Adhesive

Programmer Interface

Introduction

Note that Adhesive is SHAREWARE - you may have to register for it.

What is Adhesive?

Adhesive is a RISC-OS relocatable module which provides a means of sharing code between many programs. Procedures (code) are grouped together into *objects* which are shared between multiple applications — in a similar way to the Shared C Library.

This documentation is designed to read from start to end, you should skip over bits which are not relevant to the programming language you wish to use. I do not claim to be good at writing documentation, if you have any questions or are unclear about something do not hesitate to get in contact and ask me.

Legal stuff

The files 'License' and 'Register' should be read. In short once you have registered you may use Adhesive with your own programs.

Summary of features

- Code is more reusable, better software engineering practices and modularity.
- Shared objects may be upgraded whilst applications are still using older versions - applications will still run (without modification) even if obsolete procedures are removed from new versions.
- Save memory and application loading time by sharing code.
- Full run-time linker which can handle circular dependencies and inheritance.
- All compiled languages and BASIC can be used.

Why use Adhesive?

What languages can I use?

Adhesive *objects* must be written in either a compiled high level language (e.g. C, Pascal) or in assembler (e.g ObjAsm, BASIC assembler). Because of the way BASIC works, objects can not be written in pure BASIC.

Applications (*users*) which use *objects* can be written in any language (e.g. C, Pascal, BASIC, Assembler) which are capable of doing SWI (system) calls and calling (e.g. BASIC's CALL) machine code.

A special object is provided with Adhesive which allow you to call objects written in C and the Shared C Library from BASIC (using CALL).

Wimp or non-wimp?

Adhesive *objects* can be used from command line programs, WIMP programs, interrupt handlers, relocatable modules — in fact just about anywhere. You do not require the window manager to use Adhesive.

What versions of RISC-OS? Will it run on my RISC-PC?

Adhesive requires RISC-OS version 3.10 or later. Adhesive has not been tested on a RISC-PC yet but the (few) system calls it performs are well documented under the RISC OS 3.10 programmers reference manuals — there should be no problems on a RISC-PC.

Will it clutter up my RMA space?

Adhesive itself is not very large (about 20K). Adhesive uses little workspace and only claims RMA space in 16K chunks to minimise the amount of fragmentation which occurs.

Why share code?

Sharing code has several advantages:

- Less memory is used if many applications are sharing code.
- Loading time is reduced if the shared code is already in memory.
- Better software engineering practices result, better modularisation can occur and software upgrades to a commonly used piece of code do not require many programs to be upgraded.

Why use Adhesive objects and not relocatable modules?

- Adhesive objects are smaller than the equivalent relocatable module.
- You do not require to get a module name from Acorn.
- Adhesive objects are automatically removed from memory when they are no longer in use.
- Relocatable modules need to contain supervisor SVC mode code, Adhesive objects can be written in mode independent / USR mode only code - resulting in better trap handling.
- Calling procedures in Adhesive objects is much faster then an SWI. (Only one/two branch instructions).
- If you use the DDE it is **very** simple to create Adhesive objects.
- New versions of objects can 'inherit' older versions.

Terminology

The following terms are used throughout this documentation and in many of the examples supplied with Adhesive.

object — An object is a collection of procedures which are grouped together. These procedures can be shared between many *users*. An object can gain access to procedures in other objects including earlier versions of itself.

user — A user is an application or relocatable module which has access to *objects* via Adhesive.

Note the terms *object* and *user* are mutually exclusive.

procedures — A procedure is a piece of code in an object. A procedure can have any entry/exit parameters and register bindings it wants (though you are recommend to use APCS as used by the C compiler). Adhesive is only interested in an address in memory to call to execute the procedure.

request — A *user* requests an *object* using Adhesive, *objects* can request other *objects* using Adhesive.

How it works

When Adhesive initialises it keeps a record of which objects are installed. A user then requests an object, Adhesive will load the object and any objects which are in turn required, Adhesive then creates branch instructions between the user and the object, and between objects and other objects if needed. The user simply then branches to the code in the object. (This is a branch table, like that used by the Shared C Library.)

Adhesive takes care of finding a suitable version of an object and finding the objects which that object depends on. Circular dependencies are allowed, indeed an object may refer to itself or an older version of yourself allowing **inheritance.**

When a user 'dies' Adhesive removes all objects which are no longer needed from memory.

Each object is kept in a directory along with any resources it needs. Adhesive provides easy message lookup for objects which require a messages file.

Installation

The directory 'tools.bin' contains some executables. These should by placed somewhere in your system which is in your Run$Path (e.g. library or bin directory). The application 'adhesive.!Adhesive' should be placed in a place where it will be 'seen' by the filer when your machine starts up. You can do this by adding a line Filer_Run .....!Adhesive to your !Boot file or if you have a RISC-PC you can place it in the boot directory.

If you have the Desktop Development Environment (DDE) then you may wish to use the supplied tool !CAHG (more on this later). If so you will need to place the lines below (with no extra blank lines or spaces) into your '!Make.tools' file. (Remove the ---CUT--- lines.)

```
--CUT--
CAHG
cahg
-u -t
cahg $(cahgflags) $< -o $@
DDE:!CAHG.desc
DDE:!CAHG.!setup
--CUT--
```

You should now reboot your machine (to ensure the newly installed copy !Adhesive is used rather than the one in the distribution archive).

If you distribute the copy of Adhesive in the 'user' directory to others with your software, please ensure that you provided appropriate installation instructions or installation program.

!Adhesive

!Adhesive is an application which launches the Adhesive module. This involves loading the module and initialising Adhesive. During this initialisation it is possible that some errors will occur. If so all errors are logged in the !Adhesive.errorlog file and the error "Adhesive encountered one or more minor errors during initialisation,see the !Adhesive.errorlog file for details" will be reported.

Adhesive will still continue to run, however it is likely that these errors were caused by corrupt/invalid objects and so not all objects may be available. If a 'serious' error should occur it will be reported and the initialisation program will remove the Adhesive module from memory.

Support for different languages

The ideal environment to use Adhesive in is the Desktop Development Environment (DDE). If you do not have the DDE do not despair, examples written in BASIC are supplied.

APCS

APCS (Arm Procedure Call Standard) is a defined way of calling procedures from one language to another. The RISC-OS programmers reference manual describes this. You are advised to make procedures in objects APCS conformant (in fact APCS-R conformant) so that your objects can be used from C,Pascal,Assembler,BASIC and other languages.

To use APCS you need to have initialised with the Shared C Library, see the section below on BASIC on how you can easily do this.

DDE (Desktop C,Pascal,Assembler)

The directory ‘tools’ contains two DDE tools, !CAHG (C Adhesive Header Generator) and !Shorten. These tools are described in more detail later, choosing the Help option from the tools provides more information.

CAHG takes a short file describing which procedures your object offers and needs, or what objects your user needs. It generates a .o file ready for linking which contains information which Adhesive needs at run time. CAHG has support for !Make.

Also supplied are two .o files, ‘tools.o.Adhesive’ and ‘tools.o.AdhesiveRS’. These come with corresponding C header files in the ‘tools.h’ directory. These allow you to access Adhesive SWI’s from C, ‘Adhesive’ is for users and ‘AdhesiveRS’ for objects. These objects files also work fine from Pascal and Assembler.

When compiling objects you should NOT link with ‘stubs’, you should not choose the module option for CC but MUST choose the module option for Link.

BASIC

The directory ‘examples’ contains an example object written in BASIC. The directory ‘goodies’ contains an object (basicC) which allows you to call APCS conformant procedures from BASIC (such as the Shared C Library or other objects written in C). Documentation and examples of this object are provided in the ‘goodies.basicC’ directory.

Beebug Easy C

You can create users and objects using Easy C though you will need to use a different linker for the following reasons:

- The Easy C linker does not support a ‘module’ option as so cannot generate relocatable code — objects must be relocatable.
- The Easy C linker does not like the .o files produced by CAHG. (The DDE has no problem.) I suspect this is because the type/version of .o file produced by cahg is of too recent a version.

This applies to Easy C module version 1.99, I have not tested later versions. You can either use Acorn’s Link or you can try using DrLink (sorry but I have not tried DrLink). I have had no problems when using Acorn’s Link.
You will have to run CAHG by hand from the command line.

GNU GCC

You can create users and objects using GCC. You will need a linker which produces relocatable code (ie has a ‘module’ option). I have had no problems when using Acorn’s Link. You will have to run CAHG by hand from the command line.

Objects

Objects are contained in a directory along with their resources. The directory name indicates to Adhesive which object the object is (a unique identification number is used) and which version it is. No two objects installed at the same time may have the same object identification number and version number.

Object identification number and version number

When you register (see the file 'register') you will be given a range of object identification numbers you can use (64 in fact). The following identification numbers are reserved: (All numbers are decimal.)

0	Not valid
1	Gives access to the Shared C Library (see The Shared C Library)
2-63	Reserved for testing and debugging purposes and the supplied examples.
64-127	You may use these object numbers whilst waiting for me to respond to your registration form and fee. Do not give objects using these numbers to others.

Generally speaking each object is kept in a directory with the name `xxx.vvv` where `xxx` is the objects identification number and `vvv` is the version number. These numbers should not have leading zeros and are in decimal and must not contain alphabetical characters. Version 1.00 has a `vvv` of `100`. The version number must be greater than zero.

Only the directory name identifies the object number and name.

The object directory

Each object must have the following files in the directory:

`xxx.vvv.obj`	This file is not optional and must be present.
`xxx.vvv.info`	This file is not optional and must be present.
`xxx.vvv.messages`	This file is optional.

The 'obj' file is the object itself, it is simply a piece of ARM code starting with a special Adhesive object header. Typically run time relocation code produced by Link is found at the end of the object. The object must not be compressed, it is expected to run in an amount of memory equal to the file size.

The 'info' file describes the object. It must contain at least four lines, a title, an author, some version information and then a blank line. The blank line MUST be present for future compatibility. For example:

```
Wimp Interface Object
George Taylor
Release 1.00, beta version, 10th January 1999

All lines after the above blank line are ignored.
```

Object resources

If your object has a messages file it will be loaded by Adhesive when your object is loaded. You can access the messages using `adhesive_Messages()`, `adhesive_Error()` from C or you can use SWI Adhesive_Messages, Adhesive_Error. Do not supply a messages file unless you need one — even an empty file will waste memory.

You can obtain the name of the directory your object is in (regardless of its number and version) using `adhesive_Resource()` or SWI Adhesive_Resource. Use this if your object requires resources such as sprite files, templates, lookup tables etc. You may use any file names you wish for these including directory structures.

Installing and removing objects

When you create a new version of an object you MUST give it a HIGHER version number. Adhesive will not allow you to replace an object/version combination which already exists without explicitly deleting the old one first.

To install an object you can use

```
*Adhesive_InstallObject <filename>
```

where <filename> is of the form '`path.xxx.vvv`' which is a directory containing the object and its resources. The directory is copied recursively if the object is installed. No action (or error is reported) if the object number/version is already installed. See also SWI Adhesive_InstallObject.

To remove an object you can use

```
*Adhesive_RemoveObject xxx vvv
```

where `xxx` is an integer object identification number and `vvv` is an integer version number (use `100` for V1.00). An error is given if the object is not installed or if the object is currently in use.

The Shared C Library

Users of Adhesive objects written using C,Pascal,ObjAsm should use the Shared C Library via the 'stubs' file provided with the DDE.

Users of Adhesive objects written in BASIC should use the basicC object provided in the 'goodies' directory. Machine code programs written using the BASIC assembler will have to register with the Shared C Library themselves. (An example of this is supplied in the Adhesive Objects distribution also written by George Taylor, alternatively you can use the example given in the PRM's. No example of this is currently provided with Adhesive).

Objects written in any language should not register with the Shared C Library. This is because objects are intended to behave as library extensions to programs. A program written in C will use APCS. This program can then call an object which then calls the Shared C Library using the environment set up by the program. You should do this by requesting object 1 (a special object built in to Adhesive which gives direct access to the Shared C Library). This means when you open a file the C `FILE*` descriptor is kept in the user programs memory space - as you would expect to happen. `malloc()` and `free()` also behave correctly for example.

Notes:

- You do not need to check the Shared C Library is loaded, it must be for Adhesive to work. This means object 1 can always be requested (provided Adhesive is loaded).
- The minimum and maximum version of object 1 are ignored. You will get whatever version of the Shared C Library is currently loaded.

Procedures and how Adhesive chooses which version of an object to use

Procedures are a piece of code with any entry/exit definitions you care to use though APCS-R is the preferred system for maximum compatibility and code reuseability. Procedures within an object are given entry point numbers, these are 32bit integer numbers and include 0. Entry point numbers do not need to follow any order and are local to a particular object identification number (but must be consistent across versions of the same object). (Your object will be slightly shorter if the numbers are contiguous.)

Identifying procedures by numbers also means that if you do not like the procedure names that come with an object (then provided you change the C header file if using C) you can use whatever names you like for the procedures.

An object defines which entry points it offers. This is defined in a special header at the start of a object along with details of which other objects the object needs. (CAHG will generate this header for you.) A user must specify which entry points it requires when requesting other objects.

When you request an object you specify which object, which range of versions, and which entry points. Adhesive will give you highest version of the object possible such that

- The version provides all the entry points asked for.
- The version is greater than *or equal to* your specified minimum version.
- The version is less than *or equal to* your specified maximum version.
- The object loads and initialises successfully.

This mechanism ensures it is possible to both add new procedures to newer versions of objects and to remove obsolete procedures from newer versions *whilst still allowing programs* to run. (The older program doesn't even need to know about the new object version, it simply requests the highest version possible and gets the old version as the new version has had some obsolete procedure removed).

If the object does not load successfully (due to a reason other than lack of memory) then Adhesive tries again with a lower version of the object if a suitable one is available. This process can repeat itself as many times as possible.

If an object fails to load (due to a reason other than lack of memory) then no attempt is made to try and load the object again until a new object is installed or Adhesive is restarted. This is because:

- If the object failed to load due to reporting an error (eg missing resource file) or the object being corrupted there is no point in trying to load it again - it will only fail.
- Most objects fail to load because they require other objects which are not installed. There is no point in trying to load the object again until new objects have been installed.

The tool 'tools.bin.AdhInfo' is particularly useful in finding out the error status of objects.

The dummy object 1 provides access via this mechanism to the Shared C Library.

Writing objects

The examples provided both in this distribution and in the separate distribution of freeware objects written by George Taylor are the best way of learning how to write objects. If you are using CAHG you should read the !CAHG.!Help file before using CAHG.

There are however a few important points to note:

- Global variables are global to your object regardless of the user who calls you.
- If you use an APCS (C like) environment, then the environment the object 'sees' is the same as the user currently calling the object. (For example if you use `fopen()` then the returned file descriptor will be in the user's memory space NOT yours.)
- I believe there may be a problem of zero-initialised data areas not being initialised. Assume that global variables are not initialised to 0 unless you specify an initialisation value when you declare the variable.
- Never choose the 'module' option on the C compiler.
- ***Always choose the 'module' option on the linker. If you don't your object will crash.***
- Objects must not register as a user with Adhesive.
- Objects must not be squeezed using !Squeeze.

- If you wish to inherit an older version of yourself simply request versions greater than some minimum which is less than your own version.
- See also 'abstract types' below.

Abstract types - problems that can arise - warning this is complex!

Later versions of objects must provide an interface which is compatible with earlier versions. If you use `malloc()` to claim space, pass back a pointer to abstract data stored at this space. You need not keep the abstract data format the same between versions of objects.

Important point illustrated by a (complex) example:

i)	object C, version 1 provides an abstract hash table type (it uses say open addressing).
	For example:
	`typedef void *hashtable;`
	`hashtable add(hashtable h,char *string)`

ii)	suppose object A uses C to provide hashing of strings
iii)	suppose object B also uses C to provide hashing of strings
iv)	the procedures in A & B take/return an abstract hash table pointer which is actually passed straight to C
v)	user U requests A
vi)	Adhesive loads A and C version 1 and links Cv1 to A and A to U.
vii)	someone installs object C, version 2 which whilst providing the same call interface now uses probing instead of open addressing
viii)	user U requests B
ix)	Adhesive loads C, version 2 (its the latest version) and links Cv2 to B and B to U.
x)	user U calls a procedure in B with a hash table he created from A.
xi)	something nasty happens because Cv2 uses a different data representation to Cv1.

Alternatively A and B could specifically request different versions of C and the same problem would occur.

Conclusion:
When you request an object you must assume it's abstract types are different from other requests of the object (including those indirectly through other objects).

The way to avoid this problem all together is not to 'share' abstract types between objects. (Which with abstract types is generally a bad thing to do anyway.)

[Personally I much prefer the SML module system, it has proper data types and a secure type system.]

Similarly if you place more than one request for the same object in an object request table they may not be linked to the same version.

Examples

There are examples in the 'examples' directory, there is also an example in 'goodies.basicC'. The examples include documentation.

Tools

The following tools are provided with Adhesive, they may be found in the 'tools.bin' directory. CAHG and Shorten also come with DDE frontends.

CAHG	C Adhesive Header Generator. Generates the request and offer blocks described in 'Technical Bits' from a simple text file description.
Shorten	Remove the relocation code produced by Link from the end of an object if the object contains no relocations. This will simply make the object a little bit shorter if possible. It also change the object's filetype from 'module' to 'data'.
KillAll	Makes Adhesive think all users have died and removes all objects from memory. This can be useful when debugging your own users and objects and Adhesive thinks an object is still in use because a user did not exit properly.
AdhInfo	Display information on all installed objects and currently running users.
AdhObjInfo	Displays full information on an object or on all installed objects.

The file 'tools.readme' contains more details on these tools.

Technical bits

This section describes the format of an object header (at the start of an object) and describes the SWIs supplied by Adhesive.

Some of the SWIs can be accessed from an APCS (C like) high level language using the 'tools.o.Adhesive' and 'tools.o.AdhesiveRS' files. If you are writing in C/Pascal/Objasm you are advised to use the .o files as it makes things simpler. You are also advised to use !CAHG which will generate much of the 'data blocks' described below for you from a simple text file.

Important notes:

- All fields marked reserved for future use must be 0 when supplying data to Adhesive.
- All Adhesive SWIs may be called from applications and modules unless indicated otherwise.
- Adhesive SWIs are not re-entrant and must not be called from interrupts unless specified otherwise. The interrupt status is undefined.
- Unless indicated otherwise all Adhesive SWIs may generate an error (use the X version to trap the error).
- Registers not listed below as part of the exit status from Adhesive SWIs are preserved.
- All strings must be 0 terminated.
- All parameter blocks must be word aligned. Strings can be byte aligned.

SWIs

The SWIs below are offsets from Adhesive's SWI base number of &4A140.

Adhesive_Register **&00**

entry:	r0 = pointer to user information block (see later), a copy of this is made so you may dispose of it after this call
exit:	r0 = handle
use:	Register a new user with adhesive. The returned handle is non-zero. You can register as a user as many times as you like - once is normally enough - this is useful if you wish to only use some objects during program startup.

Adhesive_Deregister **&01**

entry: r0 = handle (as returned by Adhesive_Register)
exit: no exit parameters
use: Tell Adhesive you are no longer using previously requested objects.
After making this call you must not make any further calls to objects requested on the handle or pass the handle to any SWIs.

Adhesive_Request **&02**

entry: r0 = handle
r1 = pointer to request block (see later), a copy of this information is made so you may dispose of the block after the call.
exit: no exit parameters
If this call does not give an error you may now call procedures in the requested objects.

Adhesive_Resource **&03**

entry: r0 = pointer to the start of your object header
exit: r0 = pointer to read-only directory name
use: Ask Adhesive for the name of the directory your object is in. It will end in xxx.vvv with a 0 terminator.

You must not write to the returned string, doing so may cause improper operation later.

This SWI is re-entrant and the string returned will be valid as long as your object is loaded. Do not refer to the string after your object's finalisation entry has returned.

Typically you would use this SWI to access a resource file your object requires.

This call must only be made by an object which is currently in memory.

Adhesive_KillAll **&04**

entry: no parameters
exit: all register preserved
use: Deregisters all users and removes all objects from memory.
Do not use this SWI whilst there are users actively using objects.
This SWI is useful when debugging objects.
The program 'tools.bin.KillAll' uses this SWI.

Adhesive_ToolSupport **&05**
Adhesive_Init **&06**
Adhesive_Debug **&07**

These SWIs are for internal use by Adhesive and the support tools only. You must not use them in your own programs, undefined results may occur if you do.

Adhesive_InstallObject **&08**

entry: r0 = pointer to directory name of object to install
exit: r0 = 0 if object already installed, 1 if object just installed
use: Installs a new object, the object directory is recursively copied and the object made ready for use. No action is taken if the object/version is already

installed.

Adhesive_RemoveObject &09

entry: r0 = object number
r1 = version

exit: no exit parameters

use: Removes an object if it is not in use.

Adhesive_Messages &0A

entry: r0 = pointer to the start of your object header
r1 = pointer to message token (zero terminated) (may have default :string)
r2 = pointer to buffer for result, 0=>no buffer
r3 = size of buffer (ignored if r2=0)
r4...r7 = pointers to parameters %0...%3 (ignored if r2=0)
(you may miss out registers which correspond to parameters not used in the messages file)

exit: r0 = pointer to buffer or pointer to read-only message if r2=0 on entry
0=> no message file found or tag not found
The message is terminated with a 0 if r2<>0 on entry. If r2=0 on entry the message is terminated with a ctrl character. (MessageTrans does this.)

use: Lookup a message from the object's messages file. You may supply a default message after the tag, e.g. tag:default.

You must not write to the returned string if r2=0 on entry doing so may cause improper operation later.

This SWI is re-entrant and the string returned at r0 if r2=0 on entry will not be overwritten as long as the object is loaded.
Do not refer to the string after your object's finalisation entry has returned.

This call never returns an error but you should use the X version anyway - in case an address exception or similar occurs.
This call must only be made by an object which is currently in memory.

Adhesive_Error &0B

entry: r0 = pointer to the start of your object header
r1 = pointer to word aligned error block containing error number and zero terminated message token.
r2...r5 = pointers to parameters %0...%3
(you may miss out registers which correspond to parameters not used in the messages file)

exit: r0 = pointer to read-only error block containing error number and message
If the tag could not be found a pointer to the error block passed in r1 on entry is returned. This is a MessageTrans internal error block, note there are ten of these for foreground processes and two for IRQ processes.

use: Lookup a message from the object's messages file but using an error block.

This SWI is re-entrant. The error block's contents may change if further calls

to the message trans module are made. See the documentation on MessageTrans in the programmers reference manual for more information on message trans internal error buffers.

Typically you would use this SWI to return error messages during your initialisation entry.

This call never returns an error but you should use the X version anyway - in case an address exception or similar occurs.
This call must only be made by an object which is currently in memory.

Parameter blocks

There are three parameter block types, an object header, a request and a user information block. Some of these include another structure known as ‘pairs’.

User information

offset from start of block	description
0	flags, currently none used, must be 0
4	Pointer to zero terminated string identifying who you are, this is used for information purposes only and need not be unique.

Object Header

An object (both the file on disk and in memory) have the following layout:

offset from start of object	description
0	B __RelocCode or 0 if object has no relocation code
4	B initialisation code or 0 if none
8	B finalisation code or 0 if none
12	reserved for future use, must be 0
16	reserved for future use, must be 0
20	reserved for future use, must be 0
24	reserved for future use, must be 0
28	reserved for future use, must be 0
32	number of branches in branch table
36	number of pairs
40...	pairs
...-...	branches (branch table, you supply this)
...	requests (see below)
...	rest of object (e.g. code!)

Note the branch table above MUST contain branch (B) instructions. This is because Adhesive decodes the instruction to find the address of the procedure. (Branches do not produce relocations in compiled code and so don’t add to the object size - an absolute address would do this. Offsets are not practical accross multiple .o files.)

Request

offset from start of request	description
0	object identification number or 0 to mark end of table
4	minimum version (e.g. 100 for 1.00), this may be 0
8	maximum version (normally 0 (ie no max) except for an object inheriting and earlier version of itself)
12	reserved for future use, must be 0
16	number of branches to be placed in branch table
20	number of pairs to follow

24...	pairs
...-...	branches (branch table filled in by Adhesive)

More than one request can be placed after another but note that a zero word must always mark the end of the table.

Pairs

Pairs provide a compact means to represent a range of numbers. A pair consists of two integers, for example:

1
5
9
10
12
12

These two pairs indicate entry points 1,2,3,4,5,9,10,12 are offered/requested. The pairs must be in ascending order and must be in ‘simplest’ form. So for example,

0
5
6
8

are not in simplest form. (It can be written 0-8).

Relocation entry point

An object has a relocation entry point to enable non-relocatable code to relocate itself. This relocation code is typically the __RelocCode produced by Link. It is called before any other entry point into the object and before any procedures in the object are called.

__RelocCode is called when an object is loaded, it must not call any other objects or Adhesive SWIs.

entry:	SVC stack at R13 (processor is in SVC mode) return address in R14
exit:	You may corrupt all registers.

Initialisation entry point

The initialisation code is called after the __RelocCode entry has been called but before any procedures in object are called. It must not call any other objects (as your object may not have been linked yet). You may not call Adhesive SWIs other than Adhesive_Resource, Adhesive_Messages and Adhesive_Error.

entry:	SVC stack at R13 (processor is in SVC mode) return address in R14
exit:	R0 = return value 0 => no error -1 => not enough memory non-zero => pointer to word aligned RISC-OS error block (max of 256 bytes: 32bit number, zero terminated string) You may corrupt all other registers.

Note this is not an APCS register setup, be very careful if you write this entry point in C. You are advised to write this in assembler. If you know you have failed to initialise because of a lack of memory (and not a missing file or other error) then please return -1. This means Adhesive will assume it is not worth trying lower versions of the object and it is worth trying this version again later.

Finalisation entry point

The finalisation code is called before your object is removed from memory. It must not call any other objects (as they may already have been removed). You may not call Adhesive SWIs other than Adhesive_Resource, Adhesive_Messages and Adhesive_Error.

entry:	SVC stack at R13 (processor is in SVC mode)
	return address in R14
exit:	no return value
	You may corrupt all registers.

Note this is not an APCS register setup, be very careful if you write this entry point in C. You are advised to write this in assembler.

Final notes

- Adhesive does not support *RMTidy. Do NOT use RMTidy when Adhesive is running.
- The supplied binary ‘tools.o.KillAll’ will kill all users and remove all objects from memory without any safety checks.
- Once an object is loaded in memory it will not be removed until it is no longer in use. It will not be relocated once loaded.
- Upon a Service_Reset occurring Adhesive will assume all users have died and will remove all objects from memory. Modules must reregister upon a Service_Reset.

The Adhesive module name, SWI chunk name, SWI chunk number and the environment variables listed below have been allocated from Acorn.
The SWI chunk name (Adhesive), SWI names (see above section) and the SWI chunk number (&4A140) are never going to change. This means software which depends on these names will continue to word across all future versions of Adhesive.

Environment variables used by Adhesive:

Adhesive$ErrorFile	All Adhesive errors are logged to this file
Adhesive$InstallPath	New objects are placed in this directory
Adhesive$ObjectPath	Objects are kept in this directory.

The reason for having a separate InstallPath from ObjectPath is so that multiple directories can be placed in the ObjectPath (e.g. if your filing system has a limit on directory entries). (Writing to a path with multiple entries produces an error.)
Do not use these variables in your own programs. Adhesive's use of them may change in future versions.

You may however use <Adhesive$ObjectPath> in your C header files (or such like) to access header files which may be contained within an object.

You may also change the !Adhesive.!Run file to suit your system. (Do not modify the copy you give to others though.) You can also change the !Adhesive.!Boot file to alter the way Adhesive boots.