Technical notes on using Analog Devices DSPs, processors and development tools
Contact our technical support at processor.support@analog.com and dsptools.support@analog.com
Or visit our on-line resources http://www.analog.com/ee-notes and http://www.analog.com/processors

Using the VisualDSP++® Command-Line Installer

Contributed by Alex Gray *Rev 1 – July 11, 2005*

Introduction

Analog Devices VisualDSP++® 4.0 includes a command-line interface to the installer. It offers advanced features that may be beneficial to users and/or administrators in environments that:

- Demand rigorous version control
- Automated installation/build/test harnesses
- The maintenance of multiple update/patch levels on the same PC

It is expected that only a minority of customers will use these features.

This document is relevant to VisualDSP++ 4.0 and later. This installer is not available for earlier versions of VisualDSP++.

Overview and Command-Line Usage

For most users, software installation is a simple matter of selecting the default installation options offered by a graphical installation utility that ships with the product. Installer tools may provide minimal customizations, such as allowing the user to specify the installation directory. The VisualDSP++ installer is the recommended tool for performing installations for the majority of customers. However, some customers have a development process that demands more control over the installation of VisualDSP++. This document details the command-line of the installer and describes several advanced installation options that may be beneficial to some users' workflow.

Install_CL

The command-line installer is called `Install_CL`. `Setup` is the graphical wrapper around `Install_CL`. `Setup` itself does not modify the user's system; it shells to `Install_CL` for all "write" actively. Thus, nothing can be done with `Setup` that cannot be done with `Install_CL`. `Install_CL` generates a log as it performs actions. This log is written to standard out (`stdout`) and may be captured with the command prompt's "`>`" operator. Fatal errors and a few high-level warnings are written to both `stdout` and standard error (`stderr`).

Table 3 in the Appendix summarizes the command-line summary of `Install_CL`. Several of these switches are mandatory (Table 1).

Switch	Description
`-action <value>`	Specifies the fundamental action to be performed by the installer. The `new`, `update`, and `remove` actions are the most basic operations. `insitu` is detailed later in this document. `drysys32` and `sys32` are primarily for the GUI's benefits, allowing the GUI to update the PC's `\Windows\System32` files as an independent step, before placing application files. Command-line use of these options is not expected.
`-dir <value>`	The directory in which files are to be installed.
`-hklm <value>`	The HKEY_LOCAL_MACHINE registry key to be used. This location be must unique for each installation of VisualDSP++, otherwise installations will "cross-talk" with each other and an uninstallation may damage other installations.
`-manifest <value>`	Indicates the "manifest" file to drive the installation's actions. The manifest is an XML-format file that defines all files associated with an installation and special attributes associated with any file (registry settings, start menu items, etc.). Although manifests are text files and are easily modifiable, modification of a manifest file is highly discouraged.  Any version/variant of VisualDSP++ installed with a modified manifest will not be support by Analog Devices.
`-stage <value>`	The directory in which files will be installed *from*. This switch is not required when removing an installation.

Table 1. Mandatory Command-Line Switches for Install_CL

`Install_CL` issues a return code indicating success or failure. See Listing 1 in the Appendix for summary.

Advanced Switches

Advanced command-line switches are described below. Any other switch not discussed here either has obvious usage or is not expected to use used by the customer base.

Switch	Description
`-scrub-hku`	Causes the installer to remove any registry keys that match the pattern `HKEY_USERS/*/$HKLM`, where *`$HKLM`* is the value specified by the `-hklm` switch. Windows' `HKEY_USERS` hive is used to persist settings on a *per-user* basis (window layouts, most-recently used files lists, etc.)[1]. This switch is intended for product removal to ensure that the PC is left in as clean a state as possible. It also prevents per-user settings from "reappearing" if the product is later reinstalled.
`-scrub-inf`	Causes the installer to search the `\Windows\INF` directory for the previous version of emulator and/or EZ-KIT Lite INF files, and moves them to the Recycling Bin. This occurs *before* the installation of new INF files. This ensures that *only* the latest INF files are present. This resolves some issues with Windows 2000's tendency to not always select the most up-to-date INF file when multiple INF files are available for the same hardware. By default, `Setup` uses the `-scrub_inf` switch; it is recommend that command-line users use this switch.
`-use-date`	Indentifies the `HKEY_LOCAL_MACHINE` registry key to use. It is imperative that this location be unique for each installation of VisualDSP++. Otherwise, installations will "cross-talk" with each other and one uninstallation may likely damage other installations.

Table 2. Advanced Command-line Switches for Install_CL

[1] HKEY_LOCAL_MACHINE is used to persist settings on a *per-machine* (installation) basis.

Example 1: Simple Product Installation

Selecting the default options from `Setup` is equivalent to the following command-line[2], with the working directory being the directory in which `Install_CL` is found.

```
Install_CL   -action new
        -dir "C:\Program Files\Analog Devices\VisualDSP 4.0"
        -hklm "Software\Analog Devices\VisualDSP++ 4.0"
        -manifest "manifest.xml"
        -scrub-inf
        -stage "Stage"
        -startmenu "C:\Documents and Settings\All Users\
                    Start Menu\Programs\Analog Devices\VisualDSP++ 4.0"
        -use-date
```

Note that the `-dir`, `-hklm`, and `-startmenu` switches are specific to the *destination* of the installation. When installing multiple versions of VisualDSP++, use unique values for these switches to avoid cross-talk between discrete installations of the product.

Example 2: Simple Product Update

An *update* lays an incremental installation on top of an existing installation. Updates are distributed as a single, self-extracting executable file (you may have to rename the download with an EXE file extension).

Do not extract these files to the application directory. Place them in a temporary directory.

After the extraction (in this example, `C:\Temp\VDSP_Update\Stage`), run the following command-line to update the existing installation. The working directory is the base of the installed VisualDSP++ directory (note the "." in the `-dir` switch)

```
Install_CL   -action update
        -dir "."
        -hklm "Software\Analog Devices\VisualDSP++ 4.0"
        -manifest "C:\Temp\VDSP_Update\manifest.xml"
        -scrub-inf
        -stage "C:\Temp\VDSP_Update\Stage"
        -startmenu "C:\Documents and Settings\All Users\
                    Start Menu\Programs\Analog Devices\VisualDSP++ 4.0"
        -use-date
```

Example 3: Product Removal (Uninstallation)

A *removal* uninstalls the VisualDSP++ product. The following command-line will uninstall the product. The working directory is the VisualDSP++ directory.

[2] Throughout this document, carriage returns are inserted for readability. In reality, this is one long command-line.

```
Install_CL   -action remove
        -dir "."
        -hklm "Software\Analog Devices\VisualDSP++ 4.0"
        -manifest "manifest.xml"
        -startmenu "C:\Documents and Settings\All Users\
                        Start Menu\Programs\Analog Devices\VisualDSP++ 4.0"
```

This command removes files listed in the manifest, *including files later modified by the user*. If you have modified example code that you wish to retain, ensure that you have copied these files outside of the VisualDSP++ installation directory.

This command removes files listed in the manifest, *including files later modified by the user*. If you have modified example code that you wish to retain, ensure that you have copied these files outside of the VisualDSP++ installation directory.

Removal will not remove these files:

- Files placed in the `\Windows\System32` directory. These files are shared among multiple applications and cannot be removed.
- Files not listed in the manifest. These include your license file and any `.DOJ`, `.DXE`, etc. files that may have resulted from building example programs that ship with VisualDSP++.
- `Install_CL` and the manifest. These files are locked by the operating system during removal. Thus they cannot be deleted.

Example 4: Cloning an Existing Install and then Updating that Clone

A clone of an installation creates a new installation of a product from an existing installation, rather than from a CD or web software distribution. The use of clones allows you to maintain multiple versions of VisualDSP++ on the same PC at different update levels. Clones provide a risk-free way to "test" new updates or patches[3]. In this example, the working directory is the directory in which the existing installation of VisualDSP++ resides. The command line is very similar to a new installation.

```
Install_CL   -action new
        -dir "C:\Program Files\Analog Devices\VisualDSP 4.0 Clone"
        -hklm "Software\Analog Devices\VisualDSP++ 4.0 Clone"
        -manifest "manifest.xml"
        -scrub-inf
        -stage "."
        -startmenu "C:\Documents and Settings\All Users\
                        Start Menu\Programs\Analog Devices\VisualDSP++ 4.0 Clone"
        -use-date
```

[3] "Uninstalling" or "rolling back" an update is not supported by the installer. The ability to clone and update the clone only offers similar functionality.

After performing the clone action, the PC will have two identical installations. You can then update the clone installation using the procedure detailed above (with the working directory being that of the *clone*). Updating the clone in this manner leaves the original installation unchanged.

As discussed throughout this document, give each clone a unique `–dir`, `-hklm`, and `–startmenu` value to avoid cross-talk between installations.

In Situ Installations

In situ is Latin for "in place" or "in the original position". An in situ installation refers to a VisualDSP++ installation that opperates on a set of files that already exist on the target machine. It is important to understand that in Microsoft Windows software, an "installation" is more than a collection of files. A number of "other" steps must be performed to make the application "known" to Windows and the user. Other steps include creating static registry entries, self-registering components with the operating system, creating a Start Menu item, creating install-time `.BAT` and `.INI` files, and installing device drivers. In an in situ installation, these "other" steps only are performed; no application files are installed/copied.

An in situ installation supports scenarios in which VisualDSP++ is placed under source code control such as CVS[4] and then later retrieved onto a different machine. Once placed onto the new machine, an in situ installation performs the "other" steps discussed above. An in situ installation, in conjunction with a source code control tool like CVS provides robust version control for your development tools, address two requirements:

- To ensure that your organization uses the same build tools consistently
- To more easily recreate historic build environments when supporting legacy products

The following two examples demonstrate how to create such an environment. Basic usage and nomenclature of CVS is required to take full advantage of these examples.

Note that many VisualDSP++ licenses are node-locked, meaning that they will work only on an individual machine. Two approaches to working with VisualDSP++ licensing is to use floating licenses or to create node-locked licenses for each machine and to commit them to CVS with machine-specific tags.

Example 5: Creating, Committing, and Tagging a CVS Repository and then Performing an In Situ Installation from It

The first step in this example is to create the repository in CVS and to commit the VisualDSP++ application files to this repository. To do this, follow these steps:

1. Check out the CVS repository to a local directory (i.e., `e:\temp\cvsRepository`):

```
cvs checkout –d "e:\temp\cvsRepository" <path to repository>
```

[4] CVS is cited throughout this document, and all given examples use CVS. However, the principles involved can be applied to any number of source code control applications.

2. Create a directory in which to place VisualDSP++. (`e:\temp\cvsRepository\VisualDSP40`):

```
mkdir "e:\temp\cvsRespository\VisualDSP40"
```

3. "cd" to `e:\temp\cvsRepository\` and add it to the repository:

```
cvs add "VisualDSP40"
```

4. Extract the contents of the VisualDSP++ `Stage.zip` file to `e:\temp\cvsRespository\VisualDSP40`. We must now recursively add and commit all these file to the cvs respository. CVS has an "import" command that allows you to place an entire hierarchy under CVS. However, an alternative, albeit more labor-intensive, strategy is suggested here, as it has proven more flexible when managing multiple revisions of a tool. The proposed methodology is a two-stage process. Files must be added and then committed to CVS. Unfortunately, CVS does not include a mechanism for adding or committing an entire file hierarchy with a single (manageable) command line. Files and directories are added by specifying them individually on the command line. Since VisualDSP++ has over 1000 files in its distribution, this is impractical to do manually. Another complication is the proper handling of text files versus binary files. A CVS tag is also recommended to distinguish this installation from future installations placed into the same repository. Listing 2, located in the Appendix, shows a script that issues commands to add, commit, and tag the entire VisualDSP++ file hierarchy. For clarity, this script is named, `listing2.py`. The output of this script can be written to a temporary batch that is then executed[5].
5. Copy `fig3.py` to `e:\temp\cvsRepository\VisualDSP40` and execute it:

```
python listing2.py –Tag "VDSP40" –Description "Initial Check-in" > cvs_add.bat
cvs_add.bat
```

VisualDSP++ is now available from CVS.

The second step in this example is to check out the distribution from CVS and perform an in situ installation on the checkout. This get can be performed on the same computer as used in the first step (perhaps in a different directory), or on a different computer altogether. First, perform a CVS checkout using the tag established earlier:

```
cvs –checkout –d "c:\program files\VisualDSP40" <path to repository>
```

[5] Alternatively, if Cygwin tools are available on the PC, the output of this program can simply be piped to `sh`.

The VisualDSP++ application files are now available on the target system. An in situ installation can be performed at this time with the following command line (the working directory is the base VisualDSP++ directory in which `Install_CL` is found)[6]:

```
Install_CL   -action insitu
         -dir ".”
         -hklm "Software\Analog Devices\VisualDSP++ 4.0”
         -manifest "manifest.xml”
         -scrub-inf
         -stage ".”
         -startmenu "C:\Documents and Settings\All Users\
                        Start Menu\Programs\Analog Devices\VisualDSP++ 4.0”
         -use-date
```

Note that both `-dir` and `-stage` are both "`.`", indicating the current directory. Otherwise, the command-line is very similar to the "new" action described earlier in this document[7]. VisualDSP++ is now ready to be used from its new location.

Example 6: Committing and Tagging an Update Into the Repository

This example assumes that a repository has been created as described in the pervious example. In this example, an update to VisualDSP++ has been released and is to be placed into the CVS repository.

1. Perform a CVS get to place the "baseline" release of VisualDSP++ onto the machine being worked on:

   ```
   cvs checkout -d "e:\temp\VisualDSP40" <path to repository>
   ```

2. Perform an in situ installation and apply an update. You can apply a downloaded update (`.vdu` file) by launching the installer from the Start Menu and choosing `Apply a downloaded Update`. The updated staging area is then committed to CVS. Similar to the commitment of the initial installation, this can be cumbersome to do manually, so the Listing 1 script again is used:

   ```
   python listing2.py -Tag "VDSP40-MAR05" -Description "March Upd" >cvs_upd.bat
   cvs_upd.bat
   ```

3. The VisualDSP++ Update is now available in CVS with the `VDSP40-MAR05` tag.
4. Retrieve VisualDSP++, along with the update, from CVS (using this tag):

   ```
   cvs update -r VDSP40-MAR05
   ```

[6] An in situ installation may be performed via the GUI by double-clicking `Setup.exe` in the Windows Explorer (Start Menu items are, of course, as of yet unavailable).

[7] `-use-date` is meaningless in this contexts, but is presented here for consistency.

Note that the baseline installation is still available by performing a get on the original `VDSP40` tag.

```
cvs update -r VDSP40
```

A Wrinkle with In Situ Installation: Installation of OS-Specific Files

Complications may arise when committing an application distribution to CVS. If the distribution contains any files that are specific to a particular operating system, care must be taken when working in a heterogeneous environment where multiple versions of Windows are being run. Files that are inappropriate for a particular OS must not be placed on a target system running that OS. Conversely, a CVS repository may be incomplete if it was created from a product installation in which certain files were withheld due to the operating system being utilized on the system that created the repository.

As of the time of this writing, one file in the VisualDSP++ distribution falls into this category[8]. `gdiplus.dll`, a Microsoft file available in `Stage.zip` of the baseline VisualDSP++ distribution, is to be installed on Windows 2000 hosts, but not on Windows XP hosts. Analog Devices is prohibited from distributing this file under Windows XP and `Install_CL` will not do so. Care must be taken if a staging area is created under Windows 2000, but then retrieved under Windows XP. Do not place `gdiplus.dll` on the Windows XP host.

Use of a mixed Windows 2000/XP environment is discouraged for this reason. Alternatively, this file may be special-cased through tagging and multiple gets from CVS.

[8] The possible impact of Windows Longhorn is unknown as of the time of this writing.

Appendix

Switch	Description
-@ <file>	Places the contents of 'file' at the current position in the command line.
-action <val>	Specifies the action to be performed. Valid values are: drysys32, sys32, new, update, remove, and insitu.
-dir <dir>	Specifies the directory in which to install files.
-file-log <file>	Specifies a file name in which to write log information. STDOUT and STDERR are reserved to specify these files.
-file-mon <file>	Specifies a file name in which to write monitor information. STDOUT and STDERR are reserved to specify these files.
-format <val>	Specifies the output format the installer is to emit. Valid values are: text (default) and gui.
-help	Displays switch descriptions and exits without performing any actions.
-hklm <val>	Specifies the HKLM root to use in the registry.
-license <file>	Specifies an existing license.dat file to copy into the installed application.
-manifest <file>	Specifies the manifest file to drive the application's actions.
-no-space-check	Instructs the installer to not check for sufficient disk space before performing an -action new.
-read-only-dir	Specifies that the installation directory is read-only and that no files are to be written to or removed from it. Only valid with -action insitu and -action remove.
-reg-com <val>	Specifies the registered company for this installation.
-reg-own <val>	Specifies the registered owner of this installation.
-scrub-hku	During an "-action remove" (only), purges HKU application settings for all users.
-scrub-inf	Identifies and deletes .INF/.PFN files in WINDOWS\INF before installing current drivers. Deleted files are moved to the Recycling Bin.
-stage <dir|file>	Specifies the directory or .ZIP file in which the application files to be installed are staged.
-startmenu <dir>	Specifies the Start Menu root to use. If not specified, no Start Menu item is created.
-use-date	Instructs the installer to use a file's timestamp when comparing existing files to files to be installed. If not specified, version information only is consulted.
-verbosity <val>	Controls the level of feedback. Valid values are: silent, terse, verbose (default) and debug.

Table 3. Command-Line Summary for Install_CL

```
/* ********************************************************************************
 *
 * Copyright (c) 2005 Analog Devices Inc.  All rights reserved.
 *
 * ********************************************************************************/
typedef enum tError
{
    ERROR_NOERROR = 0,
    ERROR_UNKNOWN = 100,
    ERROR_UNIMPLEMENTED,
    ERROR_COMMAND_LINE_PARSE,
    ERROR_OS_NOT_SUPPORT,
    ERROR_OS_PATH,
    ERROR_SYSTEM_FILES,
    ERROR_ADMIN_PRIV,
    ERROR_IE_VERSION,
    ERROR_SYSTEM_FILE_DRY_UPDATE,
    ERROR_REBOOT_FOR_SYSTEM_FILE,
    ERROR_VERSION_INFO,
    ERROR_HKLM_READ,
    ERROR_HKLM_WRITE,
    ERROR_INI_READ,
    ERROR_INI_WRITE,
    ERROR_XML,
    ERROR_XML_NO_SUCH_ITEM,
    ERROR_FILE_READ,
    ERROR_FILE_WRITE,
    ERROR_FILE_SELFREGISTER,
    ERROR_FILE_SELFUNREGISTER,
    ERROR_REGISTRY_READ,
    ERROR_REGISTRY_WRITE,
    ERROR_SHORTCUT,
    ERROR_INF_COPY,
    ERROR_INF_DELETE,
    ERROR_SYS_READ,
    ERROR_SYS_WRITE,
    ERROR_SYS_DELETE,
    ERROR_OS_DISK_SPACE
};

typedef struct
{
    tError eError;
    const char *pszMsg;
} tErrorMsgMap;

const tErrorMsgMap vErrorMsgs[] =
{
    { ERROR_NOERROR,
                "Operation successful." },
    { ERROR_UNKNOWN,
        "An unknown or non-specific error occurred." },
    { ERROR_UNIMPLEMENTED,
        "This functionality is not yet implemented." },
    { ERROR_COMMAND_LINE_PARSE,
        "Error processing command line arguments." },
    { ERROR_OS_NOT_SUPPORT,
        "This operating system is not supported.  Must be running Windows 2000, \
```

```
        XP, or later." },
    { ERROR_OS_PATH,
        "Error determining the Windows installation path." },
    { ERROR_SYSTEM_FILES,
        "Error processing Windows systems files to update/register." },
    { ERROR_ADMIN_PRIV,
        "Insufficient permissions.  You must be an Administrator to
install/uninstall this software." },
    { ERROR_IE_VERSION,
        "The installed version of Internet Explorer does not meet the minimum \
        requirements." },
    { ERROR_SYSTEM_FILE_DRY_UPDATE,
        "One or more Windows system files must be updated." },
    { ERROR_REBOOT_FOR_SYSTEM_FILE,
        "A reboot must be performed before the installation can proceed further."},
    { ERROR_VERSION_INFO,
        "An error occurred processing file version information." },
    { ERROR_HKLM_READ,
        "An error occurred reading from HKEY_LOCAL_MACHINE." },
    { ERROR_HKLM_WRITE,
        "An error occurred writing to HKEY_LOCAL_MACHINE." },
    { ERROR_INI_READ,
        "An error occurred reading from VisualDSP.ini." },
    { ERROR_INI_WRITE,
        "An error occurred writing to VisualDSP.ini." },
    { ERROR_XML,
        "An XML-related error occurred." },
    { ERROR_XML_NO_SUCH_ITEM,
        "This XML item index does not exist." },
    { ERROR_FILE_READ,
        "Error reading or opening a file." },
    { ERROR_FILE_WRITE,
        "Error writing or creating a file." },
    { ERROR_FILE_SELFREGISTER,
        "Error registering a file with Windows." },
    { ERROR_FILE_SELFUNREGISTER,
        "Error unregistering a file with Windows." },
    { ERROR_REGISTRY_READ,
        "Failed to read from the registry." },
    { ERROR_REGISTRY_WRITE,
        "Failed to write to the registry.  Are you an Administrator?" },
    { ERROR_SHORTCUT,
        "Error creating a shortcut (.lnk file)." },
    { ERROR_INF_COPY,
        "An error occurred installing an .INF file." },
    { ERROR_INF_DELETE,
        "An error occurred removing an .INF or .PNF file." },
    { ERROR_SYS_READ,
        "An error occurred reading device driver information." },
    { ERROR_SYS_WRITE,
        "An error occurred writing device driver information." },
    { ERROR_SYS_DELETE,
        "An error occurred deleting device driver information." },
    { ERROR_OS_DISK_SPACE,
        "Insufficient disk space for this installation." },
    { ERROR_ZIP_READ,
        "An error occurred reading from a compressed archive" },
    { ERROR_ZIP_PARSE,
```

```
        "An error occurred parsing a compressed archive." },
    { ERROR_ZIP_EXTRACT,
        "An error occurred extracting a file from a compressed archive." },
    { ERROR_LICENSE,
        "A license-related error occurred." },
    { ERROR_COM,
        "A COM-related error occurred." }
};
```

Listing 1. Return Code Summary of Install_CL (C source code defining the return codes and mapping these return codes to text messages. Permission is granted to use this code in your applications.)

```
/* ********************************************************************************
 *
 * Copyright (c) 2005 Analog Devices Inc.  All rights reserved.
 *
 * ********************************************************************************/

#################################################################################################
#    Global Variables.
#
#################################################################################################
g_Description = ""
g_Tag = ""
g_BinaryExtensions =
[".exe",".dxe",".dll",".doj",".dlb",".wav",".chm",".doc",".gif",".aps",".opt",".pdf",".da3",".das",
 ".sys",".jpg"]


#################################################################################################
#    ParseCommandLine: Parses the command line for -Tag and -Description.  The user does
#    not have these required switches, you'll get a warning and a message on how to correct
#    the commandline.
#################################################################################################
def ParseCommandLine():
     global g_Tag
     global g_Description
     import sys
     import string
     continueParse = False;
     tag = False;
     descript = False;

     for arg in sys.argv:
          if (string.lower(arg) == string.lower("-Tag")):
               tag = True;
               continueParse = True;
          else:
               if (string.lower(arg) == string.lower("-Description")):
                    descript = True;
                    continueParse = True;

          if (continueParse == False):
               if (tag == True):
                    tag = False;
                    g_Tag = arg;
               if (descript == True):
                    descript = False
                    g_Description = arg;
          else:
               continueParse = False;

     if (g_Tag == "" or g_Description == ""):
          print( "" )
          print("***Error in the command line arguements!")
          print("Please use -Tag <TagName> and -Description <Description>")
```

```
          print("Example: python.exe fig3.py -Tag MyTag -Description \"My Description\"")
          print( "" )
          return -1

##########################################################################################################
#    WalkDirectory: Generic function to walk through a directory.  It calls the function
#    CreateCVSCommandLine for every directory in the tree
##########################################################################################################

def WalkDirectory(funcName):
     import os
     os.path.walk(".",funcName, None)

##########################################################################################################
#    CreateCVSCommandLine:  For every file and directory this
#    function will be called.  It will output to stdout the commandline
#    that should be captured in a .bat file
##########################################################################################################

def CreateCVSCommandLine(arg, dirname, filenames):
     import string
     import os
     switch = ""
     global g_BinaryExtensions

     for filename in filenames:
          fullpath = os.path.join(dirname,filename)
          # Let's make everything lowercase, since we'll be doing a lot of string comparison:

          if ( os.path.isfile(fullpath) ):
               # Ignore the CVS hidden directories and files:
               # The last directory cannot be "cvs":
               dirFilePair = os.path.split ( fullpath )
               dirFilePair = os.path.split ( dirFilePair[0] )
               if ( dirFilePair[1] != "cvs"):
                    # If it is a file, determine if the file is a binary or text file.
                    extension = string.lower(filename[len(filename)-4:len(filename)])
                    for el in g_BinaryExtensions:
                           if ( extension == string.lower(el) ):
                                  switch = "-kb "
                                  break
                    # "unicld.dat" and "unihlp.dat" are the only .dat files that are binary.
                    if ( string.lower(filename) == "unicld.dat"
                    or string.lower(filename) == "unihlp.dat" ):
                           switch = "-kb "

                    # Now that we know if it is binary or text file, print the CVS commandline
                    print "cvs add " + str(switch) + "\"" + str(fullpath) + "\""
                    print "cvs commit -m " + "\"" + str(g_Description) + "\"" + " \"" + str(fullpath)
                           + "\""
                    print "cvs tag -F " + str(g_Tag) + " " + "\"" + str(fullpath) + "\""
          else:
               # If it is a directory, add it to the repository:
               # Ignore the CVS hidden directories:
               # The last directory cannot be "cvs":
               dirFilePair = os.path.split ( fullpath )
               if (string.lower(dirFilePair[1]) != "cvs"):
                    print "cvs add " + "\""+ str(fullpath) + "\""


##########################################################################################################
#    Run the Script!
#
##########################################################################################################

if ( ParseCommandLine() != -1 ):
     WalkDirectory(CreateCVSCommandLine)
```

Listing 2. Python Script to Add and Commit VisualDSP++ to CVS. (This Script is Called listing2.py Throughout this Document.)

Document History

Revision	Description
Rev 1 – July 11, 2005 *by Alex Gray*	Initial Release