RISC OS Networking

Example manual for ROUGOL

Contents

Networking

Indexes

Selected chapters for Networking extracted as a demonstration of the RISC OS PRMs using PRM-in-XML.

Freeway

Introduction and Overview

Freeway is a narrow software layer which maintains information about the location of objects on a network, in a decentralised fashion.

Objects are classified according to Freeway Type. At the Freeway level the attributes of an object are its name, location, an optional descriptor and an optional authentication value. All other characteristics are undefined.

Freeway itself only knows about the IP location of objects on the network. It is not concerned with how those objects may be accessed, or with any specific access control mechanisms. These areas are handled by higher-level type-specific application software, as needed. Each type will have a 'controlling' application.

Freeway types are created by Acorn as required : third parties may request a type via the usual defined channel for allocation requests. Each type has a unique identifier : for example, if there is a need for software to enable 'Bach partitas' to be shared across a network, then an available Freeway type number will be allocated by Acorn - e.g. 97=BachPartitas - and then the controlling application will be written to 'share' Bach partitas - whatever that means.

Technical Details

Type number

This is a 16-bit integer identifying a Freeway type. The following type numbers are allocated to Acorn Access+. (Informal names with no significance in software will also be allocated, for easier human reference)

Type	Informal name
1	Discs
2	Printers
5	Hosts

In Acorn Access, ShareFS is the controlling application for types Discs and Hosts, and !Printers is the controlling application for type Printers.

Object name

This is the name of an object.

An object name is unique within the set of objects of the same type within a site. Object names are terminated by a control character (and will return a terminator of zero on output), case independent, alphanumeric strings, created by users or application software. Maximum length is 64 bytes, including the terminator.

Object descriptor

This provides extra information about the object.

Its format is undefined at this level - it has significance to higher-level application software. Object descriptors are arrays of bytes of any value, maximum length 255.

In Acorn Access+ the descriptor for type Discs indicates whether a disc is shared protected or unprotected, or as a CDROM or subdirectory, and visible or invisible to other desktop users.

The descriptor for type Printers is a 'Printer Type' string.

The descriptor for type Hosts is null.

Object authentication value

Objects are either authenticated or unauthenticated. Authenticated objects require an authentication value to be provided before their attributes are made visible. An authentication value is a 32 bit integer. The authentication value of zero is reserved and should not be used.

Refresh interval

The Freeway software in each computer will periodically re-notify other computers about unauthenticated objects which are held by the local machine, so that they know that the objects continue to be accessible. This permits, for example, other machines to know when a computer holding Freeway objects has been powered down. The renotification period is called the "refresh interval", and is determined by the Freeway software and is typically 30 seconds.

Service calls

Service_FreewayStarting (Service Call &95)

Freeway is starting up

On entry

R1 = service call number
R2 = flags :

Bit(s)	Meaning
0-31	Reserved, must be 0

On exit

R1 preserved

Use

This service is issued when the Freeway module starts up. You may not call any Freeway SWIs during this service. If you wish to register with the module you should use a transient callback to do so.

Related services

Service_FreewayTerminating (on page 6)

Service_FreewayTerminating (Service Call &96)

Freeway is terminating

On entry

R1 = service call number
R2 = flags :

Bit(s)	Meaning
0-31	Reserved, must be 0

On exit

R1 preserved

Use

This service is issued when the Freeway module is killed or shuts down for any other reason. You may not call any Freeway SWIs during this service.

Related services

Service_FreewayStarting (on page 5)

UpCalls

UpCall_Freeway (UpCall &C)

State of Freeway objects has changed.

On entry

R0 = UpCall number
R1 = Reason code :

Value	Meaning
0	Object has been added
1	Object has been removed
2	Object attributes have changed
3	Local object deleted by Freeway

R2 = type number
R3 = pointer to object name, 0 terminated
R4 = length of object descriptor
R5 = pointer to object descriptor
R6 = IP address of device which holds or held this object

On exit

R0 - R6 preserved

Use

This call warns controlling applications that the named object has been added, removed or changed. It is also issued when Freeway has detected that the information held about a locally held object is unreliable, (as a result of possible temporary name duplication, for example) and has removed it unilaterally. It is issued on callback, and the contents of the supplied name and descriptor buffers are guaranteed not to change provided they are read from within the application's UpCall handler.

Related SWIs

SWI Freeway_Write (on page 10)

SWI calls

Freeway_Register (SWI &47A80)

Register or deregister interest in objects of a given Freeway type.

On entry

R0 = flags :

Bit(s)	Meaning
0	0 - register interest
	1 - deregister interest
1	0 - interested in unauthenticated objects
	1 - interested in authenticated objects
2	0 - R3 not valid
	1 - R3 is pointer to type 'name' (<= 15 characters, 0 terminated)
3-31	reserved - must be set to 0

R1 = type number
R2 = authentication value if R0 bit 1 is set, otherwise undefined
R3 = pointer to type 'name' if R0 bit 2 is set, otherwise undefined

On exit

R0 - R3 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI enables a controlling application to register interest in authenticated or unauthenticated objects of a particular Freeway type with the local Freeway software, and also to give a 'name' to that type (this name is of no significance to the software, it's just for the convenience of the human user). Freeway will hold information arriving from the network about a remote object only if one or more registrations of interest have been made locally in that object's type. If the object is authenticated then at least one of those registrations must have included an authentication value which matches the object's own. An error is returned if insufficient free memory exists.

Related SWIs

SWI Freeway_Write (on page 10)
SWI Freeway_Read (on page 12)

Freeway_Write (SWI &47A81)

Add or remove an object of a given type.

On entry

R0 = flags :

Bit(s)	Meaning
0	0 - add object
	1 - remove object
1	1 - object is authenticated
2-31	reserved - must be set to 0

R1 = type number
R2 = pointer to object name, 0 terminated
R3 = length of object descriptor
R4 = pointer to object descriptor, or 0 for null descriptor, or when R0 bit 0 is set
R5 = object authentication value if R0 bit 1 is set, otherwise undefined

On exit

R0 - R5 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI adds or removes a locally held object of a given type. If the object is unauthenticated then other computers are notified immediately, otherwise notification is withheld until a remote computer requests it. An error is returned if the type number is not known (i.e. if Freeway_Register has not been called with this type number), or if an object is added and a remote object of the given name and type already exists, or if an object is removed and no local object of the given name and type is currently held, or if no memory exists to store information about the object.

If R0 bit 0 is clear and the object named is already held as a local object, the object's descriptor and authentication value are updated if they differ from those passed to the SWI.

N.B.: Controlling applications should be prepared to receive upcall UpCall_Freeway, reason code object deleted, referring to any object previously added successfully via SWI Freeway_Write. This is to cover the possibility of Freeway deciding at any time that the information held about an object is unreliable and so deciding to remove it unilaterally, for any reason, e.g. if a remote object is created with the same name as a local object.

Related SWIs

SWI Freeway_Read (on page 12)

Freeway_Read (SWI &47A82)

Read attributes of an object.

On entry

R0 = flags :

Bit(s)	Meaning
0	1 - authentication value provided
1-31	reserved - must be set to 0

R1 = type number
R2 = pointer to object name, 0 terminated
R3 = length of buffer for object descriptor
R4 = pointer to buffer for object descriptor, or 0 to read descriptor length
R5 = object authentication value if R0 bit 0 is set, otherwise undefined

On exit

R0 - R2 preserved
R3 = length of held object descriptor
R4 preserved
R5 = IP address of computer which holds the object

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI reads information about the attributes of an object. The type number and object name must be provided. The SWI returns the IP address of the holder, and optionally the descriptor. The length of a held object descriptor may be read by setting R4=0 on entry. However in this case there is no guarantee that the object attributes will not have changed, or that the object will still exist, if the SWI is called again some time later with the same object name. An error is returned if the type number is unknown, or the object name is unknown, or if a supplied authentication value does not match the object's own authentication value, or if a supplied object descriptor buffer is too short; in this case the actual length is returned in R3.

Related SWIs

SWI Freeway_Write (on page 10)

Freeway_Enumerate (SWI &47A83)

Enumerate objects of a given type.

On entry

R0 = flags :

Bit(s)	Meaning
0	0 - enumerate unauthenticated objects 1 - enumerate authenticated objects
1-31	reserved - must be set to 0

R1 = type number
R2 = length of buffer for object name
R3 = pointer to buffer for object name, or 0 to read length of name
R4 = length of buffer for object descriptor
R5 = pointer to buffer for object descriptor, or 0 to read descriptor length
R6 undefined
R7 = enumeration context (0 to start)
R8 = object authentication value if R0 bit 0 is set, otherwise undefined

On exit

R0 preserved
R1 preserved
R2 = length of held object name, including terminator
R3 preserved
R4 = length of held object descriptor
R5 preserved
R6 = IP address of computer which holds the object
R7 = next enumeration context, or -1 if no more
R8 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI allows a controlling application to enumerate currently held authenticated or unauthenticated objects of a given type, obtaining their names, location IP addresses, and descriptors if present. If an authentication value is provided then only those objects whose actual authentication value matches the supplied value are enumerated. If no authentication value is supplied then only unauthenticated objects are enumerated. The length of the held object name or descriptor may be read without filling in buffers by setting R3=0 or R5=0 respectively, on entry. However in this case there is no guarantee that the object attributes will not have changed, or that the object will still exist, if the SWI is called again some time later with the same enumeration reference.

If R7 is returned -1 then there were no further known objects of that type - the object name buffer will not have been filled in, and R6 is undefined.

An error is returned if the type number is unknown, or if a supplied name or descriptor buffer is too short. In the latter cases the actual name and descriptor lengths are returned in R2 and R4.

Related SWIs

SWI Freeway_Read (on page 12)

Freeway_Status (SWI &47A84)

This SWI call is for internal use only. You must not use it in your own code.

Freeway_Serial (SWI &47A85)

This SWI call is for internal use only. You must not use it in your own code.

*Commands

*FWShow

Show currently known unauthenticated objects

Syntax

```
*FWShow
```

Parameters

None

Use

This command is used to show the names and holder IP addresses of all unauthenticated objects of all types currently known about by this machine. Local objects are indicated via a leading asterisk.

Examples

```
*FWShow
```

Related SWIs

SWI Freeway_Enumerate (on page 13)

Document information

Maintainer(s): RISCOS Ltd <developer@riscos.com>

History:

Revision	Date	Author	Changes
1		ROL	**Initial version**

Resolver

Introduction and Overview

Internet based applications are required to connect to other hosts throughout the world. Location of these hosts is via numeric addresses which are determined by human readable names. The process of converting from names to addresses is performed by 'Domain Name Servers' (DNS).

Under RISC OS, communication with DNS is performed by the Resolver module.

This chapter describes how the Resolver module is configured to use DNS, and applications should interact with it.

Terminology

The Resolver module provides shared Internet IP address resolution facilities to the system.

The Hosts file is the file InetDBase:Hosts. It is used to initialise the local DNS cache with entries.

DNS servers are queried by the Resolver and the results are cached.

Application is used within this chapter as a description of the Resolver querant, but this may also apply to Modules.

Technical Details

How the Resolver works

The Resolver functions as follows :

- The user configures the resolver through serial variables. Usually this is performed on machine start up from pre-set values.
- When an address is requested the application requests a lookup from the Resolver.
- The Resolver module checks its cached entries and if one exists that matches the request, it is returned immediately. Otherwise a request is sent to the configured DNS servers.
- The application continues to request a lookup from the Resolver. The Resolver continues to try to look up the request until it times out.
- If the request to the DNS servers is satisfied, or a failure is returned, the result is returned to the application.
- If the request is not responded to within a period, a failure is returned to the application.

SWI interface

The module provides two interfaces for resolving addresses :

- *SWI Resolver_GetHostByName (on page 33)* to provide an equivilent operation to that expected by unix applications. This SWI will not return until a result or failure has been determined.
- *SWI Resolver_GetHost (on page 34)* to provide a multi-tasking version of GetHostByName. This SWI returns a notification that the resolve is 'in progress' and should be polled until a result has been received.

In addition, there is a means for controlling the cache :

- *SWI Resolver_CacheControl (on page 37)* can be used to perform various operations on the cache.

Host entries

All results are returned in the form of NetBSD standard 'hostent' structures. These structures should be considered to be read only by the application.

A hostent has the following structure :

Offset	Contents
0	Pointer to host name
4	Pointer to 0 terminated list of pointers to aliases for this host
8	Address type (usually AF_INET)
12	Address length (4 for AF_INET)
16	Pointer to 0 terminated list of addresses

System variables

The Resolver module uses a number of system variables for its configuration. These determine how and where it resolves Internet addresses from. In general, these variables are initialised during machine startup. They may be modified by the user once the Internet stack has started. They will be read when the re-configure call is made to the Resolver, either via the Resolver_CacheControl SWI or by the *ResolverConfig command. In addition, they will be re-read whenever the InetDBaseChanged service is

issued.

System variables

Inet$Resolvers

Lists the DNS servers that should be queried

Use

This variable lists the DNS servers that will be queried during resolution requestes. It consists of a space separated list of dotted IP addresses. Up to three addresses may be supplied. Addresses will be queried in the order supplied.

Related APIs

None

Inet$Hostname

Local name of this host

Use

This variable gives the local name of this host. Attempts to set this variable to a fully qualified hostname will result in it and `Inet$LocalDomain` being set to the correct values.

Related system variables

Inet$LocalDomain (on page 25)

Inet$LocalDomain

Local domain name

Use

This variable gives the local domain name for the network this host is connected to. Together with `Inet$Hostname`, this provides the name of this host on a network. It may be that this host name is not visible from certain networks due to firewalls, proxying and other network-specific issues.

Related system variables

Inet$Hostname (on page 24)

Inet$SearchDomains

List of domains to use search hostnames in

Use

This variable describes the domains which should be appended to failed host lookups in an attempt to resolve them.

After being modified, the Resolver module must be notified by requesting it re-read its configuration.

Related * commands

*ResolverConfig (on page 38)

Related SWIs

SWI Resolver_CacheControl (on page 37)

Related system variables

Inet$LocalDomain (on page 25)

Inet$ResolverRetries

Number of times that the resolver should re-try resolving

Use

This variable should be set to the number of retries that the resolve should made. During resolution, the Resolver may get no response to its requests. This may be due to network bandwidth limitations, or a busy server. The number of retries may be configured to allow for greater numbers of retries if necessary.

After being modified, the Resolver module must be notified by requesting it re-read its configuration.

Related * commands

*ResolverConfig (on page 38)

Related SWIs

SWI Resolver_CacheControl (on page 37)

Related system variables

Inet$ResolverDelay (on page 28)

Inet$ResolverDelay

Delay between resolution requests

Use

This variable should be set to the number of seconds that should be waited between resolution requests. During resolution, the Resolver may get no response to its requests. This may be due to network bandwidth limitations, or a busy server. The delay between retries may be configured to allow for faster requests (if the server is known to accept such requests - consult the server operators) or longer delays if the server is known to be under load.

After being modified, the Resolver module must be notified by requesting it re-read its configuration.

Related * commands

*ResolverConfig (on page 38)

Related SWIs

SWI Resolver_CacheControl (on page 37)

Related system variables

Inet$ResolverRetries (on page 27)

Inet$ResolverServer

Determines whether the DNS relay is active

Use

This variable should be set to '1' to configure the DNS relay to be active, or '0' to deactivate it.

After being modified, the Resolver module must be notified by requesting it re-read its configuration.

Related * commands

*ResolverConfig (on page 38)

Related SWIs

SWI Resolver_CacheControl (on page 37)

Service calls

Service_InternetVars (Service Call &80C41)

Notification of variable changes

On entry

R0 = reason code :

Value	Meaning
0	*Service_InternetVars 0 (on page 31)*
1	*Service_InternetVars 1 (on page 32)*
other	Reserved

R1 = service call number

On exit

R0 preserved
R1 preserved

Use

This service is issued by the Resolver module when certain variables that it monitors change. Other components may use the service as an indication that their configuration has changed.

Related APIs

None

Service_InternetVars 0 (Service Call &80C41)

Change in the location of the internet database

On entry

R0 = 0 (reason code)
R1 = service call number

On exit

R0 preserved
R1 preserved

Use

This service is issued by the Resolver module when `InetDBase$Path` is changed. Components which hold resources in the database should re-read those resources.

Related services

Service_InternetVars (on page 30)

Service_InternetVars 1 (Service Call &80C41)

Change in the name of this host

On entry

R0 = 1 (reason code)
R1 = service call number

On exit

R0 preserved
R1 preserved

Use

This service is issued by the Resolver module when *Inet$Hostname (on page 24)* is changed. Components which use the hostname to refer to objects should update their concept of the local name. If complex operations are required, clients should schedule a transient callback.

Related services

Service_InternetVars (on page 30)

Related system variables

Inet$Hostname (on page 24)

SWI calls

Resolver_GetHostByName (SWI &46000)

Initiate a non-blocking name resolution

On entry

R1 = Pointer to host name string

On exit

R0 = Internet error number, indicating the state of the lookup :

Value	Meaning
0	Resolve successful
-1	Host was not found
-2	Remove request failure, eg no response from servers
36	Resolve in progress
other	Standard internet error numbers

R1 = Pointer to 'hostent' structure if successful, or 0 if failed

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is provided primarily for backwards compatibility, since new network applications can use the more flexible *SWI Resolver_GetHost (on page 34)*.

The main use of this SWI is by the simple ports of unix applications using the gethostbyname() function call.

Although this SWI is marked as non-reentrant, it is expected that this SWI be preempted by OS_UpCall 6 when used in a TaskWindow. This is a safe operation.

Related SWIs

SWI Resolver_GetHost (on page 34)

Resolver_GetHost (SWI &46001)

Initiate a non-blocking name resolution

On entry

R1 = Pointer to host name string

On exit

R0 = Internet error number, indicating the state of the lookup :

Value	Meaning
0	Resolve successful
-1	Host was not found
-2	Remove request failure, eg no response from servers
36	Resolve in progress
other	Standard internet error numbers

R1 = Pointer to 'hostent' structure if successful, or 0 if failed

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI provides a DNS resolution facility that runs in the background (unlike *SWI Resolver_GetHostByName (on page 33)*). Applications must poll the Resolver module for the state of the resolution.

If the hostname is a valid cache item, or present in the hosts file on disc, it is returned immediately in R1, with R0 set to 0.

If the hostname is a cache item, marked as failed, then R1 is set to 0 and R0 is either -1 (host not found), or -2 (remote failure. eg, configured resolver didn't respond).

If the hostname is a cache item, marked as pending, then R1 is set to 0 and R0 is EINPROGRESS (36 decimal). Items are marked as pending when a remote resolver lookup is in progress. The calling program is expected to periodically call Resolver_GetHost until a valid hostent is returned, or an error condition occurs.

Because name lookups can take anything up to 20 seconds or so, it is important to be able to give feedback to the user on the status of the lookup, as well as giving the foreground application time to perform other tasks.

Cache sweeps occur periodically - pending items are marked as failed (remote failure) after a short period, failed items are removed after a longer period, and valid items are removed after 24 hours.

If there is a configuration error (such as *Inet$Resolvers (on page 23)* not being set), then a RISC OS error block pointer is returned in R0, and the V flag is set on exit.

Examples

The calling program might run something like:

```
REM 'EINPROGRESS' error number is 36 decimal
REPEAT
  SYS "Resolver_GetHost","host.net" TO status,hostent;flags
  error = (flags AND 1) == 1
  REM Perhaps inform user of the state of this lookup
UNTIL (error) OR (status != EINPROGRESS)
REM hostent is a valid pointer, or 0
```

Related SWIs

SWI Resolver_GetHostByName (on page 33)

Resolver_GetCache (SWI &46002)

This SWI call is for internal use only. You must not use it in your own code.

Resolver_CacheControl (SWI &46003)

Perform operations on the Resolver cache

On entry

R0 = Reason code :

Value	Meaning
0	Flush cache of failed lookups
1	Flush cache of all items
2	Flush cache of hosts file items
3	Re-read configuration
8	Disable caching of failed lookups
9	Enable caching of failed lookups
other	Reserved

On exit

R0 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is provided to allow a calling application to control the Resolver cache in a limited way. Currently, the only reason codes supported are to force flushes of certain cache entries, and dis(abling) of the caching of lookups that fail.

Caching of lookup failures is disabled by default. You might wish to enable the caching of lookup failures if you are sure that any failures are genuine bad host errors and not due to packet loss, remote server timeouts, etc.

Related * commands

*ResolverConfig (on page 38)

*Commands

*ResolverConfig

Request that the resolver re-read its configuration

Syntax

```
*ResolverConfig
```

Parameters

None

Use

This command is used to request that the Resolver re-read the system variables for configuration.

Examples

```
*ResolverConfig
```

Related SWIs

SWI Resolver_CacheControl (on page 37)

Document information

Maintainer(s): RISCOS Ltd <developer@riscos.com>

History:

Revision	Date	Author	Changes
1		ROL	**Initial version**
2	22 Jan 2003	ROL	**Corrected unfinished sentence** • Documentation for Service_InternetVars had an unfinished sentence about clients using it.

The MimeMap module

Introduction

In order to determine the operations which can be performed on data, a system of 'typing' is used. On different operating systems and environments, different typing systems are employed:

- Under RISC OS, the typing system is known as 'file types' and is determined by numeric identifiers (the filetype of an object) in strict range (0-4095).
- Under DOS and Windows-like systems, the type system is known as an 'file extension' (sometimes known as a 'dot extension') and is determined by a string - traditionally of three characters - appended to the filename.
- Under unix-like systems, the type system varies but most commonly a similar system to that employed by DOS and Windows is used, or a system based on the content of the file.
- On Apple machines, the type is a fixed length binary identifier (8 bytes), split into two fields (4 bytes each) - the vendor identifier and the file type.

With this diverse selection of type systems in use and in order to try to standardise the manner in which such types are identified, a system of 'media types' (also known as the 'content type') was developed. This is colloqially known as the 'MIME type', from its use in the most widespread use of its application, Multipurpose Internet Mail Extensions. This media type categorises the data by category, providing a number of general categories into which data falls (for example, text or video).

The MimeMap module provides a central service for mapping between most of the types available. Of those described above, the MimeMap module does not presently cater for file identification by its contents.

Overview

When the module is initialised (either on being loaded, or during boot up) it reads the file *Inet$MimeMappings (on page 44)* and stores the parsed file internally. If the system variable is changed, or the module is explicitly requested, the old version will be discarded and the file reparsed.

The MimeMap module can handle 5 different file typing schemes:

- RISC OS file type by number, eg &FFF
- RISC OS file type by name, eg Text
- Media type string, eg text/plain
- File extension string, eg .txt
- Mac type/creator pair, eg "ttxtTEXT"

Any type can be converted to any other type by the module should such a mapping exist in the file. Dependant on the settings in the MimeMap configuration, certain conversions may not be possible. For some of the conversions, wildcards (fields identifying multiple potential matches) may be available.

Using the module

In order to ensure that type conversion is performed in a consistent and maintainable manner, it is recommended that type conversions are - whereever possible - performed through the MimeMap module. Typical applications which would require translations to be performed would be network file transfers (for example remote filing systems, FTP or HTTP transfers), non-native hardware transfers (for example CD-ROM data, floppy discs), and collated file processing (for example MIME format messages such as email).

As supplied, the RISC OS filing systems DOSFS and CDFS support the use of the MimeMap module.

Technical details

Type formats

Because the types supported by the MimeMap module are represented in different ways, a 'type format' is used to distinguish between each of the types in SWI calls.

Value	Name	Meaning
0	MMM_TYPE_RISCOS	RISC OS file type passed as an 32 bit number
1	MMM_TYPE_RISCOS_STRING	RISC OS file type passed as a pointer to a zero terminated string
2	MMM_TYPE_MIME	MIME content type passed as a pointer to a zero terminated string
3	MMM_TYPE_DOT_EXTN	File extension (without the preceeding period) as a pointer to a zero terminated string
4	MMM_TYPE_MAC	Mac type as a pointer to a 8 byte block

Type representation

The types listed above are represented textually within the MimeMap database file and as parameters to *MimeMap (on page 47)*. The following sections describe each of the types as used in these locations.

Media types

Media types are supplied in the form 'major/minor'. The format and meaning is defined in RFCs 2045-2049 (Request For Comments, the main repository of internet standards) and a database of registered media types is maintained by IANA (Internet Assigned Numbers Authority).

RISC OS type number and name

Whilst both the type name and number are provided in the MimeMap database and as a translation with *SWI MimeMap_Translate (on page 45)*, only the number is retained by the module. This allows the type name to be undefined initially and values to be provided when they are available though the standard file type naming variables.

Type numbers may take any value from &000 to &FFF, as defined by the standard file typing system. The special value &1000 is not used by the MimeMap module.

Dot extensions

Dot extensions are preceeded by a period ('.') character when passed in the *MimeMap command and in the MimeMap file. In the MimeMap file, multiple extensions can be supplied indicating that multiple extensions are used for a single type.

Mac type names

Mac types are specified in the form: "XXXXxxxx" where XXXX is the filetype and xxxx is the vendor type. ? can be used as a single character wildcard. \? means a literal ?. \t, \v, \n, \r have their usual meanings (9, 11, 10 and 13 respectively). \DD (where DD describes 2 hexadecimal digits) represents the character with ASCII code &DD.

There is a default Mac mapping defined, where "&DDDAcrn" maps to filetype &DDD (for hexadecimal digits DDD). This vendor name is registered with Apple. Within Mac mappings, the least number of wildcards is matched if multiple matches are possible.

Mac mappings must be enclosed in double quotes (") when passed to the *MimeMap command, and in the MimeMap file.

File format

The MimeMap file is split into 5 fields to provide the type mappings database. Entries within the file are searched in the same order in which they appear, with wildcards matched last. Fields are tab delimited, and consist of :

```
<mimetype> <tab> <ro-text> <tab> <ro-hex> <tab> <extensions> <tab>
<mactype>
```

Value	Meaning
mimetype	Media type in the form 'major/minor'
major	Major component of the media type, or '*' wildcard to match with any major type.
minor	Minor component of the media type, or '*' wildcard to match with any minor type.
ro-text	Textual form of the 'ro-hex' field for readability purposes
ro-hex	Hexadecimal RISC OS type
extensions	File extensions, in the form of a period (.') prefixed, tab separated list of file extensions.
mactype	Mac vendor/type string as defined above.

System variables

Inet$MimeMappings

Defines the filename of the MimeMap database file

Use

This variable names the file which will be used to provide mappings information for the MimeMap module. Assigning a value to this variable will cause the MimeMap module to re-parse its contents.

Usually this variable will take the value `InetDBase:MimeMap` during normal use. When the system boots, its value will point to a file in ResourceFS which contains a basic file with simple mappings.

Related * commands

*ReadMimeMap (on page 48)

SWI calls

MimeMap_Translate (SWI &50B00)

Perform translation between type systems

On entry

R0 = Input type format
R1 = Input type, or pointer to buffer containing type data
R2 = Output type format
R3 = Pointer to output type buffer of up to 128 bytes, if required by output format

On exit

R0 - R2 preserved
R3 = Result from conversion, or preserved if output data in buffer

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to access the type mapping database to convert from one type format to another. The conversion is performed as cleanly as is possible, returning the most relevant match as provided by the mapping database file.

If no mapping can be performed, an error will be returned.

Related * commands

*MimeMap (on page 47)

MimeMap_Configure (SWI &50B01)

Configure the MimeMap module

On entry

R0 = Configure type :

Value	Meaning
0	Re-read MimeMap file
other	Reserved

On exit

None

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to configure aspects of MimeMap type conversion. At present, only a single reason code is provided; that for forcing a re-read of the MimeMap file.

If the reason code is not recognised, an error is returned.

Related * commands

*ReadMimeMap (on page 48)

*Commands

*MimeMap

Perform a translation and display the results

Syntax

```
*MimeMap &<type>
*MimeMap .<extension>
*MimeMap <major>/<minor>
*MimeMap <typename>
*MimeMap "<mactype>"
```

Parameters

<type>	- Hexadecimal value of RISC OS type
<extension>	- File extension type
<major>	- Major category of media type
<minor>	- Minor type of media type
<typename>	- Textual form of RISC OS type
<mactype>	- Mac type mapping string as used in the configuration file

Use

This command is used to display mappings of types in a human readable manner. It is usually used as a aid for debugging the mapping of types. If issued with no parameters, it will list all the known types.

Examples

```
*MimeMap application/octet-stream *MimeMap &1AD
```

Related SWIs

SWI MimeMap_Translate (on page 45)

*ReadMimeMap

This command causes the MimeMap module to re-read the mappings file.

Syntax

```
*ReadMimeMap
```

Parameters

None

Use

This command is used to force the MimeMap module to re-read the database and parse it into internal structures. Although the MimeMap module will automatically read the database if the system variable changes, if the file is changed the module should be informed through this command.

Examples

```
*ReadMimeMap
```

Related system variables

Inet$MimeMappings (on page 44)

Document information

Maintainer(s): RISCOS Ltd <developer@riscos.com>

History:

Revision	Date	Author	Changes
1		ROL	**Extended from ANT release note** • Added documentation of Mac file type conversion.
2	21 Feb 2003	ROL	**Finished file format** • The file format is now described by the 'code' element.
3	21 Mar 2003	AMH	**Fixes** • Previous version didn't compile. This one does.
4	22 Mar 2003	ROL	**Validation** • Last changes didn't validate at all

Index (Commands)

Index (SWIs)

Index (SWIs by number)

Index (UpCalls)

Index (UpCalls by number)

Index (Services)

Index (Services by number)

Index (SysVars)