Getting Started With int**e**nt® on Linux®

Version 1.48

1. Introduction

int**e**nt® 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 int**e**nt 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 int**e**nt 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 int**e**nt 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.

int**e**nt incorporates the following features:

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

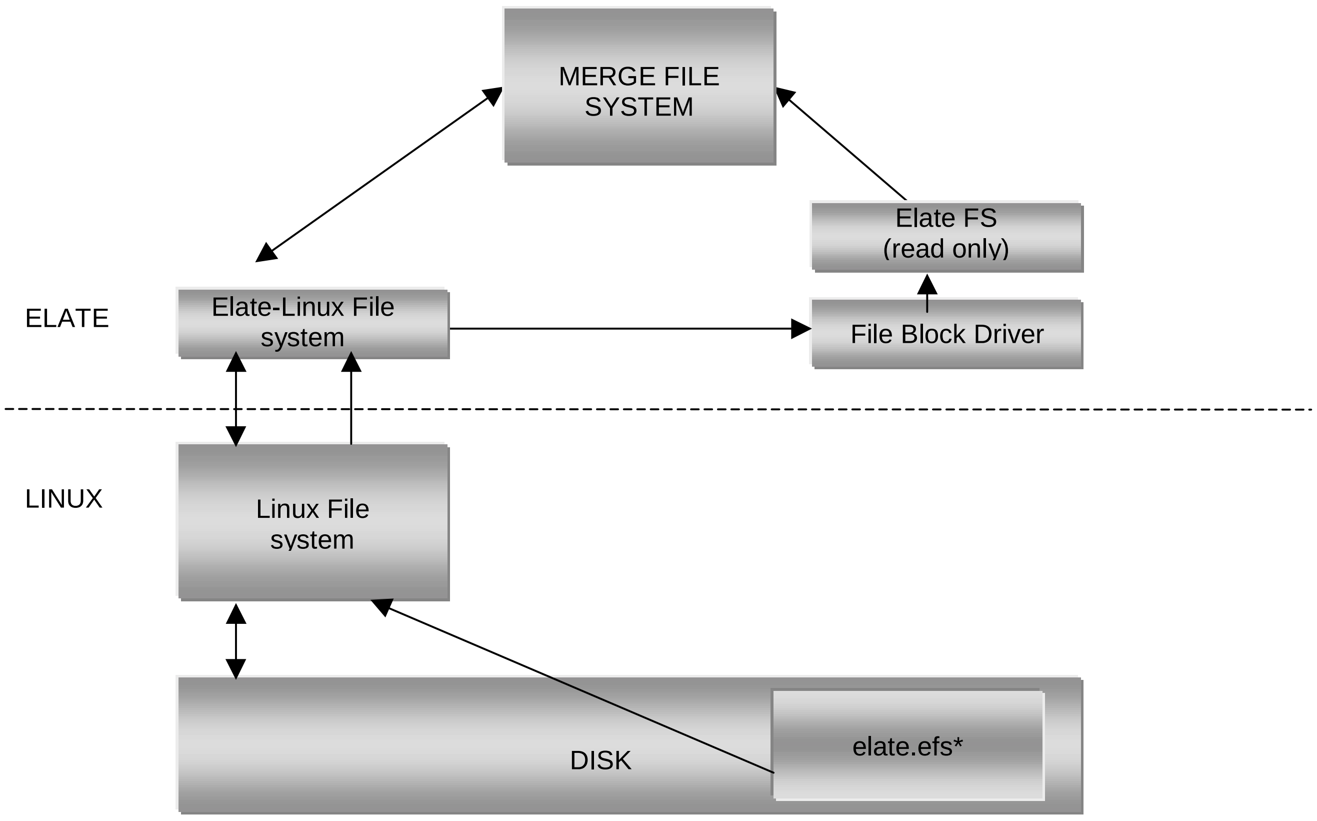
The int**e**nt media system has been designed to drive interactive and other multimedia content across the broad range of wireless and wired digital client devices including low powered, small footprint products. Tao's drawing, windowing, compositing and gadget 2D libraries form the cornerstone of the multimedia toolkit for creating differentiated, highly branded products. The int**e**nt system includes a Sun authorised implementation of the Java Virtual Machine and libraries (int**e**nt 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 int**e**nt. Further technical information is available, either in the form of online help or as printed user guides.

1.2. 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 Elate file system

Figure 1. The Merge file system

The packaged version of intent sets up the root file system as a merge of the following components

- A shared read-only elate.efs
- A read-write file system, rooted in the directory in which the user has installed it.

The read-only elate.efs file is only shared when no add-ons are installed. When an add-on is installed, the shared elate.efs file is copied to the user's intent directory and customised.

1.3. Basic Instructions

In cases where intent are shipped in the form of Red Hat packages (rpm files) or Debian packages (.deb files) it should be installed on the system using the standard package management tools. For example:

Debian Packages

```
dpkg –i <packagefile.deb>
```

Red Hat Packages

```
rpm –Uvh <packagefile.rpm>
```

intent is shipped as four packages;

- *intent*–bin – The 'host-side' binaries and scripts
- *intent*-img – The various images and the file system image elate.efs, which must be installed
- *intent-gpl-utils* - This need only be installed if make, the C compiler and zip/unzip tools are required. This also contains gzip and vim.
- *intent-doc* - This provides system documentation

Please note that the int**e**nt SDK requires Red Hat 6.1 or Debian 2.2. int**e**nt may then be installed onto the system, thereby making it available to all users. However, because each user may wish to have int**e**nt setup in a different way, they must then run a user installation process. The user installation will configure a directory to run int**e**nt with any add-ons (such as patches or the gpl tools) that the user may wish to install. When a user first starts int**e**nt they will be informed that they have not run int**e**nt before and then asked if they wish to run the user install process. Answering "y" or "Y" will run the user installation process, anything else will exit. If the answer is 'yes' the user will be required to agree to a license agreement.

Furthermore, when int**e**nt is run after the initial installation it will assume that the directory from which int**e**nt was started has already been configured. If this is not the case, then it will prompt the user to select an installation or to create a new installation.

When int**e**nt is run in a directory that has already been configured it will check to see if any more add-ons or patches have been installed on the system since the last time. If they have, the user is given the option to install each of these into the currently configured directory. As such, this permits the user to have several installations of int**e**nt installed, each with a different set of add-on packages.

If the user decides at a later date that they wish to install an add-on, which was not installed initially, then they can run the installation program directly. This will prompt the user with all packages that they have not installed rather than just the new ones.

1.4. Patches and Platform Releases

For Linux, patches and platform releases are available in RPM and DEB formats. These must be installed onto the system in the usual manner. However, for these to be available to users they must choose to install them (as add-ons) when they next run int**e**nt.

Please note that in cases where Tao Group had issued unofficial patches to the int**e**nt system, this will prevent any further official upgrades from being accepted. Further unofficial patches will still work. In these cases, it is suggested that users reinstall their original system in combination with an official patch at the point where one becomes available. To ‘recover’ after adding unofficial patches users can create a fresh int**e**nt installation with the user installation process. The RPMs do not need to be uninstalled then reinstalled.

User int**e**nt installations can be removed by deleting the directory into which it was installed.

Each user installation will create an entry in the *.intentrc* in the user’s home directory. This file should not be edited manually.

1.5. A Note about Example Code

The demonstration source code in demo/example and demo/ave is built into the int**e**nt 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 int**e**nt will create copies of the demos and sources, making them accessible from the host OS filesystem.

2. intent and Linux®

2.1. The intent Loader

The intent loader is a Linux program, which forms the virtual hardware for an intent session. It provides the interface between intent and the Linux system calls.

To start an intent session run one of the following wrapper scripts:

- *intent* – Start an intent shell session suitable for developing in VP or C
- *intent_media* – Start an intent session running the intent Media System

Both of these packages contain minimal PersonalJava classes and can therefore be used to develop with the Java language.

These scripts will either use the current directory if it has been configured or prompt for an installation to use.

For example:

```
cd ~/intent
intent
```

For convenience, some wrapper scripts have been provided to allow execution of Java applications and applets directly from the Linux command line:

- *intent_java_stdio* – Run a Java application with stdin and stdout
- *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

intent_applet takes one argument, namely the html file containing the applet. Note that the applet or application must be within the directory hierarchy rooted in the Linux directory where intent has been installed.

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.

Each of these scripts will check the current directory for an intent installation, prompting the user for a valid installation directory as and when appropriate. It will also check to see is any new intent add-ons have been installed since the last time intent was run. If there are any, the user will be prompted for installation instructions. These scripts also support the options –c, which stops the script checking for new add-ons, and –d <dir>, which will use the <dir> for the intent installation. These options must go before any other options on the command line.

Please note that the current set of wrapper scripts may be re-organised in future releases. Their behaviour may change fundamentally and some scripts may be removed.

2.2. The intent User Configuration Script

```
intent_cfg command [options]
```

For example:

```
intent_cfg install ~/intent
```

This script allows the user to install and configure int**e**nt for their requirements. There are various add-on packages available for int**e**nt, which can be installed on a per user basis from this program. The commands for use in conjunction with this are as follows:

- *install* Creates an int**e**nt installation for the current user. The directory to use for the installation may optionally be given on the command line. The program will list the available int**e**nt add-ons and allow the user to select which ones they would like. The program then creates the required directory structure, below the chosen installation directory, ready for running int**e**nt. The information gathered during the install process is stored in the *~/.intentrc* file.
- *check* Checks the int**e**nt installations, using either the current directory or the directory given on the command line as the int**e**nt installation to be checked.

2.3. Loader options

The int**e**nt scripts described above only suffice for simpler invocations of int**e**nt. For more complex situations the following command is required:

```
sys/platform/linux/elate –B sys/platform/linux/ix86/<image name>.img
```

The image name should be one of the 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.

Note that the –B option is required as it defaults to f15.img. Valid options include:

-B<*imagename*>	Specify the name of the image to boot. This can be specified as an absolute filename, or given relative to the current directory. The default is sys/platform/linux/f15.img; this is the standard int**e**nt development system image that starts an int**e**nt shell.
-c<*shellcmd*>	Specify an int**e**nt shell command to be run. If this switch is used, then instead of starting the int**e**nt shell, the system will execute the command and exit. This can be useful for automated tasks.
-l<*directory*>	Specifies the directory to look for plugin DLLs in. This defaults to sys/platform/linux/ix86 and does not normally need to be specified.
-I <interactive>	Determines whether or not an interactive session (that is, a TTY device) is required. 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".
-M<*memsize*>	Specify initial memory size to allocate, in kilobytes. Default initial memory size is 4096 KB.
-U<*incsize*>	Specifies the amount of additional memory allocated from the system when int**e**nt runs

	out of memory. Whenever int**e**nt runs low, the loader will allocate this much more memory. The default is 4096 KB.
-X<*maxsize*>	Specify maximum amount of memory to use, in kilobytes. The default is now 128 MB.
-q	Don't print the banner on startup.
-h	Print usage message instead of starting int**e**nt.

Note that invoking int**e**nt in this manner does not check for add-ons.

2.4. Loader Environment Variables

Some features of loader behaviour may be modified by setting Linux environment variables. The following variables may be set:

ELATE_ISINTERACTIVE	When set to 0, disables terminal interface. Intended when you’re not using the int**e**nt shell interactively, i.e. batch jobs. Default is 1.
ELATE_KTRACE	When set redirects ktrace output to the specified file. Default is to send ktrace output to the screen.
ELATE_BREAKCHAR	Defines which character is used for the break character. Specify the ASCII code of the character desired. The default is 29, ^].

For example:

```
ELATE_ISINTERACTIVE=0 sys/platform/linux/elate
–B sys platform/linuxix86/develop.img –q –c “uname –a”
```

These environment variables may be used with both the wrapper scripts and the long form of int**e**nt command, as described in section 2.3. The ELATE_ISINTERACTIVE variable should not be used with intent_applet, intent_java or intent_java_stdio.

3. Debugging int**e**nt

The fbug debugger is an interactive system level debugger for int**e**nt, 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 int**e**nt 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 developt 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:

```
sys/platform/linux/fbug [fbug options]
sys/platform/linux/elate -B sys/platform/linux/ix86/<image name>.img [intent
options]
```

This will cause a window to appear on the screen upon which fbug commands can be entered. When some part of the int**e**nt system or any application it is running causes a fault, or a SIGINT is sent to int**e**nt, the int**e**nt 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.

For the common case of running the developt image with no int**e**nt or fbug options, the following wrapper scripts may be used.

- *intent_debug* :*developt.ing* - Starts a shell session under the fbugwin debugger
- *intent_media_debug* : *developt.ing* - Starts a intent Media System session under the fbugwin debugger

Note that these invoke the graphical user interface for fbug, known as fbugwin. For example:

```
cd ~/intent
intent_debug
```

For command line invocation, Fbugwin should be started from the int**e**nt root directory as follows:

```
"sys/platform/linux/ix86/fbugwin [options]"
```

Fbugwin may start one of a selection of int**e**nt sessions whose parameters have previously been set up (for example, develop, developt, a target board connected to using a serial line, another machine on the Internet etc), exit the int**e**nt session, and then restart and continue debugging (without quitting fbugwin, and possibly using a different target setup).

For more information on fbug and fbugwin please see the html documentation within the intent release, or the fbug debugger user guide.

3.1. Xfbug

Xfbug is a simple wrapper script for the fbug debugger which spawns a new xterm for the Elate® session's I/O.

```
xfbug [<fbug_options>] <elate_driver> [<elate_options>]
```

xfbug is a simple wrapper script for the fbug debugger on Linux when using ptrace debugging. It should be invoked in exactly the same way as fbug. The only difference is that it spawns a new xterm window, which will be used for the int**e**nt session's input and output. I/O from the debugger will go to the original terminal window, stopping each from corrupting the other's display.

4. int**e**nt Help Functions

Online help within int**e**nt is provided in standard html format, accessible through the following int**e**nt utilities:

help - view help file
html - view miscellaneous html file

If the *intent-doc* package has been installed both html documentation and .pdf manuals are available to native Linux programs, in the */usr/doc/intent* directory.

The syntax for the help command is

```
help <text>
```

For instance:

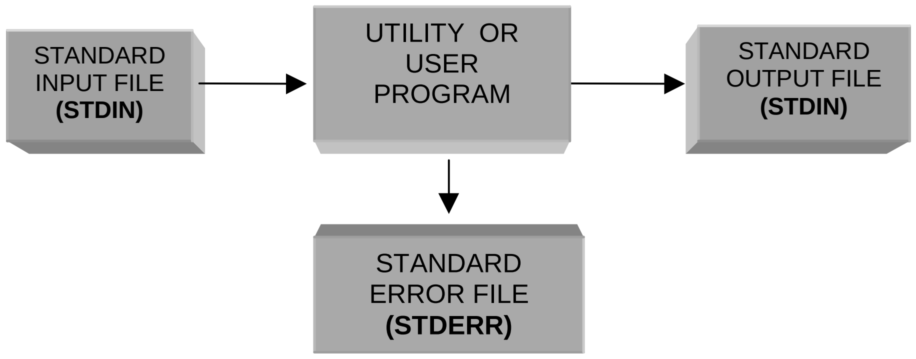
```
help grep
```

The documentation relating to the shell command *grep* is then displayed within the int**e**nt shell.
An index of all int**e**nt documentation available online can be found within the int**e**nt 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
```

5. 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.


STANDARD INPUT FILE (STDIN)
UTILITY OR USER PROGRAM
STANDARD OUTPUT FILE (STDIN)
STANDARD ERROR FILE (STDERR)

5.1. Command Conventions

Commands can be entered at the **$** prompt, displayed on the screen within the intent shell. 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

Commands can be modified by specifying additional options.

All options must be proceeded 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.

5.2. 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
date	displays date	date	date/time

exit	exit shell	exit	exit
shutdown	quit Elate	shutdown	-
java	runs Java™ class	java	java

5.3. 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 copy the input typed into the keyboard.

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 int**e**nt – Exit/Shutdown

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

6. Directory Structure

Files within int**e**nt 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 Toolkit
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.

6.1. 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.

7. 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 int**e**nt command line, type the following:

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

This is where:

<*platform*> Specifies the platform to create an image for. For example Linux/ix86.

<*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:

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

8. Using The Sysgen Utility

The entirety of the int**e**nt platform 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 int**e**nt 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.

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

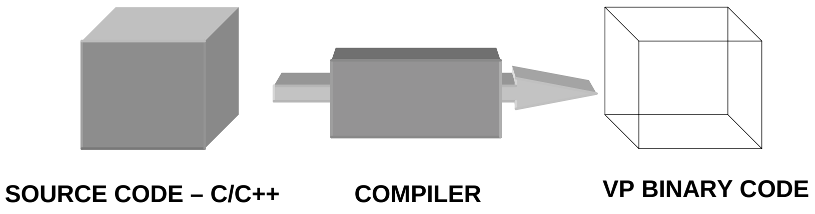
Automatically when the tool is loaded from store (e.g. disk). This is suitable for a PC or workstation environment, where the load time from store is long when compared to the time for translation. Tools can be translated as and when loaded by int**e**nt without incurring any significant performance 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.

9. 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. Note that the compiler is only available if the gpl-utils option has been selected.



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 $:

```
$ vpcc 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:

```
$ vpcc demo/example/c/hello.c –o demo/example/c/world
```

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

The option *-o* tells the compiler the name that it is to give to the output tool.

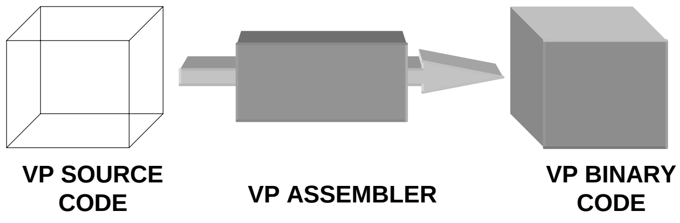
The line above invokes the compiler to compile the program *demo/example/c/hello.c,* telling it to output the tool to the same directory location and to name the tool 'world'. So, if there were no errors, the compiler will have created a separate file with the name 'world', suffixed with *.00*. This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

10. Using the VP Assembler



The int**e**nt assembler is used to take a source file, of either VP or native assembly source, assembles the contents and outputs one or more files (typically in the form of an int**e**nt tool). To run an application or to call a tool, they must have been assembled first. All VP source programs are usually suffixed with *.asm* to denote that they are source.

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

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

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

11. The Translation Procedure

VP CODE TEMPLATE
TRANSLATOR
LOADED TOOL IN STORE AS NATIVE CODE

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 can take significantly 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”.* Please note that translation is normally carried out by the system as required. The programmer does not normally need to use the translate command directly.

12. Using int**e**nt and Java™ Technology

int**e**nt JTE supports the PersonalJava™ 1.1.3 and 1.2 specifications. A choice between the two of these will be offered during the user 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 int**e**nt 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.*

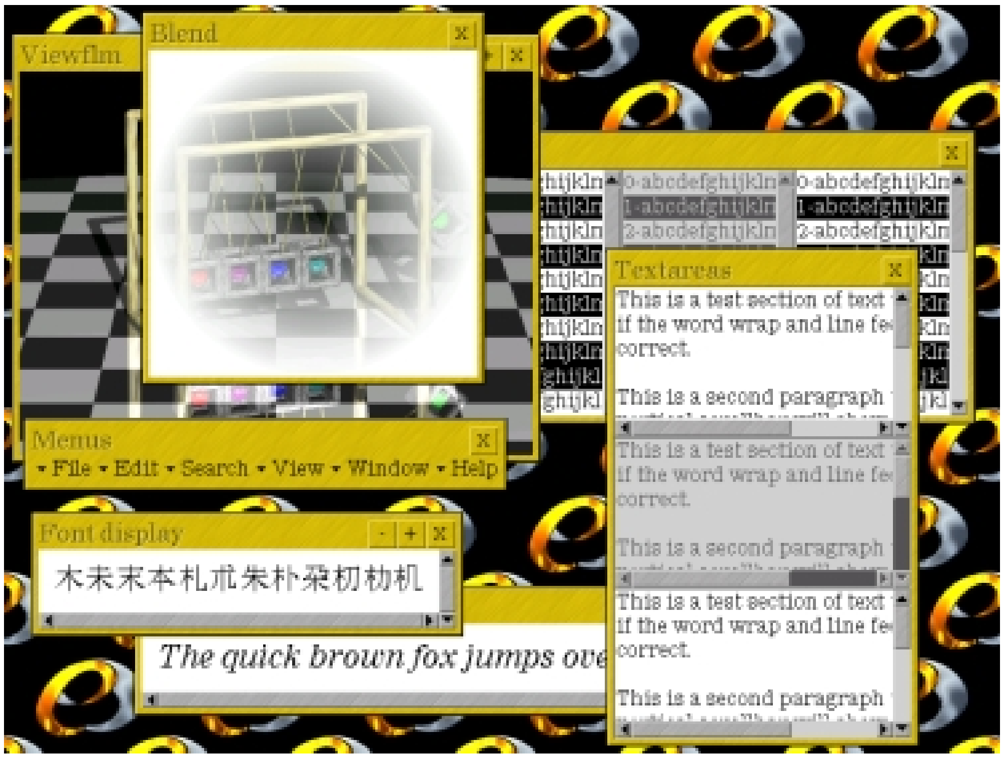
In order to view an applet embedded in a html page, a command similar to the example below should be entered at an int**e**nt shell prompt:

```
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.

13. 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, which can be incorporated into any application.


Viewflm
Blend
Textareas
This is a test section of text
if the word wrap and line fe
correct.
This is a second paragraph
Menus
▾File ▾Edit ▾Search ▾View ▾Window ▾Help
Font display
木未末本札朮朱朴朶朷朸机
The quick brown fox jumps ove

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*.

13.1. 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.*

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).

14. Documentation

The following documentation has been provided to assist the developer in all areas of using int**e**nt 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 int**e**nt
- Java And int**e**nt 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.

15. 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:

15.1. Introduction to int**e**nt

This is a course for the newcomer to int**e**nt. 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.

15.2. Pre-requisites

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

15.3. 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.

15.4. 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