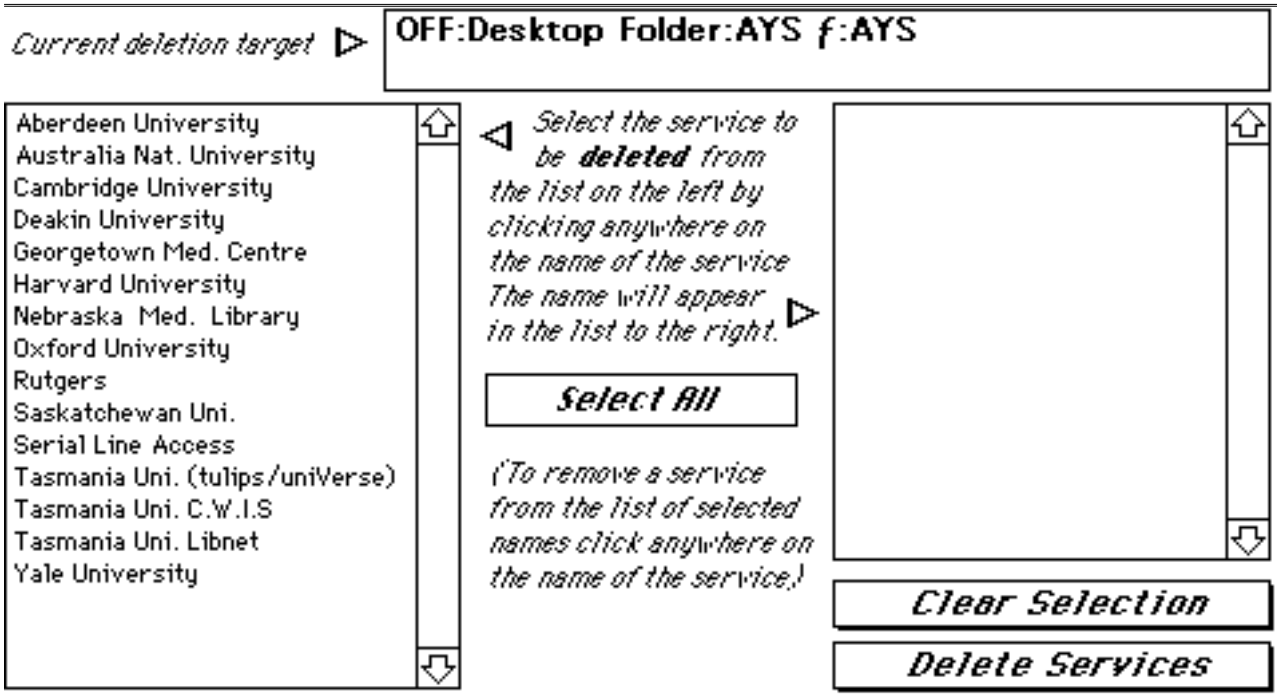

Regardless of the type of service definition being installed the following dialog will be displayed. The dialog box will be displayed each time a service is installed into the stack until the "Yes" button is clicked. For details of the resources and scripts that are copied to the target stack see the section "Technical Considerations" below.




An alternative to this "batch" importation feature is to install a single service at a time, This may be achieved by going to the card associated with that service, choosing "Define Target" to set the stack into which you wish to install the service and then choosing the "Function" menu option "Install into Target".

Delete

You may delete service definitions from any stack. If you click the "Delete" button a standard open file dialog box will enable you to choose a stack from which to delete service definitions.
Once a stack has been chosen you will be asked if you wish to delete service definition cards or resources.
The following card will then be shown with the service definitions that are available for deletion appearing into the left hand field.




These may be chosen singularly by clicking on each service name that you wish to delete from the target stack, or all at once by clicking the "Select All" button. Click the "Delete Services" button at the bottom of the card, and the selected definitions will be deleted from the selected stack. If the stack being deleted from does not contain the named service then the following warning message is output.


An alternative to this "batch" deletion feature is to delete a single service at a time, This may be achieved by going to the card associated with that service, choosing "Define Target" to set the stack from which you wish to delete the service and then choosing the "Function" menu option "Delete from Target".

## Defaults

Clicking on the "Defaults" button will take you to the first of the two default cards.
The first default card, the "Service definition default card" enables you to define default values for creating new services definitions (see the section "Defining the Default

Parameters").

The second default card, the "Keyword default card" allows you to define the default connection and terminal emulation tools that will be loaded into the "New Service" card immediately prior to scanning the selected text file for keywords, (see the section "Creating a Service Definition from a Text File").

Preferences

Clicking on the “Prefs” button will take you to the “Preferences” card that allows for the setting of four options.

The first and second options allow for the setting of the default font and point size for text edit windows.

The third option allows you to change the sort order of the service definition cards in the stack. If you click in one of the service sort boxes then the service definition cards will be sorted appropriately.

Each service window has an associated series of settings, accessed via the “Settings” sub-menu of the “Windows” menu. Clicking the fourth option will allow you specify whether or not these settings are to be “remembered” between sessions.

## Menus added by a Service

### Windows Menu

When the first "windowed" service or text edit window is opened a new menu, the "Windows" menu is added to the menubar. When additional windows are opened this menu is expanded to reflect the new situation. When the last service or text edit window is closed the "Windows" menu is removed from the menubar.

The menu provides a comprehensive set of options for the manipulation and management of service and text edit windows. Below is a complete description of each feature under its menu option name.

***Window Names***

The first options in the "Windows" menu contains the names of the windows that have been opened. The currently active window is denoted by a leading "•" character. Windows that have been hidden are denoted by a leading "◊" character. A visible window may be made the currently active window by a) Clicking anywhere in that window b) Selecting its name from this menu or c) Using one of the special keys for changing windows as described in the "Use Key Pad" option in the "Settings" submenu.

***Show Window***

This option will only be active if one or more windows have been hidden, and will display and activate the most recently hidden window.

***Hide Window***

This option will hide (render invisible) the currently active window.

***Close Window***

Closes the currently active window in the appropriate manner. If the option is selected whilst the option key is depressed then all active windows are closed.

***Clear Window***

If the currently active window is a text edit window then its entire contents are cleared. If the currently active window is a service window then only the terminal emulation screen is cleared, the cached (scrolled) area is not affected. If the option key is depressed when the option is selected then the cached area will also be cleared.

***Break Window***

This option is only highlighted for a service window, the option will send a long break (1 second) signal to the service. If the option key is depressed when the option is selected then a short break (1/4 second) signal is sent to the service.

*Settings Submenu*

*Copy Submenu*

*Search Submenu*

See below.

***Tile Windows***

Tiles all active, visible windows, excluding the card window.

***Stack Windows***

Stacks all active, visible windows, excluding the card window.

***Service Info***

This option is only highlighted for service windows. Selecting the option will display statistical information relating to the service and allow the specification and calculation of a connection charge per minute.

**Settings Submenu**

This submenu contains 7 options; the first 5 are associated with service windows, the last 3 with text edit windows

***Data Logging***

This option allows the logging of all output from a service to a text file. When the option is selected the operator is prompted for a file name. The menu option is preceded by a "tick" mark to indicate that the option is "on". All data received by the service window will be written to the log file, (for further details concerning the manner in which data is written to the logging file see Appendix C - Service Properties).

***Smooth Scrolling***

This option allows the choice between smoother-slower or faster-jerkier terminal emulation text display. If the option is active then the menu item is preceded by a "tick" mark.

***Delete = Backspace***

This option allows the character output from the "Delete" key to be set to either a "backspace" character (character 8) or to a "delete" character (character 127) for a service window. NB. This key setting may be overridden by a similar setting in the terminal emulation tool that you are using. If the option is active then the menu item is preceded by a "tick" mark.

***Slow Window Paste***

This option allows the slowing down of all pasted data (either Clipboard data or Strings menu data) to a service window to approximately 6 characters per second. The option is provided for those services that cannot accept rapid streams of input data (ie. do not have input buffers). If the option is active then the menu item is preceded by a "tick" mark.

***Use Key Pad***

This option allows for the mapping of certain window functions to the keys of the numeric key pad. If the option is active then the menu item is preceded by a "tick"

mark. The mappings are described by the table below :

| Function | NumericKeyPad | |
|---|---|---|
| Activate String no 1 -> 11 | 0 through 9 and . | (only service windows) |
| Clear window | Clear | |
| Change window Right | + | |
| Change Window Left | - | |
| Zoom window | * | |
| Copy & Paste | = | (only service windows) |
| Scroll to cursor | / | |

***Save Styling Data***

If this option is active (ie. preceeded by a "tick" mark), then each time that the currently active text window is saved to a file the text styling information will also be written to the same file. The file may subsequently be read back by AYS and all the text styling will be preserved. If the option is deactivated then only the text is written to file, and if the file is subsequently read back into At-Your-Service any styling will be lost. NB. This does not affect the files "readability" by other applications.

### *Tab Settings*

Selecting this option results in the display of a dialog box that allows the tab character to be replaced by a number of space characters. If the option is active then the menu item is preceded by a "tick" mark. The facility may be turned off by reselecting the option. If a different number of spaces is required then selecting the active option with the option key depressed will redisplay the dialog box.

## Copy Submenu

### *Print Selection/ Print Text*

This item may assume one of two values depending on the type of window that is currently active. If the currently active window is a service window then the item will appear as "Print Selection". The option will only be highlighted if there is a selection in the window and choosing it will cause the selection to be printed. If the active window is a text edit window then the option will appear as "Print Text". The option will only be highlighted if the window contains text and choosing it will cause the entire contents of the window to be printed.

### *Save Selection/ Save Text*

This item may assume one of two values depending on the type of window that is currently active. If the currently active window is a service window then the item will appear as "Save Selection". The option will only be highlighted if there is a selection in the active service window and it will cause the selection to be saved to a text file via the standard file selection dialog. If the active window is a text edit window then the option will appear as "Save Text". The option will cause the entire contents of the window to be saved to a text file via the standard file selection dialog. Styling will be saved if the "Save Styling Data" option is set. If the currently active text window has already been saved to file then a second copy may be made by depressing the option key whilst selecting the menu item. This will result in the redisplay of the standard file selection dialog from which another file name may be specified, the window will assume the new file name.

### *Copy Items*

This will copy a selection to another window. If there is a selection in the

currently active window and there is at least one other active window (service or text edit) then item 4 onwards of this submenu will contain the names of all the other windows that can receive the selected text. Selecting a service window name from the submenu will result in the selection being pasted into that window. Selecting a text edit window name from the submenu will result in the selection being appended into that window.

**Search Submenu**

***Select All***

If the currently active window is a service window then this item will cause the current terminal emulation screen to be selected, the cached (scrolled) area is not affected. If the active window is a text edit window then the option will only be highlighted if the text edit window contains data and it will cause the entire contents of the window to be selected.

***Find***

If the currently active window is a service window then this item will conduct a find operation on the data that is scrolled off the top of the terminal emulation. The data is searched in reverse order, that is the most recently scrolled data is searched first. If the window is a text edit window then the item provides a standard implementation for text location.

### *Find Again*

Repeats the find operation.

### *Replace, Replace & Find, Replace All*

If the currently active window is a service window then these items will be disabled since their action is inappropriate. If the window is a text edit window then these items provide standard implementations for text replacing.

## Other Menus

### Strings Pop-up Menu

This menu provides for function key emulation by allowing up to 11 strings to be stored and recalled for each active service window. The menu will appear immediately below the window bar of the active service window if the mouse is clicked in the window bar whilst the "option" key is depressed. The menu comprises an item for each string that has been "installed" plus a final item, "*Add String to Menu*" that allows further strings to be installed. Each string is limited to 255 characters in length and may contain character pairs that together define a control character. That is, the character "^" (shift 6) is interpreted as defining the following character to be a control character. The character is converted to its numeric control value by subtracting 64 from its character value, eg. ^A = control A and ^[ = escape. Strings may be modified or deleted by selecting the required item whilst the command key is depressed. The strings may be mapped to one of two sets of keyboard keys, the extended keyboard "Function keys" or the standard keyboard "Numeric Key Pad" keys depending on the setting of the "Use Key Pad" option in the "Settings" submenu. Selecting a string item from the menu results in the contents of the associated string being sent to the service.

### Scripts Pop-up Menu

A pop-up menu of script handlers may be associated with each active service window. The menu (if it has any items) will appear immediately below the window bar of the active service window, if the mouse is clicked in the window bar whilst the "command" key is depressed. If the service window does not have any associated handlers then attempting to display the menu will result in a single "beep". The menu comprises an item for each script handler that may be executed for the service. Selecting an item will result in the execution of that script.

### Tool Menus

The menu bar for a service window may contain up to 2 additional menus. These menus are associated with the Communications Toolbox tools used to connect to a particular service. Whether or not the additional menus will be present will depend on the tools used, as some do not have menus. The display of these menus may be inhibited when defining a service.

## Technical Considerations

### Scripting Areas

When creating new communications stacks that use service definition resources, care must be taken not to place scripts in the card script area of any card. This area is reserved by AYS for scripts associated with services and any script placed in the area may be overwritten by the service scripts. The reason for this, is that the scripts associated with the service are held along with the other service details in the resource. When one of the associated script handlers is executed, (either by use of the pop-up 'Scripts' menu or via the use of the 'ExecuteServiceScript' function) all the scripts associated with the service are firstly copied to the card script area of the current card. It is from here that the requested handler is executed.

### Stack StartUp

The HyperTalk scripts that are installed into target stacks to support AYS communication services, contain the Communications Toolbox initialisation code in the "OpenStack" handler. Consequently, any attempt to launch a service before the "Openstack" handler is executed will result in disaster.

### Installation Process

Whenever a service definition is installed into a target stack the installation process checks to see if the stack contains the full set of supporting resources, and scripts that are necessary to interpret the service definitions being installed. If there isn't then you are informed and asked if the stack is to be updated. If the reply is "Yes" then the stack is updated with the appropriate HyperCard scripts and resources.

(NB. If you wish to replace all of the supporting resources in the target stack with a fresh set from the installer then install a new service or reinstall an existing service into the target stack with the command key depressed.)

The supporting resources that are copied to a target stack at installation time include the code segments (XFCN and XCMDs) necessary to interpret the service definitions. A slightly different set are copied to stacks whose definitions are held as cards as to those whose definitions are held as resources. The definition of each set of code resources are held in appendices A and B.

Although it is possible to install both "resource" and "card" service definitions into the same stack, the set of supporting HyperTalk scripts will only support "resource" or "card" definitions. If a custom stack is required that contains mixed service definition types then the supporting HyperTalk scripts must be modified appropriately.

There are two sets of HyperTalk script handlers that are copied to the target stacks script area by the installation process. The first set contain "essential" handlers and functions that are necessary to carry out such operations as launching and terminating the service definitions. The deletion of one or more of these scripts will render the target stack inoperable.

The second set of scripts that are copied are "optional" since they are not essential to the operation of the stack. These scripts are responsible for creating a set of card buttons whose names reflect the installed service definitions and when pressed will launch the named service. The scripts are activated by typing the command "AYS" into the message box. The script will attempt to set the host stack to level 4 (authoring level). If the host stack cannot be set to this level of operation these scripts will fail. These scripts may be deleted from the target stack with no resulting ill effect.

## Resources

The installer stack uses Pop-Up menus to define a service definition's "Location" and "Type" parameters. These menus may be modified or extended, by the addition of further sub-menus, by using ResEdit to edit the stack's MENU resources.

Several of the error messages generated by AYS contain the word "AYS" or refer to AYS specifically. Should you be using AYS as a toolkit to build your own stacks then you may wish to change the wording of some of these error messages. This may be done by using ResEdit to edit the STR# resources held in the installer stack.
Likewise you may wish to change some of the DLOG, ALRT or DITL resources in order to customise your dialog boxes.

With the exception of the "vers" resource **DO NOT CHANGE THE NAME OR ID's** of any of the resources. Doing so will almost certainly render the AYS stacks and any stacks derived from them inoperable.

## HyperTalk Global Variables

It is advisable NOT to use AYS HyperTalk variables as arguments to HyperTalk script handlers or functions, for instance -

```
on ListenNotify
 Global activeService, NotifyString

 if NotifyString = "Listen Accepted" then
   set the visible of window activeService to true
   if the optionkey is down then
      get ListenService(activeService, true, true)
   end if
 else
   if NotifyString = "Listen Failed" then
      CloseService activeService, false
   end if
 end if
end ListenNotify
```

... may cause problems that are eliminated by the following rewrite -

```
on ListenNotify
 Global activeService, NotifyString

 put activeService into service
 if NotifyString = "Listen Accepted" then
   set the visible of window activeService to true
   if the optionkey is down then
      get ListenService(service, true, true)
   end if
 else
   if NotifyString = "Listen Failed" then
      CloseService service, false
   end if
 end if
end ListenNotify
```

## Warnings

a) If AYS has one or more open service or text edit windows in one or more active stacks when HyperCard is terminated via the "Quit HyperCard" option, (or command Q) then -

   1) The service windows may not be closed correctly. That is, if a service has an associated logout script then it will not be executed prior to the window being closed.
   2) If the text edit windows associated with any "background" stacks contain unsaved text then this will be lost on closure.

   This problem is unavoidable due to the manner in which HyperCard handles external windows at termination time. It is therefore recommended that all windows are closed "manually" prior to quitting HyperCard.

# Appendices

## A. XFCN and XCMDs used in "Card" Stacks.

### a) InitialiseCWindows

Code Type XFCN.

Function Initialises the Communications toolbox and AYS globals.

Arguments None.

Returns A value of 0 to indicate success or -1 for failure

### b) ServiceWindow

Code Type XFCN.

Function Creates a new HyperCard "document" window for the use of a new service, it also provides facilities necessary for the maintenance and control of that window. The HyperCard global value "WindowName" is set to the name of the newly created window.

Arguments None.

Returns "true" or "false" to indicate success or otherwise.

### c) DoCService

Code Type XFCN.

Function Performs one of two operations on a service.

a) OPEN - Opens the named service.

b) CLOSE - Terminates the named service.

c) LISTEN Initiates a listen on the named service

All information necessary for the required operation is obtained from the HyperCard "Card" of the same name as the service being operated on. The HyperCard global value "WindowName" is set to the name of the newly opened/listening window.

Arguments a) The operation required, either "OPEN", "CLOSE" or "LISTEN"

b) if closing a service then

i) the name of the service to be closed.

ii) a "true" or "false" value that specifies whether the progress of the operation should be monitored via a "progress" dialog box.

if opening a service then

i) the name of the service to be opened.

ii) a “true” or “false” value that specifies whether the “Windows” menu “Settings” sub-menu flags should be reinstated from the last session.

ii) a “true” or “false” value that specifies whether the progress of the operation should be monitored via a “progress” dialog box.

Returns “true” or “false” to indicate success or otherwise.

d) **EditWindow**

Code Type XFCN.

Function — Creates a new HyperCard "document" window for the use of a new edit process, it also provides facilities necessary for the maintenance and control of that window. The HyperCard global value "WindowName" is set to the name of the newly created window.

Arguments — None.

Returns — "true" or "false" to indicate success or otherwise.

e) **DoEdit**

Code Type — XFCN.

Function — Performs one of two operations on an text edit window.

a) OPEN - Opens the named editor.

b) CLOSE - Terminates the named editor.

The HyperCard global value "WindowName" is set to the name of the newly opened window.

Arguments — a) The operation required, either "OPEN" or "CLOSE"

b) if closing a text edit window then

the name of the window to be closed.

if opening a text edit window then

i) the name of the text editor to be opened.

ii) an optional value that specifies the default font to be used. If the value is not supplied "Monaco" is assumed.

iii) an optional value that specifies the default font size to be used. If the value is not supplied 9 is assumed.

Returns — "true" or "false" to indicate success or otherwise.

f) **SuspendWindows**

Code Type — XCMD.

Function — Suspends all active windows belonging to the stack by deactivating and hiding them. Only called upon receipt of a "SuspendStack" message.

Arguments — None.

Returns — No codes returned.

g) **ResumeWindows**

Code Type — XCMD.

Function — Resumes all active windows belonging to the stack by reactivating and showing them, if appropriate. Only called upon receipt of a

| | |
|---|---|
| | "ResumeStack" message. |
| Arguments | None. |
| Returns | No codes returned. |

h) **ExecuteServiceScript**

See appendix C

i) **SendToService**

See appendix C

j) **ExpectFromService**

See appendix C

## B. XFCN and XCMDs used in "Resource" Stacks.

### a) InitialiseRWindows

Code Type XFCN.

Function Initialises the Communications toolbox and AYS globals and builds a comma separated list of available service names (ie. "srvc" resources).

Arguments None.

Returns A comma separated list of available service names or an error code = -1

### b) ServiceWindow

Code Type XFCN.

Function Creates a new HyperCard "document" window for the use of a new service, it also provides facilities necessary for the maintenance and control of that window. The HyperCard global value "WindowName" is set to the name of the newly created window.

Arguments None.

Returns "true" or "false" to indicate success or otherwise.

### c) DoRService

Code Type XFCN.

Function Performs one of three operations on a service.

a) OPEN Opens the named service.

b) CLOSE Terminates the named service.

c) LISTEN Initiates a listen on the named service

All information necessary for the required operation is obtained from the "srvc" resource of the same name as the service being operated on. The HyperCard global value "WindowName" is set to the name of the newly opened/listening window.

Arguments a) The operation required, either "OPEN", "CLOSE" or "LISTEN"

b) if closing a service then

i) the name of the service to be closed.

ii) a "true" or "false" value that specifies whether the progress of the operation should be monitored via a "progress" dialog box.

if opening a service then

i) the name of the service to be opened.

ii) a “true” or “false” value that specifies whether the “Windows” menu “Settings” sub-menu flags should be reinstated from the last session.

ii) a “true” or “false” value that specifies whether the progress of the operation should be monitored via a “progress” dialog box.

Returns “true” or “false” to indicate success or otherwise.

d) **EditWindow**

Code Type XFCN.

Function — Creates a new HyperCard "document" window for the use of a new edit process, it also provides facilities necessary for the maintenance and control of that window. The HyperCard global value "WindowName" is set to the name of the newly created window.

Arguments — None.

Returns — "true" or "false" to indicate success or otherwise.

e) **DoEdit**

Code Type — XFCN.

Function — Performs one of two operations on an text edit window.

a) OPEN - Opens the named editor.

b) CLOSE - Terminates the named editor.

The HyperCard global value "WindowName" is set to the name of the newly opened window.

Arguments — a) The operation required, either "OPEN" or "CLOSE"

b) if closing a text edit window then

the name of the window to be closed.

if opening a text edit window then

i) the name of the text editor to be opened.

ii) an optional value that specifies the default font to be used. If the value is not supplied "Monaco" is assumed.

iii) an optional value that specifies the default font size to be used. If the value is not supplied 9 is assumed.

Returns — "true" or "false" to indicate success or otherwise.

f) **SuspendWindows**

Code Type — XCMD.

Function — Suspends all active windows belonging to the stack by deactivating and hiding them. Only called upon receipt of a "SuspendStack" message.

Arguments — None.

Returns — No codes returned.

g) **ResumeWindows**

Code Type — XCMD.

Function — Resumes all active windows belonging to the stack by reactivating and showing them, if appropriate. Only called upon receipt of a

| | |
|---|---|
| | "ResumeStack" message. |
| Arguments | None. |
| Returns | No codes returned. |

h) **ExecuteServiceScript**

See appendix C

i) **SendToService**

See appendix C

j) **ExpectFromService**

See appendix C

## C. HyperTalk Extensions

Several extensions have been provided to enable HyperTalk scripts to communicate with AYS services, these language extensions fall into 4 categories -

a) XFCN extensions.
b) Service properties.
c) Service messages.
d) Useful HyperTalk scripts.

Many of the AYS extensions set and/or get the data from a series of AYS specific HyperTalk global variables, see appendix D. The use of these globals together with the language extensions allow almost any action to be taken on a service.

### 1) XFCN Extensions

#### a) **ExecuteServiceScript**(Arg1, Arg2, Arg3, Arg4,)

Allows the execution of any HyperTalk handler, either associated with a service or not. The function CANNOT be used to execute functions. Using "ExecuteServiceScript" to execute service handlers is, in most cases preferable and in some cases absolutely necessary. For instance, in a stack that contains "resource" services it is the only way to run handlers associated with a service other than using the pop-up "Scripts" menu.

Upon termination, the function returns either a 'true' or 'false' result.

A 'true' result indicates that the requested handler executed successfully. The executed handler signals that this is so by setting "the Result" to 'true'.

A 'false' result' indicates that the requested handler did not execute successfully. The executed handler signals that this is so by setting "the Result" to 'false'. Under these circumstances the HyperTalk global variable 'ScriptStatus' will contain a value that indicates the reason why the handler failed, see appendix D.

| | |
|---|---|
| Arg1 | Name of service for which the handler is to be executed. |
| Arg2 | Name of handler to be executed. |
| Arg3 | Where the handler message is to be sent |
| | 1 = The handler is sent to the card which has the same name as Arg1 (ie. the name of the service). |

2 = The handler is sent to the background of the current card.

3 = The handler is executed in the normal fashion and is subject to the normal message sending order.

Arg4 A “true” or “false” value that specifies whether execution of the script should be “announced” via a dialog box appearing in the top-right corner of the service window.

Returns a) true - if, following execution of the handler “the Result” contains true.

b) false - if, following execution of the handler “the Result” contains “false”.

b) **ExpectFromService**(Arg1, Arg2, Arg3, Arg4  Arg5 ... Arg9)

Allows the output from a service to be scanned for the presence of a number of text strings. Text strings may be defined in one of two ways.

The first way is to define a single string that is to be matched against the service output eg. “connected”. The entire contents of the string are used to find an exact match. If a match is found then the routine will return the index value of the found string, (see below).

The second method is to define a pair of related text strings, separated by the “Δ” (option j) character eg. “!!!Δ.” . The routine will scan the service output for the first occurrence of the first character string. If it is found, then the function will accumulate it and all the text up until and including the first occurrence of the second character string into the HyperTalk global variable “ControlData” and then return the index value of the found string, (see below), eg.

```
 if ExpectFromService(ActiveService, 20, true, "Command=Δ^r") then
if FoundString = 1 then
```

This function call will effectively extract all lines starting with the string “Command=“ and terminating with a carriage return from the service output.

If the second character string is not found within 1000 characters of the first OR another of the defined text strings is found first OR the function fails due to timing-out then the global variable “ControlData” is discarded. The intension of this type of text matching is to allow for the embedding of control data/error strings within service output.

The text strings to be used for searching, regardless of the manner in which they are defined, may contain character pairs that together define a control character. That is, the character “^” (shift 6) is interpreted as defining the following character to be a control character. The character is converted to its numeric control value by subtracting 64 from its character value, eg. ^A = control A and ^[ = escape.

The text strings to be used for searching, regardless of the manner in which they are defined, may comprise one or more wildcard characters . A wildcard character is defined by the character “•”, (option 8) and will cause any character from the service output to be matched against it. For example, the match string “••tch” will be matched against words “patch”, “fetch” etc.

Text strings may be matched against the service output in either case sensitive or case insensitive mode. The mode of text matching is dictated by the fourth argument. If the argument is set to “true” then the mode is set to case insensitive matching and the text string “search” would be matched against  both the strings “SEARCH” and “search”.

It should be noted that the order in which the strings to be matched appear in the function argument list is the order in which they are tested against the service output. Thus, the occurrence of two identical matching strings will result in the second string never being detected and is therefore considered an error.

Also, the function will ignore the occurence of “notification” strings but will react to the presence of “termination” strings, (resulting in the failure of the function).

Upon termination, the function returns either a ‘true’ or ‘false’ result. A ‘true’ result indicates that the function executed successfully. Under these circumstances the HyperTalk global variable ‘FoundString’  is set to the index value of the string parameter that was detected in the service output. If none of the string parameters were found the variable is set to zero.

eg. for the function call ...

ExpectFromService(SampleService, 20, true, true,“String1”, “String2”, “String3”)

If the first search string “String1” were found in the service output then the global ‘FoundString’ would contain the value 1. If the second search string “String2” were found in the service output then the global ‘FoundString’ would contain the value 2. If neither the first, second or third search strings were found then the global ‘FoundString’ would contain the value 0.

A ‘false’ result’ indicates that the requested handler did not execute successfully. Under these circumstances the HyperTalk global variable ‘ScriptStatus’ will contain a value that indicates the reason why the function failed and the variable ‘FoundString’ will contain a null value.

| | |
|---|---|
| Arg1 | Name of service to be scanned. |
| Arg2 | Number of seconds before a timeout occurs. |
| Arg3 | “true” or “false” determines whether a “moretime” dialog is to be shown when a timeout occurs. |
| Arg4 | “true” or “false” determines whether text matching is to be carried out in case sensitive or case insensitive mode. |
| Arg5 | Mandatory search string, max. 255 characters. |
| Arg6 | \| |
| Arg6 | \| Optional, additional search strings. |
| Arg7 | \| |
| Arg8 | \| |

| | |
|---|---|
| Returns | a) true - if the routine terminated correctly |
| | b) false - if the routine terminated with an error or if it was cancelled. |

c) **SendToService** Arg1, Arg2, Arg3

Allows a text string to be sent/written to the service.

The text string may contain character pairs that together define a control character. That is, the character “^” (shift 6) is interpreted as defining the following character to be a control character. The character is converted to its numeric control value by subtracting 64 from its character value, eg. ^A = control A and ^[ = escape.

Upon termination, the function returns either a ‘true’ or ‘false’ result.

A ‘true’ result indicates that the function executed successfully.

A ‘false’ result’ indicates that the function did not execute successfully. Under these circumstances the HyperTalk global variable ‘ScriptStatus’ will contain a value that indicates the reason why the function failed, see appendix D.

Normally this function and the function “ExpectFromService” appear in scripts, in pairs eg.

```
if SendToService(ActiveService, return, "") then
   if ExpectFromService(ActiveService, 20, true, "Select option") then
if FoundString = 1 then
```

However, this is not a requirement since each function can be used independently of each other.

| | |
|---|---|
| Arg1 | Name of service to which the text string is to be sent. |
| Arg2 | Text string to be sent to the service |
| Arg3 | Text modifier |

i) "slowly" text sent at 6 chars per second.

ii) "break" a long break is executed before the text is sent.

Returns a) true - if the routine terminated correctly

b) false - if the routine terminated with an error or if it was cancelled.

2) **Service Properties**

*All windows...*

a) get the visible of window windowname

...returns 'true' or 'false'

set the visible of window windowname to true/false

...requires a value of 'true' or 'false'

b) get the location of window windowname

...returns a screen point defined as 'horizontal,vertical'

set the location of window windowname to "point"

...requires a screen point defined as 'horizontal,vertical'

c) get the height of window windowname

...returns an integral number of pixels

d) get the width of window windowname

...returns an integral number of pixels

e) set the height of window windowname

...sets the height of the window to the specified number of pixels

f) set the width of window windowname

...sets the width of the window to the specified number of pixels

g) set the behindCard of window windowname to "true" / "false"

...if set to "true' then each time that this window is deactivated in favour of it's owning card window then the HyperTalk handler "WindowDeactivated' will be executed. If set to "false' no action will be taken when the window is deactivated.

*Text Edit windows...*

a) set the AppendText of window editname to "text"

...appends the text contained in "text" to the text edit window.

b) get the Text of window editname

...returns the "text" of the text edit window.

*Service windows...*

a) get the scripts of window servicename

...returns a slash (/) separated list of associated service script handlers

b) get the terminalSelection of window servicename

...returns the text framed by the current service window selection.

c) set the AttachLogFile of window servicename to "filepath"

...attaches the file defined by the path name contained in "filepath" to the service window and then proceeds to write the data received by the service to it. The manner in which data is written to the file depends on the current terminal emulation settings. The rules that apply are as follows -

If no terminal emulation has been defined for the service then ALL the data received from the service is written to the file.

If a terminal emulation has been defined for the service and the number of pages to scroll has been set to zero then ALL the data received from the service is written to file, including any special terminal emulation characters ie. the data is written to the log file before it is passed to the terminal emulation tool.

If a terminal emulation has been defined for the service and the number of pages to scroll is greater than zero then the data that is written to the file is that which is "scrolled off the top" of the terminal emulation screen. It must be noted that clearing the screen causes the whole contents of the screen to be "scrolled off the top", therefore this technique may be used to ensure that the last screen of a session is written to file.

NB. If there is no text file of the same name as that specified then, a text file will firstly be created. If a text file of the specified name already exists then the logged data will overwrite the existing file data.

3) **Service Messages**

*All windows...*

a) send ClearWindow to window windowname

...clears the contents of the window. If the window is a service window then only the contents of the terminal emulation will be cleared, the scrolled area is untouched. If the window is a text edit window then the entire contents are cleared.

b) send Activate to window windowname

...activates the named window ie. if appropriate, the window will be shown and made the frontmost active window.

*Service windows...*

a) send Copy to window servicename

...places the current selection on to the clipboard

b) send Paste to window servicename

...pastes the contents of the clipboard to the specified service

c) send SelectAll to window servicename

...selects (highlights) the contents of the window. If the window is a service window then only the contents of the terminal emulation will be selected, the scrolled area is untouched. If the window is a text edit window then the entire contents are selected

d) send DetachLogFile to window servicename

...detaches the attached file from the service window

e) send StartCapture to window servicename

...starts the capture of all text received by the service to a global variable whose name is constructed from the name of the service (see appendix D).

f) send StopCapture to window servicename

...stops the capture of all text received by the service

g) send ClearNotifyFlag to window servicename

...clears the flag associated with the service that triggers the execution of the "NotifyService" handler, (resulting from the detection of the notify string in the service output).

h) send SetUpService to window servicename

...performs essential service initialisation actions.

*Text Edit windows...*

a) send SetUpEditor to window editorname

...performs essential text edit initialisation actions.

4) **Useful HyperTalk Scripts**

*Functions...*

a) **OpenService(servicename, retainSettings, showWindow, showProgress)**

Launches the service named by "servicename". If the second argument is equal to "true" then the settings of the "Settings" submenu will be reinstalled from the previous activation of the service. If the argument is equal to "false" then the "settings" submenu will be set to default values. If the third argument is equal to "true" then the service window will be shown (and if appropriate, made the frontmost active window) after the successful launching of the service. If the argument is equal to "false" then the window will remain hidden following the successful launching of the service. If the fourth argument is equal to "true" then the progress of the service opening operation will be monitored via a "progress" dialog box. If the argument is equal to "false" then no "progress" dialog is shown. Returns either true or false to indicate whether the service was launched successfully or not.

b) **ListenService(servicename, retainSettings, showProgress)**

Initiates 'listening' for the service named by "servicename". If the second argument is equal to "true" then the settings of the "Settings" submenu will be reinstalled from the previous activation of the service. If the argument is equal to "false" then the "settings" submenu will be set to default values. If the third argument is equal to "true" then the progress of the service listening operation will be monitored via a "progress" dialog box. If the argument is equal to "false" then no "progress" dialog is shown. Returns either true or false to indicate whether the 'listening' operation was initiated successfully or not.

c) **OpenEditor(editorname, showWindow)**

Launches a text editor named by editorname. If the first argument is equal to "true" then the text edit window will be shown (and made the frontmost active window) after the successful launching of the editor. If the argument is equal to "false" then the window will remain hidden following the successful launching of the editor. Returns either true or false to indicate whether the editor was launched successfully or not.

*Commands...*

a) **CloseService servicename, showProgress**

Closes the named service. If the second argument is equal to "true" then the progress of the service closure operation will be monitored via a "progress" dialog box. If the argument is equal to "false" then no "progress" dialog is shown.

b) **CloseEditor editorname**

Closes the named text edit window.

c) **CloseAllWindows**

Closes all service and text edit windows.

## D. HyperTalk Globals Maintained by AYS.

AvailableResources — Contains a comma separated list of the names of all the available service resources within a custom built "resource" stack.

AvailableCards — Contains a comma separated list of the names of all the available service cards within a custom built "card" stack.

Services — Contains the number of active service windows for this stack. Initially set to zero.

Editors — Contains the number of active text edit windows for this stack. Initially set to zero.

Externals — Contains the total number of HyperCard external windows that have been opened during the current session (the sum of the "Services" and Editors" globals for each active AYS stack). Initially set to zero.

ServiceExternals — Contains the total number of HyperCard external service windows that have been opened during the current session (the sum of the "Services" globals for each active AYS stack). Initially set to zero.

EditorExternals — Contains the total number of HyperCard external text edit windows that have been opened during the current session (the sum of the "Editors" globals for each active AYS stack). Initially set to zero.

ActiveService — Contains the name of the currently active service, the value is set to the name of a service when either -

a) the service is launched.
b) the service is closed.
c) whenever a window associated with the service becomes the frontmost active window, eg. by clicking in the window.
d) an associated script is executed via the function "ExecuteServiceScript", see appendix C.

Initially set to empty.

ActiveEditor — Contains the name of the currently active text edit window, the value

is set to the name of a text editor when either -

a) the editor is launched.

b) the editor is closed.

c) whenever a window associated with the editor becomes the frontmost active window, eg. by clicking in the window.

Initially set to empty.

ServiceNames — Contains a comma separated list of all active service window names. Initially set to empty.

EditorNames — Contains a comma separated list of all active text edit window names. Initially set to empty.

WindowName — Used internally by the "ServiceWindow", "DoCService", "DoRService", "EditWindow" and "DoEdit" XFCNs to record the name of newly opened windows.

WindowToClose — Contains a comma separated list of the names of all the windows for which there are close requests outstanding. Initially set to empty.

StringNotifyService — Contains the name of the service for which a notify string has been detected. The global variable "NotifyString" will contain the string that was found. Initially set to empty.

ListenNotifyService — Contains the name of the service for which a 'listen' operation has terminated. The status of the operation is defined by the global variable "NotifyString" that will contain a value of either "Listen Accepted" or "Listen Failed". Initially set to empty.

NotifyString — Contains further information concerning the "StringNotifyService" or "ListenNotifyService" actions described above.

AllowCancel — This global determines whether the AYS HyperTalk extension functions "ExpectFromService" and "SendToService" (see appendix C) may be cancelled via the command-period keyboard combination. The global should contain a value of "true" (cancellation allowed) or "false" (cancellation prohibited).

If the global is set to "true" AND the HyperCard property 'CantAbort' is set to "false" then typing command-period will abort script execution in the normal fashion. If, however the HyperCard property 'CantAbort' is set to "true" then typing command-period whilst function "ExpectFromService" or "SendToService" is active will have the effect of setting the global variable 'ScriptStatus' to "cancel" (see below) and terminating the function with a return value of "false".

The global is initially set to empty.

ScriptStatus — This global is maintained by the three main HyperTalk extension functions "ExecuteServiceScript", "ExpectFromService" and

"SendToService" (see appendix C). The various settings are -

null - the function executed correctly and returned a value of "true"

error - the function encountered an error and returned a value of "false"

cancel - the function was cancelled and returned a value of "false"

The global is initially set to empty.

FoundString — This global is set by the AYS HyperTalk extension function "ExpectFromService" (see appendix C). Upon successful termination of function (returned value is "true" and the global 'ScriptStatus' is set to null) the variable is set to the index value of the string parameter that was detected in the service output. If none of the string parameters were found the variable is set to zero.The global is initially set to empty.

SetBusyCursor — This global is used by the AYS HyperTalk extension function "ExpectFromService" (see appendix C) to determine whether to turn the busy (beachball) cursor every 30 ticks (1/2 second) whilst attempting to match text strings in the service output. The global should contain a value of "true" (show/turn cursor) or "false" (don't show cursor).

The global is initially set to "true".

ControlData — This global is used by the AYS HyperTalk extension function "ExpectFromService" (see appendix C) to hold the accumulated text resulting from a match on a text string pair. That is, if the function is requested to search the service output for a text string pair, and a match is found then, upon function termination, the global will contain all the text, including the first text string, up until and including the second text string.

The global is initially set to empty.

DiscardControlData — This global is used by the AYS HyperTalk extension function "ExpectFromService" (see appendix C) in conjunction with the above global, (ControlData).

The global should contain a value of "true" or "false".

The value of the global indicates whether the data accumulated in the global variable "ControlData" should also be written out to the three possible output streams. The three streams are, the terminal emulation (if there is one) the data capture global variable (if the data capture feature has been turned on) and the data logging file (if one has been specified).

The global is initially set to "false".

In addition to the above, a further global may exist for each active service. These service specific globals are for the accumulation of captured data resulting from the use of the "StartCapture/StopCapture" extension described in appendix C. The name of the global is constructed from the name of the service with spaces and periods removed and "_CD" appended.

eg. a) the service named "Local OPAC" will place captured data into the global named "LocalOPAC_CD". A second activation of the same service would place captured data into a global named "LocalOPAC#1_CD".

b) the service named "Archie.au"

will place captured data into the global named “Archieau_CD”.

## E. AYS Error Codes.

1. This XFCN/XCMD has been called with invalid arguments or with an incorrect number of arguments.

   Check appendices A , B and D to determine the number and type of arguments that the specified XFCN/XCMD requires.

2. The requested action on the specified window is invalid since there are no currently open AYS windows.

   According to the internal data structures maintained by AYS there are no currently open (service or text edit) windows. AYS can only recognise windows that it has created.

3. AYS has exhausted its memory allocation resulting in an internal HyperCard error.

   An action requested by AYS of HyperCard has failed and is almost certainly due to the lack of available memory. HyperCard can be reconfigured to use a greater amount of memory. To check and/or change this setting, refer to the TeachText document “Installation Instructions”.

4. There is insufficient memory remaining for the following operation.

   The specified operation has failed due to the lack of available memory. HyperCard can be reconfigured to use a greater amount of memory. To check and/or change this setting, refer to the TeachText document “Installation Instructions”.

5. The Connection tool will not accept the “termination” string as a search string.

   The Communications toolbox Connection tool that you are using has refused to accept the termination string defined for the service. This error is likely to be Connection tool specific and we therefore suggest that you refer to the accompanying tool documentation.

7. AYS is attempting to use an invalid resource of the following type.

   AYS is unable to locate a specific resource. Either the stack has been corrupted or resources have been removed from the resource fork of the stack. To ensure that the stack contains a full set of resources reinstall one of the existing services.

8. This document has already been opened for writing and will therefore only be opened with “Read Only”, (RO) access

   This message is output if the text file specified at the launching of a text edit window is already opened for writing (by AYS, another stack or application) . AYS is thus prevented from writing to the text file.

9. AYS is unable to initialise the specified Connection tool.

   The Connection tool associated with the service being launched cannot be initialised. Firstly check that your system has all the necessary network software installed eg. MacTCP and then check the connections in the back of your machine. If neither of these are at fault then try reinstalling the affected service Connection

tool.

10. This document or its disk is locked. You MAY not be able to save any modifications.

    This message is output if the text file specified at the launching of a text edit window is locked by another stack/application or that the disc that it resides on is locked. You may edit the file, but you may lose any changes made at the time that you attempt to save the file.

11. AYS is unable to initialise the specified Terminal Emulation tool

    The Terminal Emulation tool associated with the service being launched cannot be initialised. Firstly check that your system has all the necessary network software installed eg. MacTCP and then check the connections in the back of your machine. If neither of these are at fault then try reinstalling the affected service Terminal Emulation tool.

12. The maximum number of strings, (11) have already been created

    The maximum number of strings that may be added to the “Strings” menu has been reached. To add a new string, one of the existing strings must firstly be removed.

13. AYS is unable to obtain environment data for specified tool.

AYS is unable to complete an internal Comms. toolbox related task. It is suggested that the system be rebooted.

14. AYS is unable to open the specified service Connection.

    The Connection tool is unable to open the required connection. Firstly check that your system has all the necessary network software installed eg. MacTCP and then check the connections in the back of your machine. If neither of these are at fault then investigate the network address that the service is configured for. Failing this, check your network connection.

15. The service script has failed to execute to completion.

    The service script just activated has failed. Correct the script and retry.

16. The Comms. Toolbox, system software has not been installed on this system. This must be done before AYS can proceed .

    It is recommended that AYS is run under System 7.0 however if you running under System 6.0.x and your system does not have the Communications toolbox installed then this message will result. You must install the Communications toolbox before proceeding.

17. AYS is unable to initialise one or more of the following system modules - the Connection, Terminal Emulation or Comms. Resource Managers.

    It is recommended that AYS is run under System 7.0 however if you running under System 6.0.x and the Communications toolbox has not been installed correctly then this message may result. You must reinstall the Communications toolbox before proceeding.

18. The maximum number of services, (10) have already been opened.

    The number of active services is limited to 10.

19. The maximum number of text edit windows, (10) have already been opened.

    The number of active text edit windows is limited to 10.

20. There are no recognisable Connection tools available for use with the Comms. toolbox on this system.

    The Communications toolbox on your system does not contain any Connection tools. You must install at least one before proceeding. To install the Connection tool that is bundled with AYS read the document “Installation Instructions”.

21. There are no recognisable Terminal Emulation tools available for use with the Comms. toolbox on this system.

    The Communications toolbox on your system does not contain any Terminal Emulation tools. You must install at least one before proceeding. To install the Terminal Emulation tool that is bundled with AYS read the document “Installation Instructions”.

22. The Connection being read from or written to is no longer open, (possibly timed out!).

This may occur if length delays are occurring on the affected host connection. Try relaunching the service to obtain a better connection.

23. AYS is unable to obtain the Connection status data

    Indicates that the Connection being read from/written to is no longer open. Try relaunching the service.

24. Error has been detected whilst opening or reading from the specified file.

    Indicates that there is an fault with the attached file. Try closing and then reopening the file.

25. AYS is unable to locate the specified Connection tool in the system folder.

    The Connection tool that the service being launched is configured for is not installed in your System. Reconfigured the service or install the required Connection tool.

26. AYS is unable to locate the specified Terminal Emulation tool in the system folder.

    The Terminal Emulation tool that the service being launched is configured for is not installed in your System. Reconfigured the service or install the required Terminal Emulation tool.

27 The specified Connection tool failed to accept the configuration tokens.

The Connection tool associated with the service being launched is unable to accept the configuration tokens defined for the service. This indicates that the too or the string of configuration tokens have been corrupted. It is suggested that you reinstall the affected service.

28 The specified Terminal Emulation tool failed to accept the configuration tokens.

The Terminal Emulation tool associated with the service being launched is unable to accept the configuration tokens defined for the service. This indicates that the tool or the string of configuration tokens have been corrupted. It is suggested that you reinstall the affected service.

29 An error was detected whilst writing to the active log file.

Indicates that there is an fault with the attached file. Try detaching and then reattaching the log file.

29 An error was detected whilst writing to the active log file.

Indicates that there is an fault with the attached file. Try detaching and then reattaching the log file.

30 The text accumulation window has reached its maximum size of 32000 characters.

AYS is limited to 32k text edit windows.

31 The specified file exceeds the max. allowable size (32000 chars.) and will be truncated upon input.

AYS is limited to 32k text edit windows.

33 The document cannot be saved since there is insufficient disk space remaining.

Try freeing up some disk space by deleting a file and then try to save the file again.

35 Unable to initiate a listen for this Service connection, (either there is a listen already in place or listening is not supported by the connection tool).

The listen attempt failed and the most likely reason is that either the Connection tool that has been configured for use with this service is unable to 'listen' (eg. a serial tool) or that there is already a 'listen' in effect.

36 This file cannot be written too since it has already been opened for writing.

This message is output if the text file specified for logging purposes or for the saving of a service window selection has already been opened for writing (by AYS, another stack or application).

37 This file cannot be written too since it or its disk is locked.

This message is output if the text file specified for logging purposes or for the saving of a service window selection is locked by another application or that the disc that it resides on is locked.

38 The specified text file does not contain any recognisable keywords.

Output following an unsuccessful scan of a text file. service parameters will have to "manually" specified.

39 The specified service name does not point to a valid HyperCard “card” within the current stack.

An action requiring the location of a HyperCard card associated with an AYS window has failed since the card cannot be found within the active stack. Either the stack has been corrupted or window has been incorrectly named.

40 The specified window name cannot be found in the list of active windows.

According to the internal data structures maintained by AYS there is no currently open window of the specified name (service or text edit). Either the stack has been corrupted or the window has been misnamed. AYS can only recognise windows that it has created.

41 The installer is unable to locate the target stack. Please redefine the target via the “Define Target” item in the “Function” menu.

Attempting to install or delete a service from a nonexistent target.

42 The resource fork of the target file cannot be opened.

Almost certainly due to the fact that the target resource fork has already been opened by another application. Terminate the offending application before proceeding.

43 The target stack does not contain the specified service definition.

Attempting to remove a service from a target stack that does not contain the service. Check the contents of the target stack.

44 The amount of memory that HyperCard has been loaded with is set too low. The minimum size necessary to run AYS is 1300k.

In order to run AYS successfully it is recommended that HyperCard is configured with a minimum of 1300kb memory partition. To check and/or change this setting, refer to the document "Installation Instructions"

45 The resource to be copied to the target stack cannot be found in this stack.

The installer stack is unable to locate one or more of the resources that are to be copied to the target stack. This indicates that the stack has been corrupted.

46 Invalid password.

Output if the password supplied to the "protected" service does not match that defined for the service.

47 The time restrictions associated with services launched by this target stack have now been removed.

Output to indicate successful unlocking of the target stack following registration.

48 AYS is unable to locate the specified logging file.

The path named supplied to the HyperTalk "property extension cannot be located. Check that the file exists and respecify.

49 There is a logging file already attached to this service.

Attempting to attach a logging file to a service that already has a logging file attached.

50 Conflicting (identical) strings detected in the function argument list.

The strings passed, in arguments 5 -> 9, to the function 'ExpectFromService' contain more than one occurrence of the same string. This will result in the second occurrence of the string never being found and is thus considered to be an error.

# F. Sample Scripts

**Example No. 1 - Use of a Logging File.**

```
on DownLoadMail LogFileName
 Global ActiveService, ServiceNames
 --
 -- If the service is not already active then launch it
 -- otherwise make the service window the active window
 --
 if "Sample Service" is not in ServiceNames then
    if not(OpenService("Sample Service", true, true, true)) then
       beep 2
       answer “Unable to launch ‘Sample Service’”
       exit DownLoadMail
    end if
 else
    send activate to window "Sample Service"
 end if
 --
 -- If there is enough room, position the service window
 -- below the card window
 --
 PositionWindows
 --
 -- Attach the receiving file to the service and then
 -- execute the service script that reads the mail and
 -- finally detch the logging file.
 --
 set AttachLogFile of window ActiveService to LogFileName
 put ExecuteServiceScript(activeService, "ReadMail", 3, true) into status
 send "DetachLogFile" to window ActiveService
 --
 -- If the reading of the mail went OK (the service handler
 -- returned ‘true’) then ask if its OK to log out otherwise
 -- post an error and logout
 --
 beep
 if not(status) then
   answer "Error detected whilst reading mail" with "Continue"
   CloseService ActiveService, true
 else
   answer "Logout from Service now ?" with "No" or "Yes"
   if it = "Yes" then CloseService ActiveService
 end if
end DownLoadMail


o=====================================================================o


on ReadMail
 Global ActiveService, FoundString
 --
 -- Start the process off by sending a carriage-return to bring up the
 -- menu, and then select the appropriate option to download the mail
 --
 get SendToService(ActiveService, return, "")
 if ExpectFromService(ActiveService, 20, true, true, "Select option") then
    if FoundString = 1 then
       get SendToService(ActiveService, "2" & return, "")
       --
```

```
      -- Keep reading service until “End of Mail” detected
      --
      if ExpectFromService(ActiveService, 300, true, true, "End of Mail.") then
         if FoundString = 1 then return true
      end if
   end if
 end if
 return false
end ReadMail


o=====================================================================o


on PositionWindows
 Global ActiveWindow
 --
 -- Determine if there is enough space on the screen to position the
 -- Service window below the card window.
 --
 put the height of the card window into xHeight
 put the width  of the card window into xWidth
 put (item 4 of the screenrect) - (item 2 of the screenrect) into yHeight
 put (item 3 of the screenrect) - (item 1 of the screenrect) into yWidth
 put the height of window ActiveWindow into zHeight
 put the width  of window ActiveWindow into zWidth
 if (yHeight - xHeight) > (zHeight + 50) then
   put the trunc of ((yWidth - xWidth)/2) into xLeft
   set the left of the card window to xLeft
   set the top  of the card window to 45
   put the trunc of ((yWidth - zWidth)/2) into zLeft
   put (zLeft - xLeft) into item 1 of sLoc
   put (xHeight + 25)  into item 2 of sLoc
   set the loc of window ActiveWindow to  sLoc
 end if
end PositionWindows
```

**Example No. 2 - Direct Data Capture.**

```
on CaptureReport
 Global ActiveService, ServiceNames
 --
 -- Firstly start the service keeping the window hidden
 --
 if OpenService("Sample Service", true, false, true) then
  --
  -- Now make sure that the card window cannot obscure the service window
  --
  set the behindCard of window “Sample Service” to true
  --
  -- Open succeeded so the login script should have been run
  -- and ActiveService global should now be set appropriately
  -- Now run the data capture script
  --
  if ExecuteServiceScript(activeService, "ReportCapture", 3, true) then
    --
    -- Capture script worked OK so we can now assume that there is
    -- something in the global variable that is specific to the service
    -- This can be retrieved via a call to the following funtion
    --
    Data = GetCapturedData(ActiveService)
    --
    -- or if the name of the service is known, is unlikely to change and
    -- if there will only ever be one activation of it at any one time then
    -- simply refer to the global directly eg. if the service name is
    -- “Campus Information” then refer to “CampusInformation_CD”
    --
  end if
  --
  -- Regardless of what happened close the service
  --
  CloseService ActiveService
end CaptureReport


o=========================================================================o


on ReportCapture
 Global ActiveService, FoundString
 --
 -- Step through the various prompts looking for the required one
 -- and then set capture on until the terminating prompt is found
 --
 if ExpectFromService(ActiveService, 10, true, true, "Some prompt") then
    if FoundString = 1 then
       get SendToService(ActiveService, "Some command" & return, "")
       --
       -- Found it so start capturing now
       --
       send StartCapture to window ActiveService
       get ExpectFromService(ActiveService, 10, true, true, "Terminating prompt")
       --
       -- Stop capturing regardless of whether the string was found or not
       --
       send StopCapture to window ActiveService
       --
       -- If the string was found then quit the process and return
       -- indicate ‘success’ to the calling code
       --
```

```
      if FoundString = 1 then
        get SendToService(ActiveService, "Quit" & return, "")
        return true
      end if
    end if
 end if
 return false
end ReportCapture


o=====================================================================o


function GetCapturedData(ServiceName)
 Global temp
 --
 -- This function constructs the name of the ‘captured data’
 -- global from the name of the service passed as a parameter.
 -- It does this by deleteing all spaces from the name and appending
 -- ‘_CD’ to the name. The function then retrieves the contents of
 -- the global value.
 --
 repeat with x = (the number of chars in ServiceName) down to 1
   if char x of temp = space then delete char x of ServiceName
 end repeat
 put "_CD" after ServiceName
 put "on LoadCapturedData" & return into TempHandler
 put "Global " & ServiceName & ", temp"  & return after TempHandler
 put "put " & ServiceName & " into temp" & return after TempHandler
 put "end LoadCapturedData" after TempHandler
 do TempHandler
 return temp
end GetCapturedData
```

**Example No. 3 - Script Cancelling.**

```
on mouseUp
 Global Servicenames, ScriptStatus, AllowCancel, SetBusyCursor, DiscardControlData

  -- Log in to target machine

  if not(OpenService("TargetMachine", true, true, true)) then
    beep
    answer “Login Failed"
  else
    --
    -- Set the AYS globals for the following enviroment -
    --
    -- Don’t turn the beachball cursor whilst searching for strings
    --
    put false into SetBusyCursor
    --
    -- Whilst searching for character string pairs don’t output to any
    -- of the output streams any of the service data that is detected
    -- and loaded into the global variable ‘ControlData’
    --
    put true into DiscardControlData
    --
    -- Allow cancelling of the AYS ‘search’ functions by the use of the
    -- Command-period key combination but prevent this action also
    -- cancelling the HyperTalk script that is running.
    --
    put true into AllowCancel
    set the cantAbort of this stack to true
    --
    -- Now execute the handler in the desired environment
    --
    put ExecuteServiceScript("GA", "TransferFile", 3, false) into reply
    --
    -- Reset the environment
    --
    set the cantAbort of this stack to false
    put false into DiscardControlData
    put true  into SetBusyCursor
    put false into allowCancel
    --
    -- Determine status and react
    --
    -- If the handler did not complete successfully then
    -- determine why it didn’t
    --
    if not(reply) then
      beep
      if ScriptStatus = "Cancel" then
        answer “Cancelling Transfer"
        get ExecuteServiceScript("TargetMachine", "CancelTransfer", 3, false)
      else
        answer “Transfer Crashed"
      end if
    end if
    --
    -- Regardless of result, close the connection
    --
    CloseService "TargetMachine", false
  end if
 end if
end mouseUp
```

**Example No. 4 - Character Pair Matching.**

```
on TransferFile
 Global ActiveService, FoundString, ControlData, DataFileName
 --
 -- Construct and transmit some machine specific command that starts the
 -- display of some file that we may capture to a logging file etc.
 --
 put "DISPLAY-FILE " & fld "AccountName" & space & fld "FileName" into Command
 if not(SendToService(“TargetMachine”, Command & return, "")) then return false
 --
 -- Now enter a loop that searchs for either -
 -- ‘###’ delimited strings that contain error messages - or
 -- ‘@@@’ delimited strings that  contain status messages
 --
 put true into status
 repeat
   if ExpectFromService(TargetMachine, 30, true, true, "###Δ###", "@@@Δ@@@") then
     if FoundString = 0 then
       exit repeat
     end if
     --
     -- Is it an error string, if so display it and exit the loop
     --
     if FoundString = 1 then
       beep
       put "status: " & ExtractMessage(controlData, "###") into fld "Status"
       exit repeat
     end if
     --
     -- Is it an status string, if so display it and then react
     -- according to its contents
     --
     if FoundString = 2 then
       put ExtractMessage(controlData, "@@@") into temp
       put "status: " & temp into fld "Status"
       --
       -- It is an initiate command, start logging the data
       --
       if temp contains "Transfer Initiated" then
         set AttachLogFile of window "TargetMachine" to dataFileName
         set cursor to cancel
       end if
       --
       -- If it is a cancel command, set the status flag and exit the loop
       --
       if temp contains "Cancelled" then
         put false into status
         exit repeat
       end if
       --
       -- If it is a completion command, detach the logging file, exit the loop
       --
       if temp contains "Transfer Complete" then
         send "DetachLogFile" to window "TargetMachine"
         exit repeat
       end if
     end if
   else
     put false into status
     exit repeat
   end if
 end repeat
```

```
 return status
end TransferFile

o=====================================================================o

function ExtractMessage Target, Text
 put the number of chars in Text into xx
 repeat
   put offset(Text, Target) into posn
   if posn = 0 then return(Target)
   delete char posn to (posn+xx) of Target
 end repeat
end ExtractMessage
```

**Example No. 5 - Miscellaneous.**

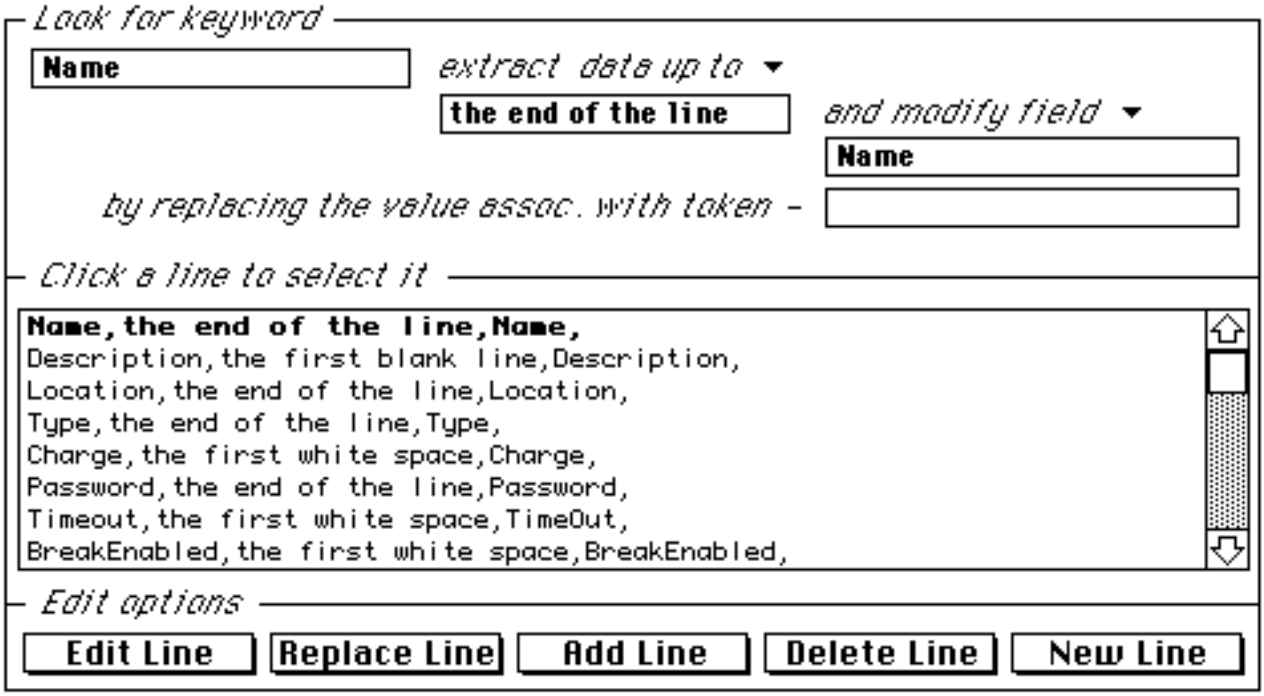
```
on EditServiceText
 Global EditorNames, ActiveService
 --
 -- Select the contents of the terminal emulation screen and then
 -- put the contents into a variable
 --
 send SelectAll to window ActiveService
 put the TerminalSelection of window ActiveService into Text
 --
 -- Create a text edit window called ‘Selected Text’ leaving the window
 -- hidden. Then clear the window and the append the retrieved data.
 -- Finally show the window
 --
 if not(OpenEditor("Selected Text", false)) then
    exit EditServiceText
 end if
 send ClearWindow to window "Selected Text"
 set AppendText of window "Selected Text" to text
 set visible of window "Selected Text" to true
end EditServiceText
```

**Example No. 6 - Standard AYS Handlers.**

```
function openService ServiceName, RetainSettings, ShowWindow, showProgress
 Global allowCancel, WindowName
 --
 -- Standard routine for launching a 'card' defined service.
 -- NB. If there is a 'LoginToService' handler and it fails allow
 --     the service to continue, (to allow debugging).
 --
 checkSpaceRemaining
 if ServiceWindow() then
   if DoCService("Open", ServiceName, RetainSettings, showProgress) then
     send SetUpservice to window WindowName
     set the visible of window WindowName to ShowWindow
     get the Scripts of window WindowName
     if it contains "/LoginToService/" then
       get ExecuteServiceScript(WindowName, "LoginToService", 1, showProgress)
       return true
     end if
   end if
   close window WindowName
 end if
 return false
end openService

o=====================================================================o

function listenService ServiceName, RetainSettings, showProgress
 Global activeService, WindowName
 --
 -- Standard routine for setting a 'listen' on a 'card' defined service
 --
 checkSpaceRemaining
 if ServiceWindow() then
   if DoCService("Listen", ServiceName, RetainSettings, showProgress) then
      send SetUpService to window WindowName
      return true
   end if
   close window WindowName
 end if
 return false
end listenService

o=====================================================================o

on closeService ServiceName, showProgress
 Global ScriptStatus
 --
 -- Standard routine for closing a 'card' defined service.
 -- NB. If there is a 'LogoutFromService' handler and it fails don't
 --     allow the service to proceed, (to allow debugging).
 --
 send ClearNotifyFlag to window ServiceName
 get the Scripts of window ServiceName
 if it contains "/LogoutFromService/" then
   get ExecuteServiceScript(ServiceName, "LogoutFromService", 1, showProgress)
      return false
   end if
 end if
 get DoCService("Close", ServiceName, showProgress)
 close window ServiceName
 return true
end closeService
```

```
function openEditor EditorName, ShowWindow
 Global WindowName
 --
 -- Standard routine for opening a Text Edit window
 -- The font and point size of the font are picked up from a
 -- 'Preferences'card.
 --
 if EditWindow() then
   put fld "Font" in card "Preferences" into Font
   put fld "Size" in card "Preferences" into Size
   if DoEdit("Open", EditorName, Font, Size)  then
     send SetUpEditor to window WindowName
     set the visible of window WindowName to ShowWindow
     return true
   end if
   close window WindowName
 end if
 return false
end openEditor

o=====================================================================o

on closeEditor EditorName
 --
 -- Standard routine for closing a Text Edit window
 --
 get DoEdit("Close", EditorName)
 close window  Editorname
 return true
end closeEditor

o=====================================================================o

on closeAllWindows
 Global  Services, ServiceNames, Editors, EditorNames
 --
 -- Closes all the windows defined as being open.
 --
 repeat with x = Services down to 1
   closeService item x of ServiceNames, true
 end repeat
 repeat with x = Editors down to 1
   closeEditor item x of EditorNames
 end repeat
end closeAllWindows
```

## G. Text File Scanning Rules




The keyword table may contain any number of lines, each line containing information on a single keyword.

1. The first item of a keyword line contains the keyword to be scanned for, only single words, containing only alpha-numeric characters are allowed.

   The rules for scanning the text for keywords are as follows -

   Only alphanumeric characters may be used to construct words, (all non-alphanumeric characters are ignored).

   eg. Service-Name becomes ServiceName
   $Amount becomes Amount.

   All comments are ignored. Comments are defined as being prefixed by either a hash or an exclamation character (**#** or **!**) and terminated by a carriage-return.

   eg. ServiceName=Uni Opac # Provides access to the university's OPAC
   Password ! No password required for this service.

   The characters used to determine the end of a keyword are a **space**, an **equal** sign, **colon** or a start comment character.

   eg. Password Fred
   Password=Fred
   Password:Fred

Password = Fred
Password : Fred
Password#
Password!

2. If a keyword is detected in the scanned file then the second item of the table line defines how much of the data that follows it is to be extracted. The options are either to extract data up until the first space (or comment character), the first carriage-return (or

comment character) or up until the first blank line (comments are treated as part of the data).

3. The extracted data is then used to modify the HyperCard "New Service" card field named by the third item of the table line. The rules that apply here are dependent on whether there is a fourth table item or not.

4. If there is no fourth item then the contents of the named field are overwritten with the extracted data. Otherwise the fourth item contains a single word parameter (token) name that is to be searched for in the named field. If the "token" is found in the field then the single word value that follows it will be replaced by the extracted data. This feature is only of use if Communications Toolbox tools being used have recognisable "tokens".

For example -

a) ServiceName,the end of the line,Name

is interpreted as -

Scan the text file for the keyword "ServiceName" and if found extract the data that follows the keyword up until the end of the line and then replace the contents of the background field "Name" with the extracted data.

b) Address,the first white space,ConnectionTokens,HostName

is interpreted as -

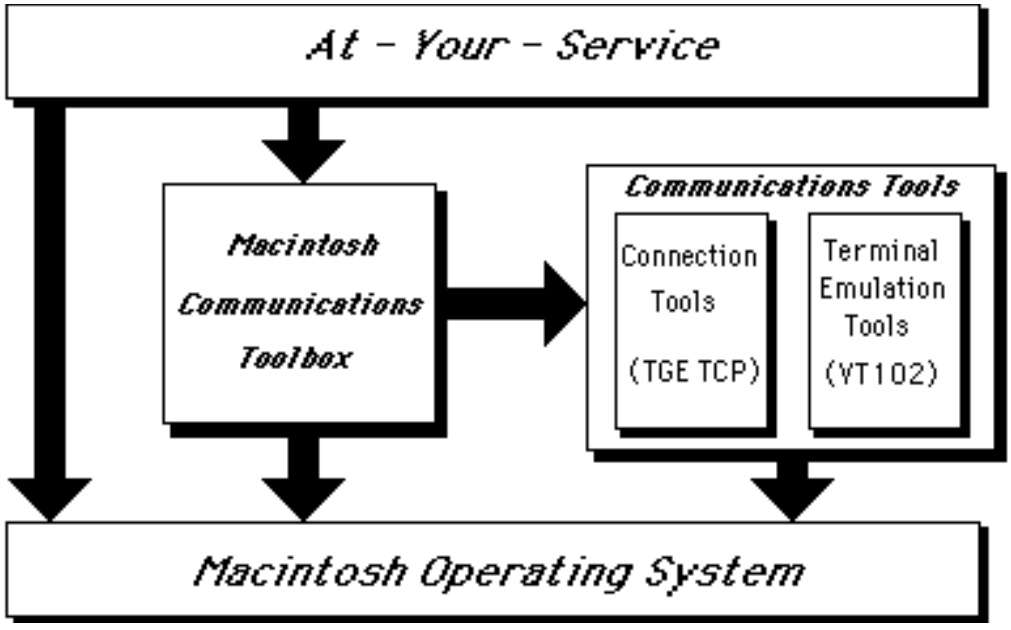
Scan the text file for the keyword "Address" and if found extract the data that follows the keyword up until the occurrence of the first space. Locate the configuration token "HostName" in the background field "ConnectionTokens" and replace it with the extracted data.

## H. The Communications Toolbox

The Macintosh Communications Toolbox is system software that provides Apple Macintosh applications with standard access to communication services, including data connection and terminal emulation.

The Macintosh Communications Toolbox is Apple computer's strategic communications development platform. It is designed to support multivendor connectivity for Macintosh computers in environments such as Digital, IBM, OSI, TCP/IP, and the Appletalk network system.

An extension of the Macintosh Toolbox system software, the Macintosh Communications Toolbox consists of several managers such as the Connection manager and the Terminal Emulation manager. These managers have been designed to work in concert with tools, such as a modem connection tool or VT102 terminal emulation tool, created by Apple and third party developers to provide applications with standard communications functions.




Communications Tool Box tools are stored in the extensions folder inside the System 7"s "Systems" folder. You need to determine which tools to use, which is dependant on the hardware you use to connect to other machines.

Serial Cable This type of connection requires the serial tool; only one connection is possible at a time with most systems.

Modem This type of connection requires a modem tool of which several are available. Only one connection is possible at a time.

Ethernet If you have access to a network that is TCP/IP based you can use MacTCP and a TCP/IP tool such as TGE TCP tool supplied with AYS.

You will need to determine which terminal emulation is supported by both your Macintosh and the remote computer. The most common emulation is vt100 and fot this you can use the vt102 tool. Other emulations are available and you should check with vendors of Communications Toolbox tools.