



Getting Started With intent[®] on Windows[®]

1. INTRODUCTION.....	4
1.1. OVERVIEW	4
1.2. INVOKING INTENT	5
1.3. UPGRADES AND PATCHES	6
1.4. A NOTE ABOUT EXAMPLE CODE	6
1.5. A NOTE ON INTERFACING TO NATIVE FILE SYSTEMS	7
2. INTENT AND WINDOWS[®].....	8
2.1. ELATE.EXE	8
2.2. LOCATION.....	8
2.3. INTENT PATH	9
2.4. IMAGEFILE PATH.....	9
2.5. SHELL COMMAND.....	9
2.6. FILE NAME CONVERSION	9
2.7. CREATING FILE TYPES	9
2.7.1. Example 1	9
2.7.2. Example 2	9
2.7.3. Example 3.....	10
2.8. START-UP DIRECTORY	10
2.9. LOADING STRING.....	10
2.10. MEMORY.....	10
2.11. TRACE FLAGS	10
2.12. LOG FILE.....	11
2.13. INTERACTIVE.....	11
2.14. INPUT FILE	11
2.15. OUTPUT FILE.....	11
3. CREATING AN INTENT SHORT CUT	12
3.1. EXAMPLE FOR ELATE.EXE	12
3.2. EXAMPLE FOR ELATECON.EXE	12
4. INVOKING ELATECON.....	13
4.1. RESTRICTIONS.....	13
5. DEBUGGING INTENT	14
6. THE INTENT BOX	15
6.1. THE FILE MENU	15
6.2. THE EDIT MENU.....	15
6.3. THE VIEW MENU.....	16
6.4. THE HELP MENU.....	16
7. INTENT HELP FUNCTIONS.....	17

8. THE INTENT SHELL - AN INTRODUCTION	18
8.1. COMMAND CONVENTIONS	18
8.2. MOST COMMONLY USED SHELL COMMANDS	18
8.3. SIMPLE COMMANDS	19
9. DIRECTORY STRUCTURE.....	20
9.1. MORE SIMPLE COMMANDS	20
10. THE SYSBUILD UTILITY	21
11. USING THE SYSGEN UTILITY	22
12. USING THE C /C++ COMPILER	23
13. USING THE VP ASSEMBLER	24
14. THE TRANSLATION PROCEDURE	25
15. USING INTENT AND JAVA™ TECHNOLOGY.....	26
16. INTRODUCTION TO THE INTENT MULTIMEDIA LIBRARIES	27
16.1. SIMPLE CUSTOMISATION	27
17. DOCUMENTATION	28
18. TRAINING	29
18.1. INTRODUCTION TO INTENT.....	29
18.2. INTRODUCTION TO VP PROGRAMMING	29

1. Introduction

intent[®] is a revolutionary content platform that can be used on anything from mobile phones to high end servers. This complete portability is brought about by means of a Virtual Processor to which all Elate[®] programs are written, without regard for the hardware platform.

Once a simple program called the Translator is written by Tao for the new architecture, all application software and libraries written for the virtual processor can run immediately upon any platform or processor, with absolutely no need for any further rewriting or recompilation. As a result of this intent constitutes an effective content layer across the very broadest range of processors – RISC and CISC for example.

Its unique translation technology takes the Virtual Processor byte code and translates it into the native code of the target processor. Normally, translation into efficient native code only takes place when loaded from store, (e.g. disk or network), rather than at compile or link time. The translator knows which processor it is running on and can generate the appropriate code. Programs for intent can currently be written in VP code, the assembler language of the virtual processor, 'C', C++ or Java[™]. Demonstration programs in VP, the Java language and C are available in the *demo/example* directory. More information on these can be found later in the document.

intent incorporates the following features:

- **Total Portability**
- **Minimal Footprint**
- **Vendor Definable Realtime Kernel**
- **Multitasking and Multithreading**
- **Parallel and Heterogeneous Processing**
- **A Unique Java[™] Implementation**

In particular, the intent media system has been designed to drive interactive and other multimedia content across the broad range of wireless and wired digital client devices including low powered, small footprint products. Tao's drawing, windowing, compositing and gadget 2D libraries form the cornerstone of the multimedia toolkit for creating differentiated, highly branded products. The intent system includes a Sun authorised implementation of the Java Virtual Machine and libraries (intent Java[™] Technology Edition) This has been meticulously constructed over many years, incorporating a great many crucial innovations developed within Tao's laboratories by its world-class engineering team, to bring desktop JIT performance to consumer electronics devices whilst carefully constraining the overall footprint. The results are the world's fastest consumer based Java engine and the most compelling multimedia, products of unrivalled excellence.

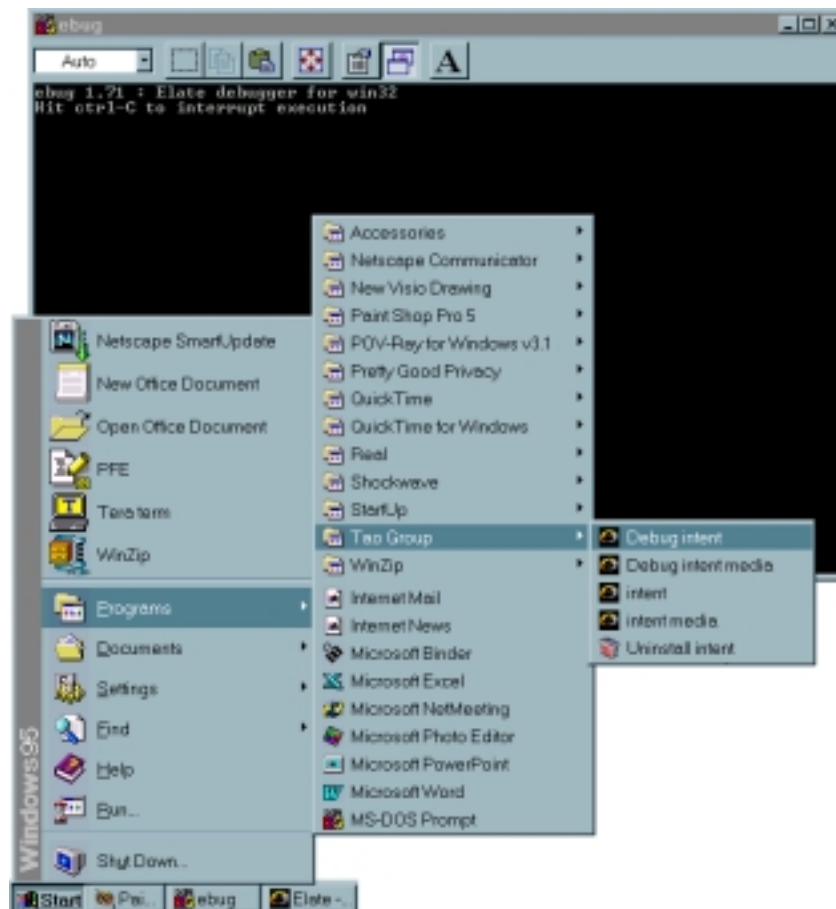
1.1. Overview

This document covers the basic details of installing, configuring, and using the intent content platform, as a development system on Windows[®] 95, 98, NT and Windows 2000. Please note that the vast majority of Windows users will install intent through a simple InstallShield[®] program. The InstallShield for intent will install the release onto the Windows system and create the appropriate shortcuts. However, if this is not the case for any reason, or users wish to fine tune their settings, the information presented in this document may be of assistance.

To install intent via InstallShield, please refer to the more detailed instructions provided with each distribution. InstallShield will present each user with a series of configuration options, which may include such items as requesting consent to the license agreement, prompting for destination directory, and so on. The individual choice of options will determine the exact nature of the installation. Multiple versions of intent can be installed on the same machine. InstallShield installs a set of pre-configured systems, called images. When intent is invoked, the most appropriate image should be selected. The images are as following:

- *develop* – This is suitable for both text and graphics applications
- *develop(t)* – as above, with checking translator (adds additional assertions to assist debugging)

These contain minimal PersonalJava classes. Custom images, including those using the full PersonalJava specification, can be created through the sysbuild utility. See the later section on this utility for more information.



An example of an installed system

1.2. Invoking intent

The following scripts are provided:

- *intent* – Invokes the *develop* image to start shell prompt in a text window
- *intent_debug* – Invokes the *develop* image, with a text window, using *fbugwin*
- *intent_media* – Invokes the *develop* image, to start a graphic window
- *intent_media_debug* – Invokes the *develop* image, with a graphic window, using *fbugwin*.

intent includes the following scripts, to invoke Java™ applications and applets:

- *intent_java_stdio* – Invokes a Java stdio application
- *intent_java* – Run a Java text or graphics application without stdin and stdout
- *intent_applet* – Run a Java applet embedded in a html page without stdin and stdout

These files are installed into the same directory that *intent* has been installed into. It will be necessary to change directory to this location, and copy your Java application to this directory, in order to use the batch files.

For an example Java application:

```
cd \intent
intent_java -classpath example.jar -Dproperty=true example.Main
```

In this case, *example.jar* will be located in the directory where *intent* was installed. The classes can also be extracted into the file system, starting from the directory where *intent* was installed.

For an example Java applet:

```
intent_applet example.html
```

In this case, *example.html* will be located in the directory where *intent* was installed. The applet classes must also be in this directory.

intent comes with a number of Java language example programs for the new user to begin with, and these can be found in the directory *demo/example/j*. simple text-based Java programs such as 'Hello World' will need to be run from the *intent_java_stdio* batch file.

Further technical information is available, either in the form of online help or as printed user guides.

1.3. Upgrades and Patches

There are three types of *intent* patches. Official patches, unofficial patches and platform releases. Official patches are accumulative and have been through full verification, they will only work on officially patched versions of *intent*. Unofficial patches contain one off fixes or updates and have not been through full verification; they are intended as a quick fix only. Once an unofficial patch has been applied, official patches cannot be installed. Platform releases contain the platform specific tools and configuration files that are needed to build images for that platform. They can only be installed on a specific *intent* release and a particular patch level and above. They can also be installed on unofficially patched *intent* installations.

Please note that in cases where Tao Group has issued unofficial patches to the *intent* media system, this will prevent any further official upgrades from being accepted. Further unofficial patches will still work. It is suggested that users reinstall their original system in combination with an official patch at the point where one becomes available.

1.4. A Note about Example Code

The demonstration source code in *demo/example* and *demo/ave* is built into the *intent* filesystem. In order to access this code, it is necessary to start the *intent* shell (as described later in this document). From an *intent* shell prompt type, as required:

```
ls /demo/ave
```

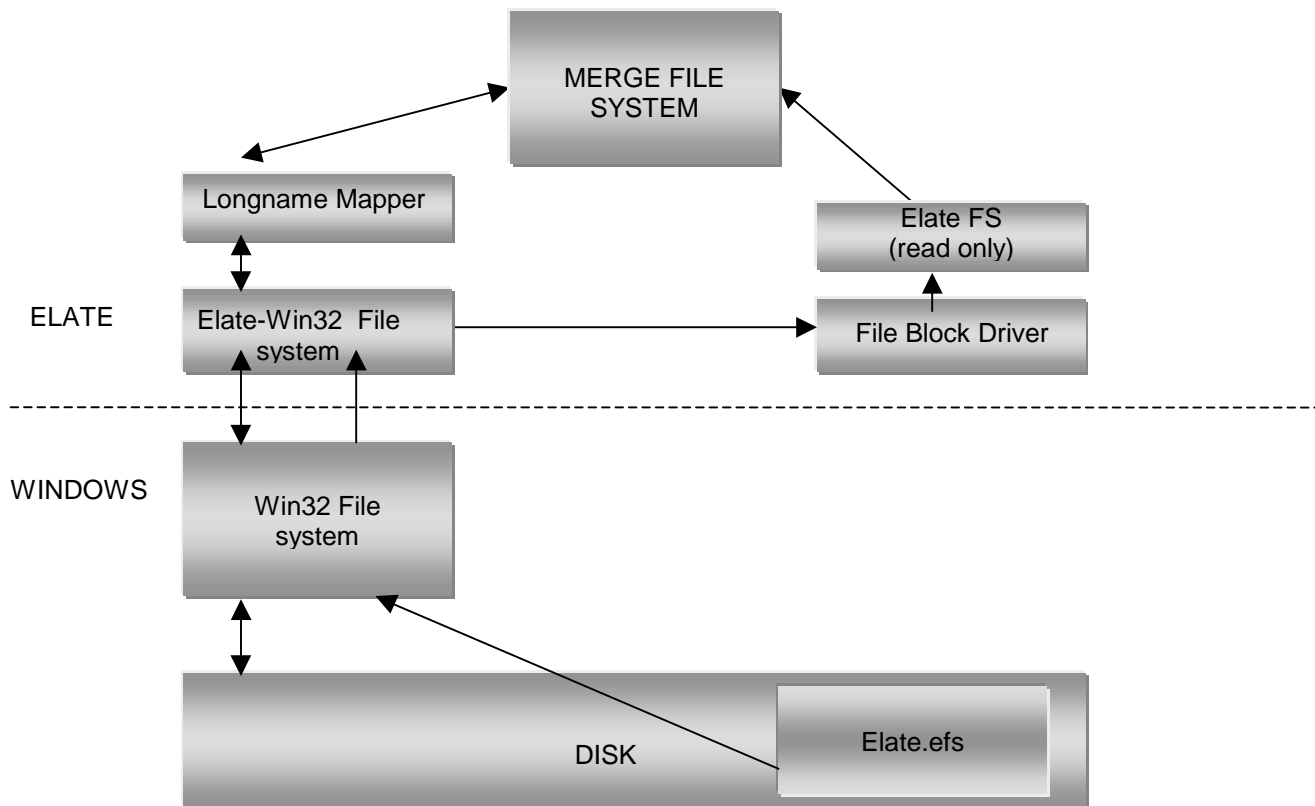
to get a listing of the demo source. Then run:

```
touch/demo/ave/*
```

from the shell, and *intent* will create copies of the demos and sources, making them accessible from the host OS filesystem (e.g in "C:\intent\demo\ave").

1.5. A Note on Interfacing to Native File Systems

The merge file system amalgamates several different filing system mount points into the merge device driver's mount point. If two or more filing systems share directories that have the same path, then the contents of these directories are concatenated together when accessed via the merge file system mount point. If an attempt is made to modify a file within a read-only file system, the merge file system will make a copy of the file on the first writable file system available. This copy is used for all subsequent accesses to the file.



* Elate.efs is a file containing an image understood by the intent file system

Figure 1. The Merge file system

2. intent and Windows[®]

2.1. Elate.exe

Elate.exe and Elatecon.exe are programs for hosting the intent platform (using the real time kernel) on Win32 based environments such as Windows 95[™], 98[™] and Windows NT[™].

Elate.exe is a Windows application, which will load and boot an intent file called an Image File. This can be done simply by clicking on it, after having created the appropriate shortcut (see below). For Windows 95 *Elate.exe* can be found at:

```
sys\platform\win32\elate.exe
```

When the *Elate.exe* program has finished loading, intent will then run inside an “intent Box” in much the same way that DOS[®] can run inside a “Dos Box”, in that a new Window will appear on the screen upon which intent commands can be entered.

The Image File contains a boot system and a set of interface tools to the Win32 host to provide basic timer, display, keyboard, file system and mouse services. The Image File is loaded and run as a Win32 thread of execution within the intent process.

Elatecon.exe is a Windows Console Application that does not have a visible window, and performs input/output using standard input and standard output.

The Image File contains a boot system and a set of interface tools to the Win32 host to provide basic timer, display, keyboard, filesystem and mouse services. The Image File is loaded and run as a Win32 thread of execution within the intent process. The Elate.exe title bar contains the following information:

intent	The product name
[<type><inst>]	This indicates the window type (Text or Graphic) and an instance number that is useful for identifying the Text and Graphic window pair that comprise a particular session
<imagepath>	The pathname of the image file

2.2. Location

intent consists of an Executable (Elate.exe or Elatecon.exe), and a number of Dynamic Link Libraries (DLLs).

intent can be placed anywhere on a hard disk or network so long as the DLLs can be found by the normal Windows DLL search mechanism. This means the DLLs can be placed in one of the following locations:

- ◆ In the same directory as the Elate executable
- ◆ In the \WINDOWS directory or \WINDOWS\SYSTEM directory
- ◆ In one of the directories specified by the PATH environment variable

Normally, the intent executable and DLLs should reside in the sys/platform/win32 directory.

intent requires a command line to tell it where to find the image file.

To invoke intent, the command line is of the form:

```
<intent path> [-Bimagefile path] [-cshellcommand] [-F] [-D[startupdirectory]]  
[-L[loadingstring]] [-Minitialmemorysize] [-Umemoryunit] [-Xmaximummemorysize]  
[-T[traceflags]] [-Klogfile] [-Iinteractive] [<inputfile> [>outputfile]]
```

For any of the options, if it is necessary to enclose the parameter in quotation marks, either double quote (“) or single quote (') may be used.

2.3. intent Path

The `intent` path is simply the full pathname for either `Elate.exe` or `Elatecon.exe`.

2.4. Imagefile Path

`intent`'s `-B` command line switch enables you to tell it where to find the image file. If none is supplied then it looks for `"sys/platform/win32/f*.img"` starting from the current directory, where `f*` depends on the processor.

2.5. Shell Command

The shell command parameter enables the `intent` shell to execute commands specified on the Windows command line. For example to execute the echo commands:

```
echo one
echo two
```

use the `-c` option `-c"echo one;echo two"`. This will execute the commands, and exit `intent`. To run an interactive shell afterwards, add `shell` to the commands `-c"echo one;echo two;shell"`.

2.6. File Name conversion

`intent`'s `-F` command line option enables `intent` to convert Win32 file names into `intent` file names within the shell command option. The file names must be delimited by the `$` character. If the characters `;` follow the initial `$` the extension is removed from the file name before passing to `intent`. A `$` may be inserted by using `$$`. This is normally used to set up associations between `intent` files and `intent`.

2.7. Creating File Types

- ◆ Within Windows Explorer, use the view/options command and select the file types tab.
- ◆ Double-click the My Computer icon.
- ◆ On the View menu, click Options, and then click the File Types tab.
- ◆ To create a new file type, click New Type. To modify the settings for an existing file type, click the type, and then click Edit.
- ◆ Specify a description for the file type. E.g. `intent`
- ◆ Specify the filename extension associated with this type of file. E.g. `class`
- ◆ Click New to define an action for this file type. If you are modifying an existing type, you can click the command in the Action box that you want to modify, and then click Edit.
- ◆ Specify the action that you want to define, such as Open (used when a file is double-clicked) or Java (which will appear on the right-click menu).
- ◆ Specify the command that should run to complete this action including the options. i.e. the path to `Elate.exe`
- ◆ Click "OK"
- ◆ Click "Close"
- ◆ An extension association or file type will be created

2.7.1. Example 1

If `intent` is installed in `C:\INTENT` then an association for `.00` which contains an `intent` main tool would need the "application used to perform action" line to be:

```
C:\INTENT\SYS\PLATFORM\WIN32\ELATE.EXE -f -c"$;f%1$"
```

2.7.2. Example 2

If `intent` is installed in `C:\INTENT` then an association for `.class` would need the "application used to perform action" line to be:

Getting Started With intent[®] on Windows[®]

```
C:\INTENT\SYS\PLATFORM\WIN32\ELATE.EXE -f -c"%1$" -
bsys\platform\win32\tst\appboot.img
```

2.7.3. Example 3

If `intent` is installed in `C:\INTENT` then an association for `.00` which contains a Java tool would need the "action" to be "Java" and the "application used to perform action" line to be:

```
C:\INTENT\SYS\PLATFORM\WIN32\ELATE.EXE -f -c";f%1$"
```

In this last example the action Java would appear in the "right click" menu under explorer.

2.8. Start-up Directory

`intent`'s `-D` command line option enables you to tell `intent` the startup directory. If `-D` is specified without a directory, `intent` will start in the current directory. If `-D` is not specified at all, `intent` will start in the `intent` base directory found during file name conversion, or the `intent` base directory found when the `intent` executable was loaded. The `sys\platform\win32` directory structure is removed to get to the `intent` base directory.

2.9. Loading string

`intent`'s `-L` command line option instructs `intent` to display a dialog box while loading. If `-L` is specified without a loadingstring, "intent loading" will be used by default. This string can be overridden by specifying a loadingstring. If the string contains space characters, it must be enclosed in quotes.

2.10. Memory

By default `intent` initially gets 4 megabytes of memory. This is sufficient to run the simple tools such as the assembler. `intent` will automatically get more memory from Windows when the initial memory is exhausted. To control how the memory is allocated and extended, use the `M`, `U`, and `X` switches as follows:

- ◆ `M` specifies the initial memory allocation in K. The default is 4096K
- ◆ `U` specifies the increment unit in K. That is the amount requested from Windows each time more memory is required. The default is 4096K
- ◆ `X` specifies the maximum amount that `intent` should extend the initial allocation. If this is the same as the value specified for `-M`, automatic extension will not occur. The default is 131072k

2.11. Trace flags

`intent`'s tracing mechanism permits output to a file and/or a debugger:

If trace to file is requested, by default the file will be `ktrace.log` in the current directory. However, you can specify an alternative file by using the `-K` switch.

If trace to debugger is requested, then the trace strings will appear in the debugger's output window, if a suitable debugger is running. `EBUG` and `MSDEV` are known to support this feature. (In fact, any software capable of trapping Windows Debug Events using `WaitForDebugEvent` will display the trace strings.)

Traceflags is an optional decimal value, interpreted by `intent` as a bit-field. The binary representation of this value is:

00000001	Trace to file (this is the default if the <code>-T</code> switch is not specified)
00000010	If "Trace to file" is enabled, flush buffer after each string output
00000100	Trace to debugger

If -T is specified without a traceflags value, then all flags will be set.

Therefore,

- ◆ -T0 disables tracing
- ◆ -T1 traces to file, without flushing the buffer after each write (this is the default)
- ◆ -T2 (same effect as -T0)
- ◆ -T3 traces to file, flushing the buffer after each write
- ◆ -T4 traces to debugger
- ◆ -T5 traces to file (without flushing the buffer) and debugger
- ◆ -T6 (same effect as -T4)
- ◆ -T7 traces to file (flushing the buffer) and debugger
- ◆ -T (same effect as -T7)

2.12. Log file

intent uses *ktrace.log* in the startup directory, as the *ktrace* log file. The log file is also used by *Elatecon* to capture Standard Error (*stderr*) output.

If you need to use a different file to *ktrace.log*, use the -K switch on the command line (before the image file path).

The format of the -K switch is “-Klogfile” where logfile is the full name of the log file.

e.g. *-Kj:\dir\mylog.log* will create the *ktrace* log file as *j:\dir\mylog.log*.

2.13. Interactive

intent's -I command line switch determines whether or not an interactive session (that is, a TTY device) is required.

The format of the -I switch is “-linteractive” where interactive is either “0” (meaning an interactive session is not required) or “1” (meaning an interactive session is required).

If the -I parameter is omitted, the default is “interactive”.

2.14. Input file

This parameter is only applicable to *Elatecon.exe*. If this parameter is used, *Elatecon* takes its input from Standard Input. Input file contains the *intent* commands that you want to execute.

If this parameter is omitted, Standard Input defaults to the keyboard.

The alternative method for inputting commands is to use the -c parameter (see above).

2.15. Output file

This parameter is only applicable to *Elatecon.exe*, which sends its output to Standard Output. Output file will contain all *intent* output produced during the session.

If this parameter is omitted, Standard Output defaults to the display.

3. Creating an intent Short Cut

The easiest way to configure intent is to create an intent shortcut. Shortcuts also make it easy to run several versions of intent using the same Elate.exe or Elatecon.exe. The procedure to create one is as follows:

- ◆ Right click on the desk top
- ◆ Select "new/shortcut"
- ◆ Enter the command line, i.e. the path to the intent executable plus any parameters (see examples below)
- ◆ Click "Next"
- ◆ Enter a suitable name for the shortcut
- ◆ Click "Finish"
- ◆ A shortcut will be created on the desk top

3.1. Example for Elate.exe

If intent is installed in C:\INTENT, then a typical command line would be:

```
C:\INTENT\SYS\PLATFORM\WIN32\ELATE.EXE
```

Exactly the same procedure may be followed to create a shortcut within the Windows start menu.

3.2. Example for Elatecon.exe

A batch file may be created, containing the following:

```
C:\INTENT\SYS\PLATFORM\WIN32\ELATECON.EXE SYS/PLATFORM/WIN32/DEVELOP.IMG  
-KC:\ERR <C:\IN >C:\OUT  
EXIT
```

where C:\IN contains intent commands, for example:

```
make -f app/example/SpecialTool.asm  
sysgen sys/platform/myplat/myimage.sys  
exit
```

This example would take input from C:\IN, send standard output to C:\OUT and send error output to C:\ERR.

4. Invoking Elatecon

As Elatecon is a Windows Console Application, Standard Input and Standard Output can optionally be redirected. Therefore, Elatecon can be used either interactively or non-interactively. Of course there are other combinations of Stdin/Stdout redirection, but these are the most common:

- ◆ **Interactive:** If neither Stdin nor Stdout are redirected, input and output default to the keyboard and display. That is, Elatecon operates interactively.
- ◆ **Non-interactive:** If both Stdin and Stdout are redirected, then Elatecon can be used in an automated way. For example, to run a sequence of intent commands from a command file and redirect the results to another file.

(Note there are two methods of inputting commands to Elatecon, either the -c switch or Standard Input)

The most convenient way of invoking Elatecon is to create an intent shortcut. In this way, you can specify all the parameters you need once when you set up the icon, and subsequently all you need to do is double-click the icon.

Alternatively, invoke a Dos Box, and either type in the Elatecon command line manually each time or run a batch file that contains the Elatecon command line.

4.1. Restrictions

1. If double quotes are used to delimit strings, they must be preceded by a backslash (\). For example,

```
sys\platform\win32\elatecon -c\"echo one;echo two\"
```

2. If a batch file that takes parameters is used to invoke Elatecon, and semi-colons are used within a string parameter, then double quotes (not single quotes) must be used to delimit the string. Otherwise, Windows will strip out the semi-colons. For example, if a batch file called c:\elatec.bat contains:

```
sys\platform\win32\elatecon %1 %2 %3 %4 %5
```

then the following command must use escaped double quotes:

```
c:\elatec -c\"ls etc;ls\"
```

5. Debugging intent

The fbug debugger is an interactive system level debugger for intent, which can either be used in text mode or with a graphical user interface. The fbug debugger has been designed to perform such tasks as debugging applications, device drivers and other pieces of intent code. The debugger can be of use in the following areas:

- Examination and modification of registers
- Single stepping through code
- Detecting Incorrect parameters*
- Identifying stack overflows*
- Setting and clearing multiple breakpoints
- Expression evaluation (VP and Native)
- Detecting misaligned loads and stores*

* These features are only available with the develop image.

The fbug debugger can be invoked with either a text or graphical user interface. The basic format of fbug text mode invocation is as follows. Under Windows[®] *fbug* should be run by modifying the icon for starting intent (although a shortcut should already have been provided by the InstallShield[®] process). For example, if it says:

```
d:\work\intent\sys\platform\win32\elate.exe -Bdevelopopt.img
```

This should be changed to:

```
d:\work\intent\sys\platform\win32\fbug.exe  
d:\work\intent\sys\platform\win32\elate.exe -Bdevelopopt.img
```

The above example assumes local debugging, although a shortcut for debugging a remote target can be created in a similar example. Double clicking the shortcut will cause a window to appear on the screen upon which text mode fbug commands can be entered. When some part of the intent system or any application it is running causes a fault, or the user switches to the fbug window and hits ctrl-C for Windows, the intent system is stopped and fbug is entered into. It prints a brief description of the problem (e.g. a hard coded breakpoint), a register dump, the name of the tool where the problem occurred and the offset into that tool, and the source line where the problem occurred (only if the tool was assembled with the -g option). If source is not available a disassembly of the instruction which caused the problem can be displayed.

Note that in some cases it may be advisable to further adjust the default properties of the shortcut. For example, text mode users may prefer to adjust the layout and screen buffer size so as to obviate the need for a scroll bar, as on some versions of windows, shortcut is created with a massive screen buffer, which means that fbug uses up the whole buffer and displays a scrollbar. This makes fbug unusable, as it becomes necessary to adjust the properties to make the size of the screen buffer the same as the size of the window, thereby making it impossible to see the whole of fbug at any one time.

The debugger may also be invoked through the following scripts (assuming no specialised intent or fbug options are required):

- *intent_debug* – Starts a shell session under the debugger
- *intent_media_debug* – Starts a intent Media System session under the debugger

Note that these invoke the graphical user interface for fbug, known as fbugwin. Before starting, the current directory should be the intent root. For, Windows, this is achieved by setting the "start in" part of the shortcut. To create a shortcut for fbugwin under Windows, a shortcut should be created whose target is set as follows:

```
"C:\intent\sys\platform\win32\fbugwin.exe [options]"
```

fbugwin may start one of a selection of intent sessions whose parameters have previously been set up (for example, develop, developopt, a target board connected to using a serial line, another machine on the Internet etc), exit the intent session, and then restart and continue debugging (without quitting fbugwin, and possibly using a different target setup). For more information on fbug please see the html documentation within the intent release, or the fbug debugger user guide.

6. The intent Box

Once intent has been installed and a shortcut created, clicking on this shortcut will bring up the intent box.. This may take a few seconds, in which time an " intent loading" message box is displayed.

This box has three menus located upon its upper bar, which are as follows:

6.1. The File Menu

The file menu has two items.

- **Print Setup**

Print Setup allows you to configure a printer for intent to use.

In order to use a printer from within intent you will need to load the intent generic Windows printer device driver.

- **Exit**

Exit will terminate intent under all circumstances, so remember to close applications with open files first or data will be lost.

6.2. The Edit Menu

The edit menu has four items:

- **Mark**

Mark disables mouse support within intent and displays a "rubber band" style selection rectangle to allow you to select an area to transfer to the clipboard.

Selecting Mark again will remove the selection rectangle and return mouse control to intent.

This item is duplicated on the toolbar as the Mark button.

- **Copy**

Copy will copy a marked area to the clipboard.

Selecting Copy with no marked area will cause the entire intent display to be copied to the clipboard.

This item is duplicated on the toolbar as the Copy button.

- **Paste**

Paste will transfer any text located within the clipboard into the keyboard buffer.

This item is duplicated on the toolbar as the Paste button.

- **Fonts**

The Fonts item will bring up a standard Font selection box. Only fixed pitch fonts are displayed. You should be aware that not all fonts will display properly since intent uses the OEM character set - a typical problem is that box drawing characters do not appear correctly.

This item is duplicated on the toolbar as the Font button.

Getting Started With intent[®] on Windows[®]

An alternative method of changing the font size is to drag the edge of the intent window to the required size, and intent will automatically calculate the largest point size of the currently selected font typeface that will fit into that window size.

6.3. The View Menu

The view menu has two items:

- Menu bar

This option allows the Menu bar to be displayed or hidden.

- Tool bar

This option allows the Tool bar to be displayed or hidden. Whenever the options on this menu are modified, the intent window is automatically resized.

6.4. The Help Menu

The Help menu has two items:

- **Help Topics**

Displays more detailed information about using intent with Windows. This incorporates a Frequently Asked Questions section. Invoking Elate.exe usually does this, and then clicking on the Help menu, but the help viewer can also be directly invoked:

To do this on Windows NT run:

```
winhlp32 <pathname>elate.hlp
```

On Windows 95 either:

```
winhlp32 <pathname>elate.hlp
```

or

```
winhelp <pathname>elate.hlp
```

- **About Elate**

Displays information relating to whatever version of the intent program has been installed.

7. intent Help Functions

Online help within *intent* is provided in standard html format. Online help is also viewable through the *intent* box itself. In particular two functions are provided to assist the user in this respect. These are as follows:

- `help` - view help file
- `html` - view miscellaneous html file

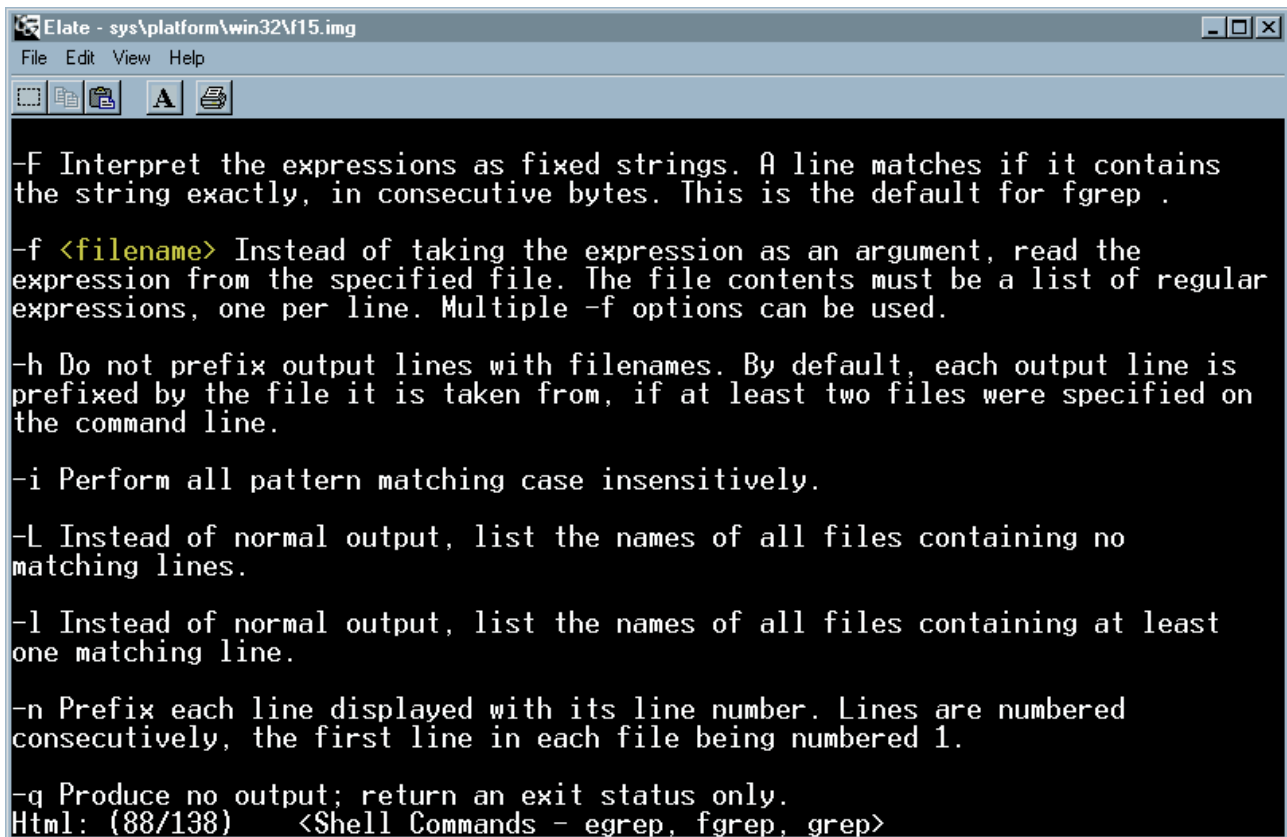
The syntax for the `help` command is

```
help <text>
```

For instance:

```
help grep
```

The documentation relating to the shell command *grep* is then displayed within the *intent* box.



```
Elate - sys\platform\win32\l15.img
File Edit View Help
- F Interpret the expressions as fixed strings. A line matches if it contains
the string exactly, in consecutive bytes. This is the default for fgrep .
- f <filename> Instead of taking the expression as an argument, read the
expression from the specified file. The file contents must be a list of regular
expressions, one per line. Multiple -f options can be used.
- h Do not prefix output lines with filenames. By default, each output line is
prefixed by the file it is taken from, if at least two files were specified on
the command line.
- i Perform all pattern matching case insensitively.
- L Instead of normal output, list the names of all files containing no
matching lines.
- l Instead of normal output, list the names of all files containing at least
one matching line.
- n Prefix each line displayed with its line number. Lines are numbered
consecutively, the first line in each file being numbered 1.
- q Produce no output; return an exit status only.
Html: (88/138) <Shell Commands - egrep, fgrep, grep>
```

Press `Q` to be able to enter new commands and to depart from the help display.

An index of all *intent* documentation available online can be found within the *intent* root directory. The documentation is listed according to category, (i.e. device drivers, shell commands and so on). To view this file type:

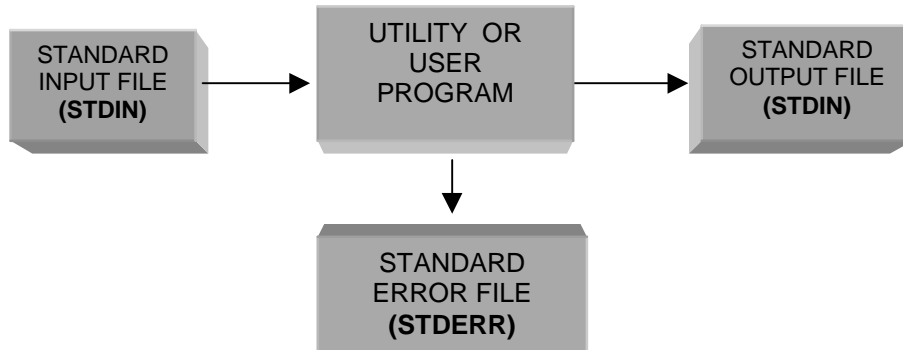
```
html contents.html
```

Updates to the online documentation can be found on the Tao Website.

8. The intent Shell - An Introduction

The intent shell is a scripting command language interpreter. It is able to read and execute commands from the user, and can therefore provide an interface to the underlying mechanisms.

Entered commands are dealt with by the command processor, which calls on the services provided by the intent kernel as and when required. The results of most commands are displayed to the location at which standard output has been specified, for instance the screen. The intent shell has a similar feel to standard UNIX[®] shells, but although designed to offer a level of functionality comparable to a zsh shell, it also has a much smaller footprint. Please note that the behaviour of the intent shell is not always identical to that of other shells, such as bash.



8.1. Command Conventions

Commands can be entered at the \$ prompt, displayed on the screen within the intent box. Where a command line is shown, text enclosed in angle brackets <thus> should be replaced by an actual parameter when typing a command. Parts of the command line shown in square brackets [thus] are optional.

So for example :

```
command <parameter1> [parameter2] [parameter3> ...]
```

In this case parameter 1 is mandatory, and parameters 2 and 3 are optional. However, parameter 3 cannot be specified unless parameter 2 also is. Parameter 3 may be repeated.

• Options

Specifying additional options can modify commands. All options must be preceded by "-". Multiple options can be specified together, so for example, "-abc" would behave identically to "-a -b -c". However some options may take additional arguments, in which case multiple options should not be specified.

Options can appear anywhere on the command line, between parameters. For clarity however, it is recommended that options be placed immediately after the command name, and therefore before any parameters.

8.2. Most Commonly Used Shell Commands.

All of these commands can be run from the command line.

intent Command		Unix Equivalent	DOS [®] Equivalent
ls	lists directories	ls	dir
speed	benchmark	-	-
cat	concatenates named files	cat	type
echo	copies arguments to standard output	echo	echo
touch	updates timestamps	touch	-
exit	exit shell	exit	exit
shutdown	quit	shutdown	-

Getting Started With intent[®] on Windows[®]

java

runs Java™ class

java

java

8.3. Simple Commands

- **Listing directories - ls**

This command lists the named files. If a directory is named, its contents are listed. If no filenames are given, the contents of the current directory are listed instead. Typing this in after the command prompt:

```
$ ls [<filename> ...]
```

would produce something like this:

```
dev  ebug.exe  feq.exe  lang  makefile
app  docn
```

- **Concatenate the Named Files - cat**

By typing in this:

```
$ cat [<filename> ... ]
```

it is possible to either place the content of any named files into a new file or to copy the input typed into the keyboard. For example:

```
$ cat demo/example/hello.asm
```

- **Benchmark current speed - speed**

This function benchmarks the speed of the current processor, and displays the results to standard output. Typing in this after the command prompt:

```
$ speed
```

will result in something like this:

```
VP MIOPS = 62.060606          (integer)
VP MLOPS = 0.969696          (long)
VP MFOPS = 0.290909          (float)
VP MDOPS = 0.028985          (double)
```

Millions of these ILFD operations are performed every second.

- **Leaving intent – Exit/Shutdown**

Exit terminates the shell, while shutdown shuts down intent. These have almost the same effect from the initial shell, but nowhere else.

It is recommended that one of these commands should be used to shutdown the system, rather than clicking on the cross in the upper right-hand corner of the “intent box”.

9. Directory Structure

Files within *intent* are organised into directory structures - as is commonly the case, a directory is itself a file, containing the name and locations of the files it contains. Consequently any commands that apply to files are also applicable to directories. One or more files can be specified. The following directory areas can be used as a guide to locate files:

Applications	app	All applications
AVE	ave	Multimedia Libraries
Com.uk	com	Java™ classes, following Java™ namespace conventions
Demonstration	demo	Example Programs
Device Drivers	dev	Device Drivers
TCP/IP Subsystem	etc	General configuration files often used by TCP/IP, network and Host OS related files
Fonts	fonts	TrueType® and PostScript® Type 1 fonts.
Home	home	The user's home directory
Java	java	Java™ Libraries
Languages	lang	Programming Languages
Libraries	lib	General Library
Sounds	sounds	Audio specific parts of multimedia toolkit
System	sys	System Directory

It is easiest to find a required program by following this directory structure.

9.1. More Simple Commands

- **Change Directory - *cd***

```
cd [<directory>]
```

This function changes the current working directory to another specified directory.

- **Print Working Directory - *pwd***

```
pwd
```

Displays the name of the current directory to standard output.

- **Create New Directory - *mkdir***

```
mkdir <pathname>
```

Creates directories with the pathnames that have been specified.

10. The Sysbuild Utility

The sysbuild utility can be used to generate system image files, suitable for downloading to the target hardware. To do this, it calls upon the underlying functionality provided by the sysgen utility. To create an application sysfile see the more detailed documentation located at *sys/platform/sysbuildapp.html*), or to create a platform sysfile see *sys/platform/sysbuildplat.html*.

Demonstration application sysbuild files are available in the *sys/platform* directory, although it should be noted that the file *sysbuild.sys*, in the same directory, is **not** a demo file. Each demo has a corresponding *.html* file of the same name, which contains further information. Some demos may require third party application files to work - where this is the case, it will be stated in their *.html* files.

To generate a bootable system image from an intent command line, type the following:

```
sysbuild <platform> <appsysfile>
```

This is where:

<platform> Specifies the platform to create an image for. For example win32.

<appsysfile> Specifies the application's sysbuild file. This file describes the application in terms of the tools and data files required, as well as how to run the program. All application sys files must have the extension *.sys*, but it does not need to be specified on the command line.

In particular, two sys files are provided; *pjavafull.sys* and *pjavamin.sys*. The *pjavafull.sys* file produces a full PersonalJava image suitable for Java program development, whereas the *pjavamin.sys* file produces a minimal PersonalJava image that does not include any of the optional PersonalJava components. These images can be used to run a Java application from the platform's file system. It is therefore only useful on a hosted platform. To run a Java application the whole application and any data files must exist in the host file system below the intent root directory. On the intent command line use the *-c* option to run the program, for example to run the Java program *my/java/app.class* with the full PersonalJava image enter the following:

```
sys/platform/<platform>/elate
-Bsys/platform/<platform>/pjavafull.img
-c" jcode my/java/app"
```

Further information is available from the following sources:

<i>app/stdio/sysbuild.html</i>	How to use the sysbuild command.
<i>sys/platform/sysbuildapp.html</i>	How to create a new application system configuration file.
<i>sys/platform/sysbuildplat.html</i>	How to create a new platform system configuration file,
<i>sys/platform/sysbuildref.html</i>	Reference guide to sysbuild values.

11. Using The *Sysgen* Utility

The entirety of *intent* is tailored around the needs of the application it is running, incorporating the precise minimum set of tools needed to successfully run that application and no more. The *sysgen* utility (system image generation) is used to generate a bootable system image from a collection of tools and a specification of the user's requirements.

Essentially *sysgen* takes an "instruction file", and then follows the instructions contained within that file, to make one or more *intent* system images. It does this by analysing the references made by the applications and device drivers to the kernel, library and user tools, and determines the exact requirements of the target system. All necessary tools are then loaded, translated, bound and written to an image file, along with any necessary data.

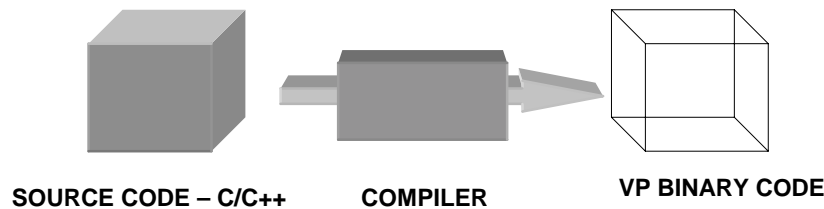
intent is able to translate a tool from virtual byte codes (VP) to native code in any one of three ways:

- Automatically when the tool is loaded from store (e.g. disk). This is suitable for a PC or workstation environment, where the load time from store is long when compared to the time for translation. Tools can be translated as and when loaded by *intent* without incurring any significant performance penalty in this situation.
- Manually from the shell prior to run-time of the program. This is performed using the *translate* command.

When the system image file is generated using *sysgen*. Using *sysgen* it is possible to pre-translate VP byte codes into native code at the time the system is built. For example in a PDA, there is no hard disk from which to load the tools the code typically being stored in ROM.

12. Using the C /C++ Compiler

The compiler is used to convert C or C++ source code into VP source code. The assembler is then used to convert this into VP binary code.



intent comes with a number of 'C' example programs for the new user to begin with, and these can be found in the directory *demo/example/c*, as can the source code for the example program used here, 'hello world'. The following program was coded in source file *demo/example/c/hello.c* :

```
#include <stdio.h>

int main (int argc, char **argv)
{
    printf("Hello from C\n");
    return 0;
}
```

To compile this program type this in at the shell prompt \$:

```
$ vpccl demo/example/c/hello.c
```

This will then create a tool *vpout.00* in the root directory. To then run that tool type this in at the shell prompt :

```
$ ./vpout
```

To compile the program '*hello.c*' while using more advanced options type the following at the intent command prompt:

```
$ vpccl demo/example/c/hello.c -o demo/example/c/world
```

The instruction to invoke the compiler is *lang/cc/bin/vpccl*. The compiler is then given the name of the source program, including its full pathname, which it is to compile.

The line above invokes the compiler to compile the program *demo/example/c/hello.c*, telling it to output the tool to the same directory location and to name the tool 'world'. So, if there were no errors, the compiler will have created a separate file with the name 'world', suffixed with *.00*. This is the executable file.

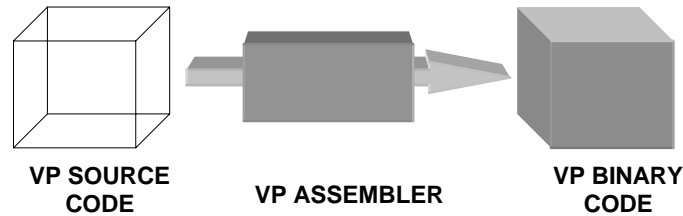
To run this program, type at the command prompt:

```
$ demo/example/c/world
```

The words "Hello from C" are then printed to the screen.

N.B. Note that execution, compilation and assembly should be performed from the root directory in intent.

13. Using the VP Assembler



The *intent* assembler is used to take a source file, of either VP or native assembly source, assemble the contents and output one or more files (typically in the form of an *intent* tool).

To run an application or to call a tool, they must have been assembled first. All *intent* source programs are usually suffixed with *.asm* to denote that they are source.

To assemble the program '*firsthello.asm*' type the following at the *intent* command prompt **while in the root directory** :

```
$asm -v demo/example/firsthello
```

If there were no errors, the assembler will have created a separate file with the name *firsthello*, suffixed with *.00*.

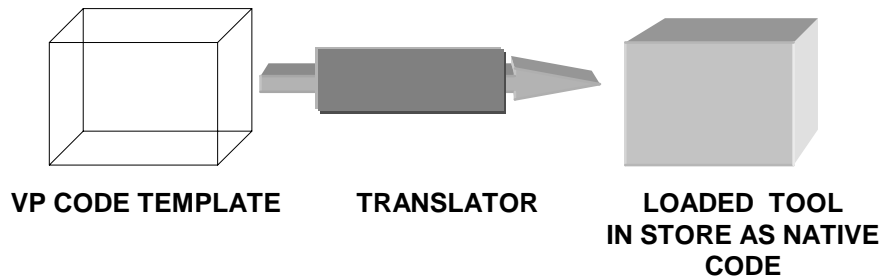
In the above example command line, the verbose option has been used (-v) to provide additional diagnostic information. It is recommended that developers new to *intent* use this option. This is the executable file and to run, type from the command prompt:

```
$demo/example/firsthello
```

Please note that the full pathname is required but no suffix. The words "Hello world" are printed to the screen.

N.B. Note that execution, compilation and assembly should be performed from the root directory in *intent*.

14. The Translation Procedure



Translators are used in three situations:

- When called by the sysgen utility.
- When called by the tool loader for loading tools (dynamic binding).
- "Pre-translation" of code using the "translate" command.

The translate command can be used to pre-translate a Java™ class (.class) to the VP tool (.00) files, or to pre-translate a VP tool (.00) to a native tool (some other numerical extension, for example .15 for Pentium™). Doing this means that sysgen and the tool loader will use the pre-translated versions, rather than automatically calling the translator.

```
translate [options] <filename> ...
```

For each specified file, a translator is run on the tool contained within that file. The translated tool is then stored in a file with the same name except that an appropriate ".nn" suffix will be added, and any ".00" suffix will be removed. Directories are created as necessary to do this. The translation of a VP code into native less time than that taken to load that code from storage.

The main option to be used with this command is:

```
-t<translator>
```

Use the specified translator. (By default the current system translator is used. The prefix `sys/tr/' is added).

A more detailed list of options is described in *"The intent Shell Commands Reference Manual"*.

15. Using intent and Java™ Technology

intent JTE supports the PersonalJava™ 1.1.3 and 1.2 specifications. A choice between the two of these will be offered during the installation process. Java™ applets and applications may be invoked through the scripts discussed in section 1.2 of this document. Otherwise, the following command is used to load and, if necessary, translate a Java class, so for example to run this demonstration application type the following at a intent shell prompt:

```
java demo.example.j.Hello
```

It is also possible to format this command as follows:

```
java demo/example/j/Hello
```

The java command starts a Java virtual machine (JVM) and proceeds to run a Java application within it. Please note that the name given to the Java command needs to be the absolute name of the class file. Execution starts in the main method of <classname>. Zip and jar files may be specified on the command line. The classpath option simply specifies the classpath to be used when loading classes from the local filesystem. Many directories or jar files can be specified, each separated by a ','

The Jcode VM has a notion of two different types of classes, system classes and application classes (which comprise of classes loaded through the classpath or by a classloader). System classes are those classes comprising the Java libraries, and are loaded using a built in classpath of '/' and will stay resident in memory (i.e. translated and bound) during the lifetime of a VM and after the VM exits, until a tool flush event occurs. Conversely, as far as application classes are concerned, the classpath is valid for the lifetime of the VM, while a classloader is only valid during its own lifetime.

When looking for a class the VM first searches the system classpath (i.e. '/'), and then the application classpath. The classpath used for application classes is specified by using the classpath option on the command line; any mention of '/' on the application classpath will be silently ignored. . There is no mechanism for modifying the classpath used for system classes. For more information about other options to this command, please see the file *app/stdio/java.html*.

In order to view an applet embedded in a html page, a command similar to the example below should be entered at an intent shell prompt:

```
appletviewer applet.html
```

Note that the html file processing will be different to other applet viewers, and <applet> tags in incorrect html may not be recognised.

16. Introduction to the intent Multimedia Libraries

The aim of the intent multimedia toolkit is to provide a component based user interface (i.e. a tool-kit which provides the flexibility and modularity to build any form of audio visual environment). This allows multimedia elements, such as an MPEG video player to have a standard programming interface that can be incorporated into any application.



With conventional operating systems with a Graphical User Interface [GUI] the limit of the functionality of the program is directly imposed by the restrictions of the GUI. By contrast the intent multimedia toolkit imposes no such limitations, and incorporating the ability for intent multimedia toolkit users to build their own gadgets out of the tools provided, bypasses the problem of the predefined nature of the interface development kit conditioning the end result. The final vendor solution reflects the requirements and restrictions of the application, and not those of the Operating System or its graphical toolset. Because there are no restrictions on the usage of the intent multimedia toolkit, there are no limitations upon what can be produced by it. The advantages of this modular, object-based approach is that applications are not determined by the look-and-feel of the user interface. From these tool-sets they will effectively be able to build their own GUI and applications for PDAs, telephones, digital cameras, interactive television set-top boxes, workstations or any other form of product.

Sample intent applications are provided in the directory *demo/ave*.

16.1. Simple Customisation

The file *dev/ave/auto.scr* contains a list of the applications that are to be automatically launched at start-up. For example, the tiled intent backdrop can be configured to run a specific script when clicked. In practice, any application could be started through these means. However, it defaults to the system menu program *dev/ave/dsk/runapp.scr* (as described above).

This simply lists the directory tree rooted in */app/start/*; each script file found here is displayed as a menu item, while subdirectories become submenus. Thus, to add an option to the system menu, create a script file to invoke an application at the appropriate place in the *app/start/* directory tree (the script's filename, minus the extension, becomes the label).

17. Documentation

The following documentation has been provided to assist the developer in all areas of using intent and programming with Virtual Processor code:

- VP Programming Guide
- VP Reference Manual
- Object Based Programming Guide
- The Elate Device Driver Design Guide
- System Programming Guide
- Debugger User Guide
- Reference Manual for the Sysbuild Utility
- Reference Manual For The *Sysgen* Utility
- Porting Elate
- Introduction to Java™ Technology on intent
- Java And intent User Guide
- Intent Media Libraries Programming Guide
- C/C++ Compiler User Guide
- Glossary Of All Key Terms

Further information upon forthcoming Tao Group's products can be obtained from www.tao-group.com, where it is also possible to register upon the company email list.

18. Training

Tao boasts a highly skilled training department, which is responsible for running training courses for all new employees and new customers. Tao takes a relaxed but professional 'workshop' approach to training. The courses are designed to be as 'hands-on' as possible while covering key theoretical concepts thoroughly.

Tao currently offer the following courses:

18.1. Introduction to intent

This is a course for the newcomer to intent. Topics covered include:

- ◆ Installation
- ◆ Architecture
- ◆ Configuration
- ◆ Utilities
- ◆ Intro to C, Java™ technology and VP programming tools
- ◆ System building
- ◆ Debugging

A complete list of all the topics covered in the course is available.

- **Pre-requisites**

Delegates are not expected to have any previous experience of intent. They are expected, however, to have some programming experience in a language such as C or assembler and familiarity with basic operating system concepts.

18.2. Introduction to VP Programming

This course is designed to enable programmers to rapidly familiarise themselves with programming in VP assembly language. The course also includes coverage of Object Based programming in VP.

- **Tailored Courses**

Versions of the above course tailored to customer's specific requirements can be created. For certain situations we are prepared to hold the course at the customers location. Further information regarding this or any other queries on training, please contact Tao's Training Manager, Tony Bedford at bedford@tao-group.com

Further information upon forthcoming Tao Group products can be obtained from www.tao-group.com, where it is also possible to register upon the company email list.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

*InstallShield[®] is a registered trademark of the InstallShield Software Corporation
Winzip[®] is a registered trademark of Nico Mak Computing, Inc.*

© Tao Group Ltd or Tao Systems Ltd. 2000, 2001. All Rights Reserved.

Copyright in the software either belongs to Tao Group Ltd or Tao Systems Ltd. The software may not be used, sold, licensed, transferred, copied or reproduced in whole or in part or in any manner or form other than in accordance with the licence agreement provided with the software or otherwise without the prior written consent of either Tao Group Ltd or Tao Systems Ltd.

No part of this publication may be reproduced in any material form (including photocopying or storing it in any medium by electronic means and whether or not transiently or incidentally to some other use of this publication) without the written permission of the copyright owner.

*Elate[®], intent[®] and the Tao logo are registered trademarks of Tao Group Ltd.
Digital Heaven[™] is a trademark of Tao Group Ltd.
The rights of third party trademark owners are acknowledged.*