Apple® IIGS

Versions
Developer Manual

Dedicated to the memories of Joe Kohn 1947-2010
and Ryan Suenega 1967-2011

Credits

My thanks go to all who have helped me develop Versions, and especially to Chris Vavruska for his many suggestions, his extensive beta-testing, and his help with integrating the GSummer NDA.

Getting started in twelve easy steps

1. Install Versions (see Page 7 Users Manual).
2. Start up and Quit Versions.
3. Place marker file named 'Developer' in 'Versions.Data' folder (Page 6).
4. Start up Versions.
5. From the 'Developer' menu, open 'Developer Preferences'.
6. Enter the URL, download folder path, User Name and Password, for the FTP access to your web server (Page 16).
7. Select ' Edit File List' from the Developer menu.
8. Click 'New' and then 'Save' with a suitable name for your List file, and Click 'Save' (Page 9).
9. Now 'Add' entries for your downloadable archives (Page 10).
10. Click 'Send List' (Page 13).
11. Click 'Save' (Page 15).
12. Edit and update each file entry as and when you release new updates.

Contents

Preface

Welcome to Versions - Developer

Introduction

Please read the Versions Users Manual first, as it explains how the User interface to the application works, and will make more sense of what follows here in the Developer Manual that lets you support upload checking for your applications.

Versions allows you as a Developer to give an easy way for your users to know that you have a update to a project or have released a new project.

As a Developer, you will see an additional 'Developer' menu, where you create and edit the reference lists that Versions uses to scan for outdated files. From there, you can upload these reference files to your web site directly using its FTP portal, and if necessary, upload your application archive files as well, generating and sending MD5 checksum files automatically.

You can also create, edit, and send .inf (information) files to the server,. These are small text files that describe your applications, and which Versions displays from the Updates list Info option.

From then on, you just maintain and update these reference and information files whenever you make changes to your software, or add new files. Users will then instantly be notified of any updates to their favourite software.

Check out some of my other contributions to the IIgs world on my web site:

http://speccie.uk

Versions Update Checker

Requirements

Please refer to the User Manual on how to install Versions.

To activate the Developer Menu, add a dummy marker file called 'Developer' to the Versions.Data folder. When you next start Versions, you will see the Developer menu to the right of the Menu bar. Alternatively, hold down the Closed and Open Apple keys at boot to temporarily turn on the Developer's Menu:

.

To obtain Versions, the UNDO Manager, or the FTP Tool,

and any of my other software:

http://speccie.uk

To obtain the latest version of the Marinetti TCP/IP stack:

http://www.apple2.org/marinetti/

To find out about, and order an Uthernet card:

http://www.a2retrosystems.com/

Versions Developer Quick Reference

The Menus

Apple Menu

File Menu

Edit Menu

Developer Menu

Edit File List
Create Info File
Upload Files
Developer Preferences

Developer Menu

Edit File List

Creates File Lists, Add, Edit, and Delete items to and from a selected File List, and then Send the File List to your web server.

Create Info File

Create and Edit the .inf files that describe your applications, and then Send the .inf files to your web server.

Upload Files

Uploads files, archives, and their optional .md5 files, and .inf files.

Developer Preferences

Set Developer Preferences, and set the web site FTP portal log-in details.

Working with Developer Versions

Strategy

How Versions Works For You

As a Developer, you simply prepare a reference List using the File List Editor within Versions, and upload it to the web folder that holds your application archives. When you issue new updates to your applications, or release new applications, you simply edit that file and upload it once again to your web server.

The reference List file holds information about each application, which is then used to identify applications on your server that are newer than previously installed versions on a User's disk, and also holds the name of the archive files on your web server so they can be downloaded.

You publish a link to that reference List file, which Users can manually add to the Version.List file in their installed copy of Versions. Alternatively, you can send the List file to me at: spectrumdaddy@mac.com. I will add it to a vlist.updater file, so it is automatically added to the Users Version.List file when they scan for updates.

The next time the User checks for updates, the updated reference List file will be downloaded, and used to scan their disks for any new updates.

If you have more than one downloads folder on your web server, just create extra reference List files as needed, and publish the extra links to the Users.

You can optionally create short 'Info' files using Versions, which are uploaded to that same web directory. These short files would explain what your application does, and will be downloaded and displayed when the User selects 'Info' for an application.

It takes very little time to keep these reference files updated, and so Users will always see the latest updates and information files that you may have posted.

The Reference Lists

The reference List files you create and upload to the downloads folder on your web server, hold the information that allows Versions to do its magic. It is up to you as an author to keep these List files up-to-date with any changes or additions to your software portfolio. The path in the URL you supply is used to point to the Download folder where your archives can be retrieved from.

You can choose to distribute the URL that points to the List file yourself, or to give it to me at spectrumdaddy@speccie.uk, so I can add it to the 'vlist.updater' file I maintain. Any URLs added to that updater file will automatically be passed to the User's Versions.List file the next time they make scan of their hard disks.

Creating and Editing a List file

If you have not yet set up the access details to the FTP host for your web server, you will be prompted to do so when you select 'Edit File List'. Refer to the Developer Preference section for more details.

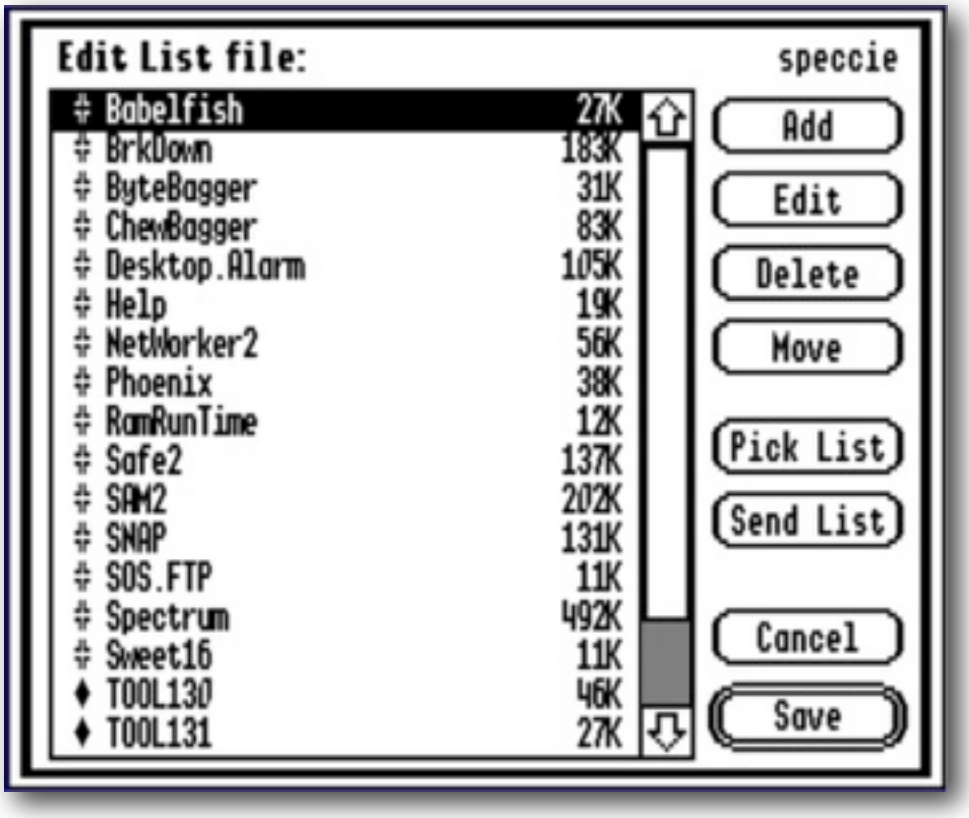
If you have not yet created a List file, you will also see the 'Select List File' dialog.

To create a List file to work with, click the 'New' button. Choose a unique ProDOS compatible name for the List File, and click Save. If you need more than one List file, click 'Pick List' from the main window, and create 'New' lists as required.

You can also 'Delete' any old List files you no longer need from here.

Only one List can be active at any one time, but you can easily select another List as required from the 'Pick List' button from the 'Edit List file' dialog.

The Editing List file dialog



Add button

To add an application to the List, click 'Add', and from the 'Add File to List' dialog, click 'Set', and choose the application. It is important that you select the latest version of the application, so the correct file information can be saved.

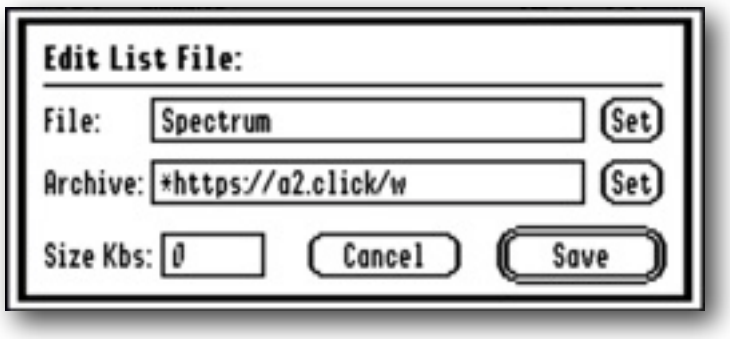
Next, either enter the archive filename into the 'Archive' LineEdit, or click 'Set' and choose the archive directly. The 'Size KBs' box displays the size of the archive file. This can be changed if necessary, but do not make it smaller than the actual archive, as that might make a subsequent download fail if the User does not have sufficient space on their hard disk. You can change this size if necessary.

Note: The List file editor lets you choose the files or applications that become the List file. It is important for the whole Update scan process to work properly, that you select the version of your application that is within the archive file. The entry added to the List gets sent to the User, and holds the File name, the Archive name, the full set of file attributes, if present the Version number from a Version resource, and the size of the archive file. Versions uses the exact file name to get the `GetFileInfoGS` file information for the subsequent match during the update scan to determine if a file is out-of-date, and where to find its related archive for downloading.

Note: If either the File or Archive names have not been set, you will not be able to Save the entry.

Multiple File Sets

In some cases, as is the case with Spectrum™, it is not practical to have only one archive linked to a specific application for download. Versions caters for this, by allowing a special entry in the List file, so that when the subsequent scan is made, if a newer version of the application is identified, instead of downloading the related archive, a URL will be shown to a web page where multiple choices of download will be available. This link will also show if you click the 'Info' button.



Select the applications as usual, as its file information is required for the match, but instead of entering an archive file, enter the archive name as a link to a download or other web page with an '*' before the 'http://www.com/' link.

If necessary, you can dispense with 'http://server.com/', and just enter the path to the web page with no leading '/'. In that case, the base URL for that reference List file will be added to the path to point to the relevant web page.

If the full URL link will not fit into the 34 character limit of the Archive name, then use a 'bitly' or other shortened URL. Juiced.GS provides a service for creating shortened URLs: https://a2.click/

If you are not yet registered with Juiced.GS to allow the creation of a2.click names, go to this web page and register: https://juiced.gs/a2click/

Edit button

Opens the selected entry from the List. You can change the Archive name and size in their respective LineEdits, or use the 'Set' buttons to choose new files and archives for the entries as and when you release updates.

When you release an update to your software, use this function to update the name and size of the new archive you post. In those cases, you must select the application once again, so the new file information is saved, though the File name would obviously stay the same.

Delete button

Removes an entry from the list for a file or application that you no longer support, or when you have removed the downloadable archive from the web server.

Move button

Moves an item in the List, to change the order of the List. When the User checks for updates, the items from the List will be scanned in the sequence shown.

Select an Item to move, click the Move button, then double click where you want it to go. The List will be redrawn to show the change.

Remember to 'Save' the changes you have made to the List.

Send List

This function is provided for your convenience. If you wish, you can just send the List file from the Versions.Lists folder to the Downloads folder on your web server using your normal method for uploading files.

You must first have entered the Host server URL, the download Server folder, and the User Name and Password into the Developer Preferences dialog. Refer to the section on 'Developer Preferences' for more details.

If you are hosting files on more than one web server, you may need to change these details when you select another List File.

Click the 'Send List' button to send the file using FTP to your web portal. A window will show the upload progress. If there are any problems, check the Log file, and if necessary use the Debug feature.

Create Info File dialog

When the User views their 'Pending Updates' window, they can choose to select an item, and click the 'Info' button to view information about that file. Versions will check on your web server for a text file with the exactly the same name as the application, and with a suffix of '.inf", eg. 'spectrum.inf". If such a text file is found, it will be downloaded, and if it contains a comment tag, this will be used to add information to the top of the display in the window. If an 'inf" file was found, the 'Info' button in that window will be active, and if clicked, any further text from the '.inf" file will be displayed in a new window.

The files are downloaded from a web server, so the process is almost instantaneous, this allows you to update the Information files as and when necessary, and the User will always see the latest version. It is suggested that you do not make these files large, as they are only intended to be a quick reference!

You can create these files in any text editor, saving them as plain text files, giving them the suffix '.inf", and either upload them manually to your web server, or use the 'Send' button. The file will be displayed as plain text, so try not to use non-standard ASCII characters that would not display in the SNAP8 font that is used by Versions. Alternatively, you can create an .inf file using basic HTML code, which will be parsed by the HTML Tool, and displayed as an HTML display. The HTML Tool can only handle basic HTML commands, so check how your files will look by adding a .htm suffix, and opening them in the Spectrum™ Editor.

If you add descriptive comment tags, they must be created in one of these two formats. For instance, for SAFE2, you could use something like this:

```
HTML:   <!--Versions--FTP client--A powerful file transfer FTP
client--2.3.3--safe234.bxy-->

Plain Text:  **--Versions---FTP client--A powerful file transfer FTP
client--2.3.3--safe234.bxy--
```

The tag or asterisks mark the start of the line, and is followed by '--Versions--'. There are four segments, ended by '--' markers. The text of the first shows at the top of the Info dialog (max 34 chars), and describes the application, the second (max 58 chars), shows on the next line, and describes what the file does. The third shows the latest version number of the application in VersionString format, and the fourth is the filename of the latest archive of the application.

Note: You can use any punctuation you like within each segment description, except for a double '--', which marks the end of each segment.

If you select 'Create Info File' from the Developer Menu, it opens a simple text editor, where you can easily create these small plain text files.

If you want to create an 'Info' file from within Versions select 'Create Info File'. The 'Uploads' directory will be displayed in a standard file dialog. You can either 'Open' an existing .inf file to load its text for editing, select an application to use its name for the subsequent 'Save' function, or click 'Cancel' to start a new file, both of those last two choices will display a blank window ready for editing.

Except when you click 'Cancel', the file name that refers to the application will be picked up and will replace 'Edit Info Files' at the bottom left of the window. This name will subsequently get .inf added when you go to 'Save' or 'Send' the file.

Note: So plain text files can be saved, no style information can be used in this window. The Undo Manager is active, so as well as Cut, Copy, and Paste, you can Undo and Redo any text you may accidentally delete.

Note: Info files will always be 'Saved' to the 'Uploads' folder.

Load File button

At any time while editing, if you wish to select a new file to work with, click the 'Load File' button. The new text will be loaded, and the name at the bottom left will change to the new file name.

Insert button

Inserts text from other text files at the cursor position. This does not change the 'Save' name.

Send button

This function is provided for your convenience. If you wish, you can instead just send the Info file to the Downloads folder on your web server using your normal method for uploading files.

You must first have entered the Host server URL, the download Server folder, and the User Name and Password into the Developer Preferences dialog. Refer to the section on 'Developer Preferences' for more details.

If the 'Save' name has not yet been defined, you will be prompted for one.

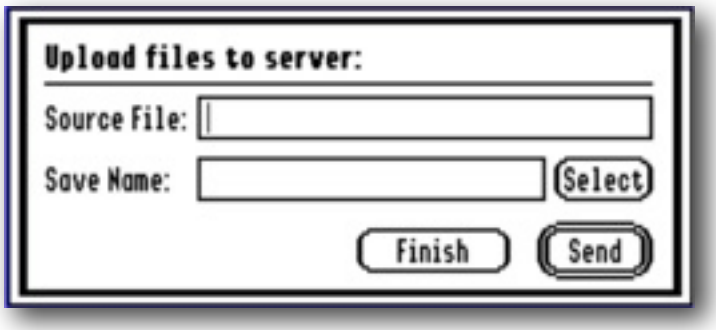
A window will show the upload progress, and the Log file will show whether the upload was successful or not. if there are any problems, check the Log file, and if necessary use the Debug feature.

Save button

Saves the Info file to the Uploads folder using the name displayed at bottom left, and adding the .inf suffix. If the 'Save' name has not yet been defined, you will be prompted for one.

Upload Files

As when sending a List file, this requires the FTP server portal details to be entered in Developer Preferences. It will upload files to the designated web server and folder defined in those settings.



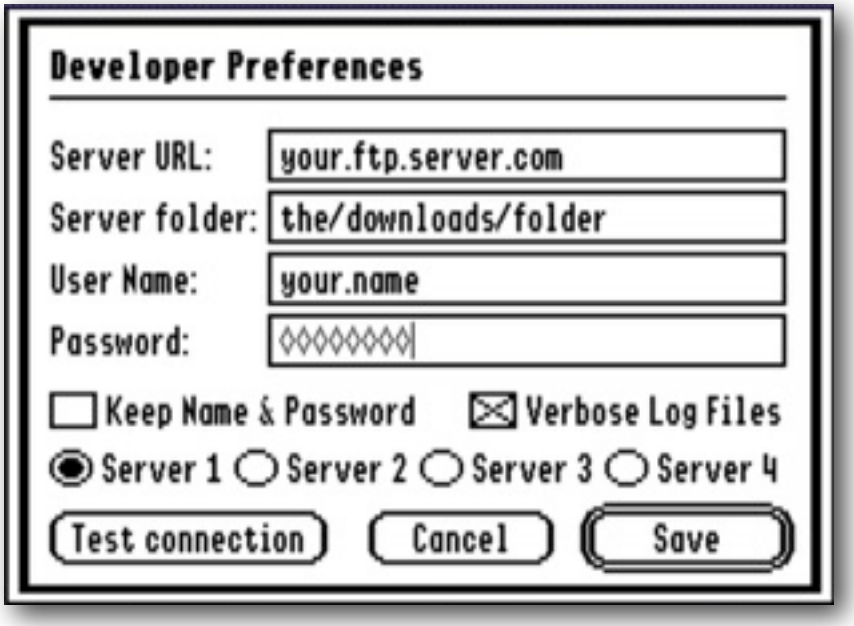
Either enter the path to the file to upload, or use the 'Select' button to choose it directly. The Save Name that the file will be saved as on the server will be filled in automatically, but you can change the name if you wish to use a different one.

You can upload any kind of file you like, but if you have the GSummer NDA installed, when you send a file with the suffix 'SEA', 'SHK' or 'BXY', a matching .MD5 file will be generated and sent as well. The Log file will show you what files were actually sent.

If you need to send files to multiple servers, you will need to change the settings for the host web server portal in Developer Preferences as necessary.

It is optional that you send files to the web server using this function, as they can just as easily be uploaded using any FTP application such as SAFE2.

Developer Preferences



Before you can upload either a List file, or Info file from the 'Edit File List' or 'Create Info File' dialogs, or use the 'Upload Files' function, you must enter the login details for the web server portal you are using for the downloads of your software.

Server URL

The address of your FTP web server. A suffix of ':xx' will specify a different port.

Server folder

The path to the downloads folder on your web server. This will be added to the File name, and the Server URL, to complete the full path that is used to Store the files.

User Name

The Username to access your account.

Password

The Password to access your account.

Test Connection

Tests that a connection can be made to your server by downloading a Directory List for the path you have given. This is done in the background, so all you will see is a confirmation that all is well, or an error message.

Keep Name & Password

If you check this box, your Username & Password will be saved in the Dev.Prefs file at Quit, and will be available the next time you open Versions. If you do not check this box, your Username & Password will not be saved, and will be deleted at Quit.

Verbose Log Files

By default, Versions saves a record of most actions that you take. By turning this feature off, the Log file will hold less information, but will still keep a record of error events as they happen.

Server Selection

For convenience, you can keep up to four server setups, each setting holds a complete set of data from the dialog controls, and the name of the List file created.

Save

Click Save to save changes you have made to the Dev.Prefs file. If you click Cancel, those changes will be lost.

Postscript

If you need to manually work with the downloads folder on your web server, I suggest you set up a 'Favourite' in SAFE2, and use that powerful FTP client to Upload, Download, Delete, Create folders, or Rename files on the web server directly. Alternatively you can use a corresponding FTP client on your Macintosh, Windows, or Linux computer.

Appendix

The Update Scan Process

A List file holds a number of entries, one for each of your applications. Each entry for an application or file is structured to carry all the information needed to check whether the file on your host server is newer than the installed copy on the User's hard disk, and to show the name of the archive file:

```
ListEntry:
List_marker         dc i'$7376'    ; marker for listing (vs)
LocFileName         ds 35          ; pString                          +2
TarFileName         ds 35          ; pString                          +37
FileAttributes:
access              ds 2           ; access flags                     +72
filetype            ds 2           ; filetype                         +74
auxtype             ds 4           ; auxiliary type                   +76
storage             ds 2           ; storage type of file             +80
CreateTime          ds 8           ; date record                      +82
modtime             ds 8           ; date record                      +90
optionList          dc i4'0'       ; input buffer Pointer for FST     +98
eof                 ds 4           ; result                           +102
blocksused          ds 4           ; result                           +106
resourceEOF         ds 4           ; result                           +110
resourceBlocks      ds 4           ; result                           +114
RezVersion:
LocVersion          ds 6           ; Version Rez                      +118
TarFileSize         ds 2           ; size in Kbytes                   +124
List_endmarker      dc i'$7265'    ; marker for listing (er)          +126
```

Each entry is $80 bytes long, and is marked by a head and tail word that is used to check that the entry is valid. The other entries are as follows:

LocFileName - The name of the application or file to be checked. The name is case-sensitive.

TarFileName - The name of the target archive file on the web server that will be downloaded, or the URL of a web page (see Multiple File Sets).

FileAttributes - The GetFileInfoGS record for the LocFileName.

LocVersion - The version number from the Version Rez if present.

TarFileSize - The size of the TarFileName archive in Kilobytes.

Note: The base URL from the List file URL for your server, will be added to the TarFileName pString, to point to the folder on your web server that holds the downloadable archive file.

How Versions Scans for Updates

When the user selects 'Check for Updates', a number of events occur. First of all, a check is made to see if there are any new links that need to be added to the Versions.List file, and if there are any links that need to be removed from the file as they are no longer valid. An alert will be shown if either of those events are true, to give the user the option to edit their copy of Versions.List before proceeding.

The reference files listed in Versions.List will then be downloaded from their respective servers, and parsed into one Index file. Any Files that have been listed in the Black List will be removed at this point.

The disk scan process now starts, and scans either the Boot Disk, or all the disks, depending on the setting in the User Preferences. Paths, folders, and disks listed in the Black List will not be scanned.

Every file found on a disk will be checked against the entries in the Index file using a case sensitive match. When a name match is made, the file type and auxiliary type are matched, then the storage type matched. If the matching process gets that far, and there is a version resource, that is matched to see if the file on disk is older. If there is no version resource, the size of the file on disk, and the modification date is matched.

Depending on whether the file matched is seen as being older, the same, or even not installed, the entry from the Index file will be displayed in one of the three sections of the Pending Updates list.

Note: There is a limit of 511 files that can be added to the Index file. I think that limit is unlikely to be reached!

Debug Option

If you are having problems with either Uploads, Downloads, or server access for any of the online functions, hold down the Closed Apple key when starting any process that goes online. A transcript of that session will then be created in the top directory of the Boot disk.

For an HTTP session, the transcript will be named `HTTP.log`, and for an FTP session, it will be named `FTP.Log`.

GSummer Support

If GSummer is installed, MD5 files will be generated and sent whenever a 'SEA', 'BXY' or 'SHK' file is uploaded.

Files Used by Versions

Versions creates a number of folders and files to handle the scan process:

The working Versions folder:

`Versions` - Application. The **Versions** application

`Versions.List` - Text file. User editable links to reference files on the authors web sites.

`Versions.Lists` - Folder. Containing:

Binary files. A number of reference files download from authors web sites.

`Versions.Logs` - Folder. Containing:

Text Log files.

`Versions.Data` - Folder. Containing:

Binary: User.Prefs

Binary: Dev.Prefs

Binary: Black.List

Developer - Dummy marker file.

`Versions.Temp` - Folder. Containing:

Binary - Index file

Binary - A number of List.xx files

Text - A number of URL.xx files

`Uploads` - Folder. Containing:

Archive files, and .INF files, for uploading to servers. **Versions** will also create matching MD5 files for any with a 'SEA', 'SHK' or 'BXY' suffix.

HTML recognised font color and font bgcolor tags:

black, blue, navy, darkgreen, olive, darkgray, gray, red, maroon, lilac, purple, orange, pink, fuchsia, green, lime, aqua, cyan, lightgreen, chartreuse, lightblue, teal, lightgray, silver, cornflowerblue, yellow, and white.

Refer to the html_manual.pdf on the web site for more details.

Programming Information

vlist.updater

The vlist.updater file is used to automatically update and maintain the Versions.List file, so you do not need to pass the URLs to your List files directly to the User. Initially, Versions.List is coded to point to files on my own web server, but optionally can be changed to point to another web server.

The resource fork holds a pString resource: ID $00001000, that holds the URL path to the vlist.updater file. This path and name can be changed as required, just make sure it is a valid path or scans might fail.

The file itself is constructed in a similar way to Versions.List:

```
*=5
* You can edit this Versions.List as you please.
* Comment lines must start with an asterisk and space.
* To disable an entry, so it does not get added back in again,
* add '** Disabled: ' to the existing URL like this:
** Disabled: http://somewhere.com/folder/file

*+5
* Software from Speccie...
* Explore my web site for the PDF manuals, and other goodies:
*       http://speccie.uk
* Speccie's Versions.List file:
http://speccie.uk/speccie/downloads/speccie
```

There are three special REM statements used by vlist.updater, that are used when the vlist.updater file is downloaded by **Versions** and parsed. The special REM statements are handled in this way:

*=x Checks to see if the entire following section exists in the User's copy of Versions.List, and if not, inserts that entire section.

*+x Checks to see if the URL in the last line of the section exists, and if not, inserts the entire following section. It ignores any existing sections with the same URL but are disabled.

*-x Checks to see if the last line of the section exists in the User's copy of Versions.List, and if so, adds '** Disabled: ' to the start of the URL line, thus disabling it so it will be ignored in future scans.

Note: x = number of REM lines, plus the URL line in this section. Range is from 1-9, not including the special REM statement line.

The rUpdateInfo Resource

From an idea by Eric Shepherd, I am proposing a new Resource format that could be used in future to aid update applications such as Versions.

Edit your 'types.rez' library file to include this resource format:

```
#define rUpdateInfo $DEAD  /* RIP Jerry Garcia */

/*---------------------- rUpdateInfo -------------------------*/
type rUpdateInfo {
    integer = 0;           /* version must be zero */
        longint;           /* Resource ID of string holding app's display name */
        longint;           /* ID of string holding author's name               */
        longint;           /* ID of string URL to the downloads folder         */
        longint;           /* ID of string for downloadable archive name       */
        longint;           /* ID of string for Versions.List data file         */
};
```

Then in your rez file, include these templates and definitions:

```
// --- rPString Templates

// app's display name
resource rPString (PSTR_00002000, $0000) {
       "Versions"
};

// author's name
resource rPString (PSTR_00002001, $0000) {
       "Ewen Wannop (Speccie)"
};

// URL to downloads folder
resource rPString (PSTR_00002002, $0000) {
       "http://speccie.uk/speccie/downloads/"
};

// Name of downloadable archive
resource rPString (PSTR_00002003, $0000) {
       "versions1.0.bxy"
};

// Name of Versions.List file
resource rPString (PSTR_00002004, $0000) {
       "speccie"
};

// --- rUpdateInfo Definitions

resource rUpdateInfo (RUPDATE_00000001, $0000) {
       PSTR_00002000,              // app's display name
       PSTR_00002001,              // author's name
       PSTR_00002002,              // URL to downloads folder
       PSTR_00002003,              // Name of downloadable archive
       PSTR_00002004               // Name of data file for Versions.List
};
```

Using the Resource:

Make sure the resource is defined as $DEAD and has a resource number of $00000001, that all five pStrings are present (their resource numbers are not important, as they will be picked up from the rUpdateInfo resource itself), and if possible have been filled in, or left as null pStrings. It is important that the fields are valid entries, especially 'Name of downloadable archive' and its 'URL to downloads folder', as those will be used for the 'Get' button in Versions and Check File to retrieve the archive.

If you have placed a Versions format .inf file onto your server, please make sure it contains either one of these descriptive comment tags. If that tag is not present, Check File cannot fully handle any downloadable archives that are newer that you may have on your server. This is the only way that you can tell the user through Check File that there is an update waiting.

The tags must be created in one of two formats, depending on whether the .inf file is plain text, or formatted as HTML. For instance, for SAFE2, you could use something like this:

```
HTML:   <!--Versions--FTP client--A powerful file transfer FTP
client--2.3.3--safe234.bxy-->

Plain Text:  **--Versions---FTP client--A powerful file transfer FTP
client--2.3.3--safe234.bxy--
```

The <! tag or double asterisk marks the start of the line, and is then followed by '--Versions--', which defines it clearly as a custom tag . There are four segments, each ended by '--' markers. The text of the first segment will show on the first line of the main window (max 34 chars), and describes the application, the second (max 58 chars), shows on the next line, and should describe what the application or file does. The third shows the latest version number of the application in VersionString format, and the fourth is the filename of the downloadable archive on your server.

Note: You can use any punctuation you like within each segment description, except for a double '--', which always marks the end of each segment.

The IPC Call

Check for updates when your application starts:

Many developers may have decided not to use the Versions update checking system, as being perhaps over complicated to set up for their use. Now, with the addition of a single IPC call to Check File at startup, applications can easily check for newer versions of themselves automatically.

To achieve this, the rUpdateInfo resource must be present in the applications resource fork, and a minimally formatted .inf file, along with a downloadable archive, be placed on the web server passed in the rUpdateInfo resource. Marinetti must also be installed and connected to the Internet, though the application itself does not need to start or use TCP/IP itself.

How the IPC call works:

When the IPC call is made, the application passes its Resource ID, and a configurable Input flag to the IPC call. At its simplest, that is all the application needs to do, and Check File will do the rest. The Input flag gives the application choices as to will happen during the call, so the application is always in full control. Please refer to the descriptions below on how that works.

When Check File is called, it retrieves the pStrings from the rUpdateInfo resource of the application, then attempts to retrieve the related .inf file from the target web server named in that resource. If the .inf file is found, it will be parsed to extract the version number of the latest update that is on the server, and also extract the filename of the latest update. It then compares the version number it found against the current version number of the application itself, and if they are different, will show an Alert with Download, Ignore, and OK buttons. By clicking Ignore or OK, or if the .inf or archive files are not found, or the version on the server is the same, Check File will simply return from the call as if nothing had happened. If the Download button is clicked, a Standard File dialog will open to let the user choose where to save the file, and Check File will then download the archive. An associated .md5 file will also be downloaded if one is found.

Typically, this whole checking process takes no more than a second, and if a download is called for, that may only take a few seconds more.

The Input and Output data blocks:

Please refer to the section above for the rUpdateInfo resource to see how to correctly format that resource, and also how to build the required tag for the .inf file that must also be placed on the server along with the downloadable archive.

The Input DataIn block tells Check File what to do when it has been called:

```
Input flag:
0 = Checks for presence of files, does not show an alert
1 = Displays an alert to say an update was found
2 = Displays the Download option if an update and an archive were found
```

The IPC call:

```
          pea   $6200             tellCFToCheckFile
          pea   $8001             stop after one
          PushLong #CFileName
          PushLong #CFDataIn
          PushLong #CFDataOut
          _SendRequest

CFDataIn anop
         dc    i'2'              parameter count
         dc    i'$0002'          Input flag
         dc    i'0'              this apps ResourceID

CFDataOut anop
         dc    i'0'              recvcount
         dc    i'0'              Foundflag
         dc    i4'0'             Pointer to Version string (pString)
         dc    i4'0'             Pointer to update name (pString)

CFileName  str   'Speccie~Check.File.PIF~Wannop~'
```

On return from the call, the Foundflag reports what it found:

```
Foundflag:
Bits set:
Bit 0 = a matching .inf file was found for the application
Bit 1 = a newer version was found on the server
Bit 2 = the version on the server is the same as the application
Bit 3 = a downloadable update archive was found
Bit 4 = the update archive was not downloaded
Bit 5 = the update archive was downloaded
Bit 6 = the user chose to ignore the update
Bit 15 = general error, no required files were found
```

A pointer to the version string, and the update file name, that were found in the matching .inf file, are also returned if the .inf file was found.

Strategies

By first setting the Input flag to a suitable value, and copying the apps Resource ID to the data block, the calling application has three basic options:

Input flag = 0

Check File will not display an Alerts and on return, the application can check the Foundflag to see which files were found, and whether an update is available.

Input flag = 1

If an update is found, Check File will show an Alert, and report that it has found an update. The user will not be able to download an update if one was found found, and the Foundflag will report whether the user clicked the Ignore button. No alert will be shown if there is no new update available.

Input flag = 0

If an update is found, Check File will show an Alert, and report that it has found an update. The user can choose to Download the update, or click Ignore, or OK, and return. The Foundflag will show whether the user clicked the Ignore button. No alert will be shown if there is no new update available.

In all cases, it is up to the Application to decide what to do after the IPC call. From the Foundflag, it can see if there was an update available, and if so, if the user chose to Ignore it, or just clicked the OK button. If the user chose to Ignore an update, the application may then decide not to make that call again, by setting a flag in its Preferences. This could be whether to let the IPC call be made at Startup, or perhaps make it available as a Menu item choice.

If a matching .inf file was found, even if it was not a newer version, a pString in VersionString format of the archive on the server will be returned, as well as pString holding the name of the update file. If the pStrings are empty, then the Version data and filename were not found in the matching .inf file, or the .inf file itself was not found.

The IPC Call

Check for updates when your application starts:

Many developers may have decided not to use the Versions update checking system, as being perhaps over complicated to set up for their use. Now, with the addition of a single IPC call to Check File at startup, applications can easily check for newer versions of themselves automatically.

To achieve this, the rUpdateInfo resource must be present in the applications resource fork, and a minimally formatted .inf file, along with a downloadable archive, be placed on the web server passed in the rUpdateInfo resource. Marinetti must also be installed and connected to the Internet, though the application itself does not need to start or use TCP/IP itself.

How the IPC call works:

When the IPC call is made, the application passes its Resource ID, and a configurable Input flag to the IPC call. At its simplest, that is all the application needs to do, and Check File will do the rest. The Input flag gives the application choices as to will happen during the call, so the application is always in full control. Please refer to the descriptions below on how that works.

When Check File is called, it retrieves the pStrings from the rUpdateInfo resource of the application, then attempts to retrieve the related .inf file from the target web server named in that resource. If the .inf file is found, it will be parsed to extract the version number of the latest update that is on the server, and also extract the filename of the latest update. It then compares the version number it found against the current version number of the application itself, and if they are different, will show an Alert with Download, Ignore, and OK buttons. By clicking Ignore or OK, or if the .inf or archive files are not found, or the version on the server is the same, Check File will simply return from the call as if nothing had happened. If the Download button is clicked, a Standard File dialog will open to let the user choose where to save the file, and Check File will then download the archive. An associated .md5 file will also be downloaded if one is found.

Typically, this whole checking process takes no more than a second, and if a download is called for, that may only take a few seconds more.

The Input and Output data blocks:

Please refer to the section above for the rUpdateInfo resource to see how to correctly format that resource, and also how to build the required tag for the .inf file that must also be placed on the server along with the downloadable archive.

The Input DataIn block tells Check File what to do when it has been called:

```
Input flag:
0 = Checks for presence of files, does not show an alert
1 = Displays an alert to say an update was found
2 = Displays the Download option if an update and an archive were found
3 = Silently checks for an update, and downloads if one is found
```

The IPC call:

```
         pea   $6200             tellCFToCheckFile
         pea   $8001             stop after one
         PushLong #CFileName
         PushLong #CFDataIn
         PushLong #CFDataOut
         _SendRequest

CFDataIn anop
         dc    i'2'              parameter count = 2 or 3
         dc    i'$0002'          Input flag
         dc    i'0'              this apps ResourceID
         dc    i4'Folder'        path to download folder

CFDataOut anop
         dc    i'0'              recvcount
         dc    i'0'              Foundflag
         dc    i4'0'             Pointer to Version string (pString)
         dc    i4'0'             Pointer to update name (pString)
         dc    i'0'              Returned error of Input flag = 3

CFileName  str   'Speccie~Check.File.PIF~Wannop~'
```

On return from the call, the Foundflag reports what it found:

```
Bits set:
Bit 0 = a matching .inf file was found for the application
Bit 1 = a newer version was found on the server
Bit 2 = the version on the server is the same as the application
Bit 3 = a downloadable update archive was found
Bit 4 = the update archive was not downloaded
Bit 5 = the update archive was downloaded
Bit 6 = the user chose to ignore the update
Bit 7 - file already exists, not downloaded
Bit 14 - GS/OS error during download
Bit 15 = general error, no required files were found
```

A pointer to the version string, and the update file name, that were found in the matching .inf file, are also returned if the .inf file was found.

Strategies

By first setting the Input flag to a suitable value, and copying the apps Resource ID to the data block, the calling application has four options:

Input flag = 0

Check File will not display any Alerts and on return, the application can check the Foundflag to see which files were found, and whether an update is available.

Input flag = 1

If an update is found, Check File will show an Alert, and report that it has found an update. The user will not be able to download an update if one was found found, and the Foundflag will report whether the user clicked the Ignore button. No alert will be shown if there is no new update available.

Input flag = 2

If an update is found, Check File will show an Alert, and report that it has found an update. The user can choose to Download the update, or click Ignore, or OK, and return. The Foundflag will show whether the user clicked the Ignore button. No alert will be shown if there is no new update available.

Input flag = 3

Check File will not display any Alerts, and if an update is found, Check File will silently download the file to the supplied download path. If the file already exists, or the path is invalid, the call will fail, and report any GS/OS error at the end of the DataOut block. It will also set the bits of the Foundflag as appropriate.

In all cases, it is up to the Application to decide what to do after the IPC call. From the Foundflag, it can see if there was an update available, and if so, if the user chose to Ignore it, or just clicked the OK button. If the user chose to Ignore an update, the application may then decide not to make that call again, by setting a flag in its Preferences. This could be whether to let the IPC call be made at Startup, or perhaps make it available as a Menu item choice.

If a matching .inf file was found, even if it was not a newer version, a pString in VersionString format of the archive on the server will be returned, as well as pString holding the name of the update file. If the pStrings are empty, then the Version data and filename were not found in the matching .inf file, or the .inf file itself was not found.

Extras

Help

Online Help for the User menus of **Versions** is available if you have installed the optional !Help! desk accessory, and its Help files, to the Desk.Accs folder.

Problems

Hopefully you will have none, but if you do, and they cannot be answered by reading these notes, please contact me on:

spectrumdaddy@speccie.uk

Other information

Check for the latest version of Marinetti:

http://www.apple2.org/marinetti/

If you do not already know about Spectrum™, please drop by my home pages, and read more. Apart from all the other wonderful things it does, Spectrum™ offers many useful tools for processing files, such as post processing text files that you have received, that may have obstinate formatting.

Spectrum™ is now Freeware, and amongst my other applications, is now available from my web site:

http://speccie.uk

Someone once said to me, 'Spectrum™ does everything!'