

Section 15 - Generic options for routers

15. Generic options for routers

This chapter describes the generic options that apply to all routers. Those that are preconditions are marked with `‡` in the `“use”` field.

For a general description of how a router operates, see sections 3.10 and 3.12. The latter specifies the order in which the preconditions are tested. The order of expansion of the options that provide data for a transport is: `errors_to`, `headers_add`, `headers_remove`, `transport`.

`address_data`Use: routersType: string†Default: unset

The string is expanded just before the router is run, that is, after all the precondition tests have succeeded. If the expansion is forced to fail, the router declines, the value of `address_data` remains unchanged, and the more option controls what happens next. Other expansion failures cause delivery of the address to be deferred.

When the expansion succeeds, the value is retained with the address, and can be accessed using the variable `$address_data` in the current router, subsequent routers, and the eventual transport.

Warning: If the current or any subsequent router is a redirect router that runs a user's filter file, the contents of `$address_data` are accessible in the filter. This is not normally a problem, because such data is usually either not confidential or it `“belongs”` to the current user, but if you do put confidential data into `$address_data` you need to remember this point.

Even if the router declines or passes, the value of `$address_data` remains with the address, though it can be changed by another `address_data` setting on a subsequent router. If a router generates child addresses, the value of `$address_data` propagates to them. This also applies to the special kind of `“child”` that is generated by a router with the `unseen` option.

The idea of `address_data` is that you can use it to look up a lot of data for the address once, and then pick out parts of the data later. For example, you could use a single LDAP lookup to return a string of the form `uid=1234 gid=5678 mailbox=/mail/xyz forward=/home/xyz/.forward`

In the transport you could pick out the mailbox by a setting such as `file = ${extract{mailbox}{$address_data}}`

This makes the configuration file less messy, and also reduces the number of lookups (though Exim does cache lookups).

The `address_data` facility is also useful as a means of passing information from one router to another, and from a router to a transport. In addition, if

When `$address_data` is set by a router when verifying a recipient address from an ACL, it remains available for use in the rest of the ACL statement. After verifying a sender, the value is transferred to `$sender_address_data`.

`address_test`Use: routers‡Type: booleanDefault: true

If this option is set false, the router is skipped when routing is being tested by means of the `-bt` command line option. This can be a convenience when your first router sends messages to an external scanner, because it saves you having to set the `“already scanned”` indicator when testing real address routing.

`cannot_route_message`Use: routersType: string†Default: unset

This option specifies a text message that is used when an address cannot be routed because Exim has run out of routers. The default message is `“Unrouteable address”`. This option is useful only on routers that have more set false, or on the very last router in a configuration, because the value that is used is taken from the last router that is considered. This includes a router that is skipped because its preconditions are not met, as well as a router that declines. For example, using the default configuration, you could put: `cannot_route_message = Remote domain not found in DNS`

on the first router, which is a dnslookup router with more set false, and `cannot_route_message = Unknown local user`

on the final router that checks for local users. If string expansion fails for this option, the default message is used. Unless the expansion failure was explicitly forced, a message about the failure is written to the main and panic logs, in addition to the normal message about the routing failure.

`caseful_local_part`Use: routersType: booleanDefault: false

By default, routers handle the local parts of addresses in a case-insensitive manner, though the actual case is preserved for transmission with the message. If you want the case of letters to be significant in a router, you must set this option true. For individual router options that contain address or local part lists (for example, `local_parts`), case-sensitive matching can be turned on by `“+caseful”`; as a list item. See section 10.19 for more details.

The value of the `$local_part` variable is forced to lower case while a router is running unless `caseful_local_part` is set. When a router assigns an address to a transport, the value of `$local_part` when the transport runs is the same as it was in the router. Similarly, when a router generates child addresses by aliasing or forwarding, the values of `$original_local_part` and `$parent_local_part` are those that were used by the redirecting router.

This option applies to the processing of an address by a router. When a recipient address is being processed in an ACL, there is a separate control modifier that can be used to specify case-sensitive processing within the ACL (see section 39.18).

`check_local_user`Use: routers‡Type: booleanDefault: false

When this option is true, Exim checks that the local part of the recipient address (with affixes removed if relevant) is the name of an account on the local system. The check is done by calling the `getpwnam()` function rather than trying to read `/etc/passwd` directly. This means that other methods of holding password data (such as NIS) are supported. If the local part is a local user, `$home` is set from the password data, and can be tested in other preconditions that are evaluated after this one (the order of evaluation is given in section 3.12). However, the value of `$home` can be overridden by `router_home_directory`. If the local part is not a local user, the router is skipped.

If you want to check that the local part is either the name of a local user or matches something else, you cannot combine `check_local_user` with a setting of `local_parts`, because that specifies the logical and of the two conditions. However, you can use a `passwd` lookup in a `local_parts` setting to achieve this. For example: `local_parts = passwd;$local_part : lsearch;/etc/other/users`

Note, however, that the side effects of `check_local_user` (such as setting up a home directory) do not occur when a `passwd` lookup is used in a `local_parts` (or any other) precondition.

`condition`Use: routers‡Type: string†Default: unset

This option specifies a general precondition test that has to succeed for the router to be called. The condition option is the last precondition to be evaluated (see section 3.12). The string is expanded, and if the result is a forced failure, or an empty string, or one of the strings `“0”`; or `“no”`; or `“false”`; (checked without regard to the case of the letters), the router is skipped, and the address is offered to the next one.

If the result is any other value, the router is run (as this is the last precondition to be evaluated, all the other preconditions must be true).

The condition option provides a means of applying custom conditions to the running of routers. Note that in the case of a simple conditional expansion, the default expansion values are exactly what is wanted. For example: `condition = ${if >{$message_age}{600}}`

Because of the default behaviour of the string expansion, this is equivalent to `condition = ${if >{$message_age}{600}{true}{}}`

If the expansion fails (other than forced failure) delivery is deferred. Some of the other precondition options are common special cases that could in fact be specified using `condition`.

`debug_print`Use: routersType: string†Default: unset

If this option is set and debugging is enabled (see the `-d` command line option), the string is expanded and included in

the debugging output. If expansion of the string fails, the error message is written to the debugging output, and Exim carries on processing. This option is provided to help with checking out the values of variables and so on when debugging router configurations. For example, if a condition option appears not to be working, `debug_print` can be used to output the variables it references. The output happens after checks for domains, `local_parts`, and `check_local_user` but before any other preconditions are tested. A newline is added to the text if it does not end with one.

`disable_logging`Use: routersType: booleanDefault: false

If this option is set true, nothing is logged for any routing errors or for any deliveries caused by this router. You should not set this option unless you really, really know what you are doing. See also the generic transport option of the same name.

`domains`Use: routers‡Type: domain list†Default: unset

If this option is set, the router is skipped unless the current domain matches the list. If the match is achieved by means of a file lookup, the data that the lookup returned for the domain is placed in `$domain_data` for use in string expansions of the driver's private options. See section 3.12 for a list of the order in which preconditions are evaluated.

`driver`Use: routersType: stringDefault: unset

This option must always be set. It specifies which of the available routers is to be used.

`errors_to`Use: routersType: string†Default: unset

If a router successfully handles an address, it may assign the address to a transport for delivery or it may generate child addresses. In both cases, if there is a delivery problem during later processing, the resulting bounce message is sent to the address that results from expanding this string, provided that the address verifies successfully. The `errors_to` option is expanded before `headers_add`, `headers_remove`, and `transport`.

The `errors_to` setting associated with an address can be overridden if it subsequently passes through other routers that have their own `errors_to` settings, or if the message is delivered by a transport with a `return_path` setting.

If `errors_to` is unset, or the expansion is forced to fail, or the result of the expansion fails to verify, the errors address associated with the incoming address is used. At top level, this is the envelope sender. A non-forced expansion failure causes delivery to be deferred.

If an address for which `errors_to` has been set ends up being delivered over SMTP, the envelope sender for that delivery is the `errors_to` value, so that any bounces that are generated by other MTAs on the delivery route are also sent there. You can set `errors_to` to the empty string by either of these settings: `errors_to =`
`errors_to = ""`

An expansion item that yields an empty string has the same effect. If you do this, a locally detected delivery error for addresses processed by this router no longer gives rise to a bounce message; the error is discarded. If the address is delivered to a remote host, the return path is set to `<>`, unless overridden by the `return_path` option on the transport.

If for some reason you want to discard local errors, but use a non-empty MAIL command for remote delivery, you can preserve the original return path in `$address_data` in the router, and reinstate it in the transport by setting `return_path`.

The most common use of `errors_to` is to direct mailing list bounces to the manager of the list, as described in section 46.2, or to implement VERP (Variable Envelope Return Paths) (see section 46.6).

`expn`Use: routers‡Type: booleanDefault: true

If this option is turned off, the router is skipped when testing an address as a result of processing an SMTP EXPN command. You might, for example, want to turn it off on a router for users' `.forward` files, while leaving it on for the system alias file. See section 3.12 for a list of the order in which preconditions are evaluated.

The use of the SMTP EXPN command is controlled by an ACL (see chapter 39). When Exim is running an EXPN command, it is similar to testing an address with `-bt`. Compare VRFY, whose counterpart is `-bv`.

`fail_verify`Use: routersType: booleanDefault: false

Setting this option has the effect of setting both `fail_verify_sender` and `fail_verify_recipient` to the same value.

`fail_verify_recipientUse: routersType: booleanDefault: false`

If this option is true and an address is accepted by this router when verifying a recipient, verification fails.

`fail_verify_senderUse: routersType: booleanDefault: false`

If this option is true and an address is accepted by this router when verifying a sender, verification fails.

`fallback_hostsUse: routersType: string listDefault: unset`

String expansion is not applied to this option. The argument must be a colon-separated list of host names or IP addresses. The list separator can be changed (see section 6.19), and a port can be specified with each name or address. In fact, the format of each item is exactly the same as defined for the list of hosts in a manualroute router (see section 20.5).

If a router queues an address for a remote transport, this host list is associated with the address, and used instead of the transport's fallback host list. If `hosts_randomize` is set on the transport, the order of the list is randomized for each use. See the `fallback_hosts` option of the `smtp` transport for further details.

`groupUse: routersType: string†Default: see below`

When a router queues an address for a transport, and the transport does not specify a group, the group given here is used when running the delivery process. The group may be specified numerically or by name. If expansion fails, the error is logged and delivery is deferred. The default is unset, unless `check_local_user` is set, when the default is taken from the password information. See also `initgroups` and `user` and the discussion in chapter 23.

`headers_addUse: routersType: string†Default: unset`

This option specifies a string of text that is expanded at routing time, and associated with any addresses that are accepted by the router. However, this option has no effect when an address is just being verified. The way in which the text is used to add header lines at transport time is described in section 43.17. New header lines are not actually added until the message is in the process of being transported. This means that references to header lines in string expansions in the transport's configuration do not "see" the added header lines.

The `headers_add` option is expanded after `errors_to`, but before `headers_remove` and transport. If the expanded string is empty, or if the expansion is forced to fail, the option has no effect. Other expansion failures are treated as configuration errors.

Warning 1: The `headers_add` option cannot be used for a redirect router that has the `one_time` option set.

Warning 2: If the `unseen` option is set on the router, all header additions are deleted when the address is passed on to subsequent routers.

`headers_removeUse: routersType: string†Default: unset`

This option specifies a string of text that is expanded at routing time, and associated with any addresses that are accepted by the router. However, this option has no effect when an address is just being verified. The way in which the text is used to remove header lines at transport time is described in section 43.17. Header lines are not actually removed until the message is in the process of being transported. This means that references to header lines in string expansions in the transport's configuration still "see" the original header lines.

The `headers_remove` option is expanded after `errors_to` and `headers_add`, but before transport. If the expansion is forced to fail, the option has no effect. Other expansion failures are treated as configuration errors.

Warning 1: The `headers_remove` option cannot be used for a redirect router that has the `one_time` option set.

Warning 2: If the `unseen` option is set on the router, all header removal requests are deleted when the address is passed on to subsequent routers.

`ignore_target_hostsUse: routersType: host list†Default: unset`

Although this option is a host list, it should normally contain IP address entries rather than names. If any host that is looked up by the router has an IP address that matches an item in this list, Exim behaves as if that IP address did not exist. This option allows you to cope with rogue DNS entries like `remote.domain.example. A 127.0.0.1`

by setting `ignore_target_hosts = 127.0.0.1`

on the relevant router. If all the hosts found by a `dnslookup` router are discarded in this way, the router declines. In a conventional configuration, an attempt to mail to such a domain would normally provoke the `“unrouteable domain”` error, and an attempt to verify an address in the domain would fail. Similarly, if `ignore_target_hosts` is set on an `ipliteral` router, the router declines if presented with one of the listed addresses.

You can use this option to disable the use of IPv4 or IPv6 for mail delivery by means of the first or the second of the following settings, respectively: `ignore_target_hosts = 0.0.0.0/0`
`ignore_target_hosts = <; 0::0/0`

The pattern in the first line matches all IPv4 addresses, whereas the pattern in the second line matches all IPv6 addresses.

This option may also be useful for ignoring link-local and site-local IPv6 addresses. Because, like all host lists, the value of `ignore_target_hosts` is expanded before use as a list, it is possible to make it dependent on the domain that is being routed.

During its expansion, `$host_address` is set to the IP address that is being checked.

`initgroupsUse: routersType: booleanDefault: false`

If the router queues an address for a transport, and this option is true, and the uid supplied by the router is not overridden by the transport, the `initgroups()` function is called when running the transport to ensure that any additional groups associated with the uid are set up. See also `group` and `user` and the discussion in chapter 23.

`local_part_prefixUse: routers‡Type: string listDefault: unset`

If this option is set, the router is skipped unless the local part starts with one of the given strings, or `local_part_prefix_optional` is true. See section 3.12 for a list of the order in which preconditions are evaluated.

The list is scanned from left to right, and the first prefix that matches is used. A limited form of wildcard is available; if the prefix begins with an asterisk, it matches the longest possible sequence of arbitrary characters at the start of the local part. An asterisk should therefore always be followed by some character that does not occur in normal local parts. Wildcarding can be used to set up multiple user mailboxes, as described in section 46.8.

During the testing of the `local_parts` option, and while the router is running, the prefix is removed from the local part, and is available in the expansion variable `$local_part_prefix`. When a message is being delivered, if the router accepts the address, this remains true during subsequent delivery by a transport. In particular, the local part that is transmitted in the RCPT command for LMTP, SMTP, and BSMTP deliveries has the prefix removed by default. This behaviour can be overridden by setting `rcpt_include_affixes` true on the relevant transport.

When an address is being verified, `local_part_prefix` affects only the behaviour of the router. If the callout feature of verification is in use, this means that the full address, including the prefix, will be used during the callout.

The prefix facility is commonly used to handle local parts of the form `owner-something`. Another common use is to support local parts of the form `real-username` to bypass a user's `.forward` file `–` helpful when trying to tell a user their forwarding is broken `–` by placing a router like this one immediately before the router that handles `.forward` files:

```
real_localuser:
driver = accept
local_part_prefix = real-
check_local_user
transport = local_delivery
```

If both `local_part_prefix` and `local_part_suffix` are set for a router, both conditions must be met if not optional. Care must be taken if wildcards are used in both a prefix and a suffix on the same router. Different separator characters must be used to avoid ambiguity.

`local_part_prefix_optionalUse: routersType: booleanDefault: false`

See `local_part_prefix` above.

`local_part_suffixUse: routers‡Type: string listDefault: unset`

This option operates in the same way as `local_part_prefix`, except that the local part must end (rather than start) with the given string, the `local_part_suffix_optional` option determines whether the suffix is mandatory, and the wildcard `*` character, if present, must be the last character of the suffix. This option facility is commonly used to handle local parts of the form `something-request` and multiple user mailboxes of the form `username-foo`.

`local_part_suffix_optionalUse: routersType: booleanDefault: false`

See `local_part_suffix` above.

`local_partsUse: routers‡Type: local part list†Default: unset`

The router is run only if the local part of the address matches the list. See section 3.12 for a list of the order in which preconditions are evaluated, and section 10.20 for a discussion of local part lists. Because the string is expanded, it is possible to make it depend on the domain, for example: `local_parts = dbm:/usr/local/specials/$domain`

If the match is achieved by a lookup, the data that the lookup returned for the local part is placed in the variable `$local_part_data` for use in expansions of the router's private options. You might use this option, for example, if you have a large number of local virtual domains, and you want to send all postmaster mail to the same place without having to set up an alias in each virtual domain: `postmaster:`

```
driver = redirect
local_parts = postmaster
data = postmaster@real.domain.example
```

`log_as_localUse: routersType: booleanDefault: see below`

Exim has two logging styles for delivery, the idea being to make local deliveries stand out more visibly from remote ones. In the `“local”` style, the recipient address is given just as the local part, without a domain. The use of this style is controlled by this option. It defaults to `true` for the accept router, and `false` for all the others. This option applies only when a router assigns an address to a transport. It has no effect on routers that redirect addresses.

`moreUse: routersType: boolean†Default: true`

The result of string expansion for this option must be a valid boolean value, that is, one of the strings `“yes”`, `“no”`, `“true”`, or `“false”`. Any other result causes an error, and delivery is deferred. If the expansion is forced to fail, the default value for the option (`true`) is used. Other failures cause delivery to be deferred.

If this option is set `false`, and the router declines to handle the address, no further routers are tried, routing fails, and the address is bounced. However, if the router explicitly passes an address to the following router by means of the setting `self = pass`

or otherwise, the setting of `more` is ignored. Also, the setting of `more` does not affect the behaviour if one of the precondition tests fails. In that case, the address is always passed to the next router.

Note that `address_data` is not considered to be a precondition. If its expansion is forced to fail, the router declines, and the value of `more` controls what happens next.

`pass_on_timeoutUse: routersType: booleanDefault: false`

If a router times out during a host lookup, it normally causes deferral of the address. If `pass_on_timeout` is set, the address is passed on to the next router, overriding `no_more`. This may be helpful for systems that are intermittently connected to the Internet, or those that want to pass to a smart host any messages that cannot immediately be delivered.

There are occasional other temporary errors that can occur while doing DNS lookups. They are treated in the same way as a timeout, and this option applies to all of them.

`pass_routerUse: routersType: stringDefault: unset`

When a router returns `“pass”`, the address is normally handed on to the next router in sequence. This can be changed by setting `pass_router` to the name of another router. However (unlike `redirect_router`) the named router must be below the current router, to avoid loops. Note that this option applies only to the special case of `“pass”`. It does not apply when a router returns `“decline”`.

`redirect_router`Use: routersType: stringDefault: unset

Sometimes an administrator knows that it is pointless to reprocess addresses generated from alias or forward files with the same router again. For example, if an alias file translates real names into login ids there is no point searching the alias file a second time, especially if it is a large file.

The `redirect_router` option can be set to the name of any router instance. It causes the routing of any generated addresses to start at the named router instead of at the first router. This option has no effect if the router in which it is set does not generate new addresses.

`require_files`Use: routers‡Type: string list†Default: unset

This option provides a general mechanism for predicating the running of a router on the existence or non-existence of certain files or directories. Before running a router, as one of its precondition tests, Exim works its way through the `require_files` list, expanding each item separately.

Because the list is split before expansion, any colons in expansion items must be doubled, or the facility for using a different list separator must be used. If any expansion is forced to fail, the item is ignored. Other expansion failures cause routing of the address to be deferred.

If any expanded string is empty, it is ignored. Otherwise, except as described below, each string must be a fully qualified file path, optionally preceded by `“!”`. The paths are passed to the `stat()` function to test for the existence of the files or directories. The router is skipped if any paths not preceded by `“!”` do not exist, or if any paths preceded by `“!”` do exist.

If `stat()` cannot determine whether a file exists or not, delivery of the message is deferred. This can happen when NFS-mounted filesystems are unavailable.

This option is checked after the `domains`, `local_parts`, and `senders` options, so you cannot use it to check for the existence of a file in which to look up a domain, local part, or sender. (See section 3.12 for a full list of the order in which preconditions are evaluated.) However, as these options are all expanded, you can use the `exists` expansion condition to make such tests. The `require_files` option is intended for checking files that the router may be going to use internally, or which are needed by a transport (for example `.procmailrc`).

During delivery, the `stat()` function is run as root, but there is a facility for some checking of the accessibility of a file by another user. This is not a proper permissions check, but just a `“rough”` check that operates as follows:

If an item in a `require_files` list does not contain any forward slash characters, it is taken to be the user (and optional group, separated by a comma) to be checked for subsequent files in the list. If no group is specified but the user is specified symbolically, the gid associated with the uid is used. For example: `require_files = mail:/some/file`
`require_files = $local_part:$home/.procmailrc`

If a user or group name in a `require_files` list does not exist, the `require_files` condition fails.

Exim performs the check by scanning along the components of the file path, and checking the access for the given uid and gid. It checks for `“x”` access on directories, and `“r”` access on the final file. Note that this means that file access control lists, if the operating system has them, are ignored.

Warning 1: When the router is being run to verify addresses for an incoming SMTP message, Exim is not running as root, but under its own uid. This may affect the result of a `require_files` check. In particular, `stat()` may yield the error `EACCES (“Permission denied”)`. This means that the Exim user is not permitted to read one of the directories on the file’s path.

Warning 2: Even when Exim is running as root while delivering a message, `stat()` can yield `EACCES` for a file in an NFS directory that is mounted without root access. In this case, if a check for access by a particular user is requested, Exim creates a subprocess that runs as that user, and tries the check again in that process.

The default action for handling an unresolved `EACCES` is to consider it to be caused by a configuration error, and routing is deferred because the existence or non-existence of the file cannot be determined. However, in some

circumstances it may be desirable to treat this condition as if the file did not exist. If the file name (or the exclamation mark that precedes the file name for non-existence) is preceded by a plus sign, the EACCES error is treated as if the file did not exist. For example: `require_files = +/some/file`

If the router is not an essential part of verification (for example, it handles users' .forward files), another solution is to set the `verify` option false so that the router is skipped when verifying.

`retry_use_local_part`Use: routersType: booleanDefault: see below

When a delivery suffers a temporary routing failure, a retry record is created in Exim's hints database. For addresses whose routing depends only on the domain, the key for the retry record should not involve the local part, but for other addresses, both the domain and the local part should be included. Usually, remote routing is of the former kind, and local routing is of the latter kind.

This option controls whether the local part is used to form the key for retry hints for addresses that suffer temporary errors while being handled by this router. The default value is true for any router that has `check_local_user` set, and false otherwise. Note that this option does not apply to hints keys for transport delays; they are controlled by a generic transport option of the same name.

The setting of `retry_use_local_part` applies only to the router on which it appears. If the router generates child addresses, they are routed independently; this setting does not become attached to them.

`router_home_directory`Use: routersType: stringDefault: unset

This option sets a home directory for use while the router is running. (Compare `transport_home_directory`, which sets a home directory for later transporting.) In particular, if used on a redirect router, this option sets a value for `$home` while a filter is running. The value is expanded; forced expansion failure causes the option to be ignored – other failures cause the router to defer.

Expansion of `router_home_directory` happens immediately after the `check_local_user` test (if configured), before any further expansions take place. (See section 3.12 for a list of the order in which preconditions are evaluated.) While the router is running, `router_home_directory` overrides the value of `$home` that came from `check_local_user`.

When a router accepts an address and assigns it to a local transport (including the cases when a redirect router generates a pipe, file, or autoreply delivery), the home directory setting for the transport is taken from the first of these values that is set:

-

The `home_directory` option on the transport;

-

The `transport_home_directory` option on the router;

-

The password data if `check_local_user` is set on the router;

-

The `router_home_directory` option on the router.

In other words, `router_home_directory` overrides the password data for the router, but not for the transport.

`self`Use: routersType: stringDefault: freeze

This option applies to those routers that use a recipient address to find a list of remote hosts. Currently, these are the `dnslookup`, `ipliteral`, and `manualroute` routers. Certain configurations of the `queryprogram` router can also specify a list of remote hosts. Usually such routers are configured to send the message to a remote host via an `smtp` transport. The `self` option specifies what happens when the first host on the list turns out to be the local host. The way in which Exim checks for the local host is described in section 13.8.

Normally this situation indicates either an error in Exim's configuration (for example, the router should be configured not to process this domain), or an error in the DNS (for example, the MX should not point to this host). For this reason, the default action is to log the incident, defer the address, and freeze the message. The following alternatives are provided for use in special cases: `defer`

Delivery of the message is tried again later, but the message is not frozen. `reroute: <domain>`

The domain is changed to the given domain, and the address is passed back to be reprocessed by the routers. No rewriting of headers takes place. This behaviour is essentially a redirection. `reroute: rewrite: <domain>`

The domain is changed to the given domain, and the address is passed back to be reprocessed by the routers. Any headers that contain the original domain are rewritten. `pass`

The router passes the address to the next router, or to the router named in the `pass_router` option if it is set. This overrides `no_more`. During subsequent routing and delivery, the variable `$self_hostname` contains the name of the local host that the router encountered. This can be used to distinguish between different cases for hosts with multiple names. The combination `self = pass`
`no_more`

ensures that only those addresses that routed to the local host are passed on. Without `no_more`, addresses that were declined for other reasons would also be passed to the next router. `fail`

Delivery fails and an error report is generated. `send`

The anomaly is ignored and the address is queued for the transport. This setting should be used with extreme caution. For an smtp transport, it makes sense only in cases where the program that is listening on the SMTP port is not this version of Exim. That is, it must be some other MTA, or Exim with a different configuration file that handles the domain in another way.

`sendersUse: routers‡Type: address list†Default: unset`

If this option is set, the router is skipped unless the message's sender address matches something on the list. See section 3.12 for a list of the order in which preconditions are evaluated.

There are issues concerning verification when the running of routers is dependent on the sender. When Exim is verifying the address in an `errors_to` setting, it sets the sender to the null string. When using the `-bt` option to check a configuration file, it is necessary also to use the `-f` option to set an appropriate sender. For incoming mail, the sender is unset when verifying the sender, but is available when verifying any recipients. If the SMTP VRFY command is enabled, it must be used after MAIL if the sender address matters.

`translate_ip_addressUse: routersType: string†Default: unset`

There exist some rare networking situations (for example, packet radio) where it is helpful to be able to translate IP addresses generated by normal routing mechanisms into other IP addresses, thus performing a kind of manual IP routing. This should be done only if the normal IP routing of the TCP/IP stack is inadequate or broken. Because this is an extremely uncommon requirement, the code to support this option is not included in the Exim binary unless `SUPPORT_TRANSLATE_IP_ADDRESS=yes` is set in `Local/Makefile`.

The `translate_ip_address` string is expanded for every IP address generated by the router, with the generated address set in `$host_address`. If the expansion is forced to fail, no action is taken. For any other expansion error, delivery of the message is deferred. If the result of the expansion is an IP address, that replaces the original address; otherwise the result is assumed to be a host name – this is looked up using `gethostbyname()` (or `getipnodebyname()` when available) to produce one or more replacement IP addresses. For example, to subvert all IP addresses in some specific networks, this could be added to a router: `translate_ip_address = \`
 `${lookup{${mask:$host_address/26}}|search{/some/file}\`
 `{$value}fail}}`

The file would contain lines like `10.2.3.128/26 some.host`
`10.8.4.34/26 10.44.8.15`

You should not make use of this facility unless you really understand what you are doing.

`transportUse: routersType: string†Default: unset`

This option specifies the transport to be used when a router accepts an address and sets it up for delivery. A transport is never needed if a router is used only for verification. The value of the option is expanded at routing time, after the expansion of `errors_to`, `headers_add`, and `headers_remove`, and result must be the name of one of the configured

transports. If it is not, delivery is deferred.

The transport option is not used by the redirect router, but it does have some private options that set up transports for pipe and file deliveries (see chapter 22).

`transport_current_directoryUse: routersType: string†Default: unset`

This option associates a current directory with any address that is routed to a local transport. This can happen either because a transport is explicitly configured for the router, or because it generates a delivery to a file or a pipe. During the delivery process (that is, at transport time), this option string is expanded and is set as the current directory, unless overridden by a setting on the transport. If the expansion fails for any reason, including forced failure, an error is logged, and delivery is deferred. See chapter 23 for details of the local delivery environment.

`transport_home_directoryUse: routersType: string†Default: see below`

This option associates a home directory with any address that is routed to a local transport. This can happen either because a transport is explicitly configured for the router, or because it generates a delivery to a file or a pipe. During the delivery process (that is, at transport time), the option string is expanded and is set as the home directory, unless overridden by a setting of `home_directory` on the transport. If the expansion fails for any reason, including forced failure, an error is logged, and delivery is deferred.

If the transport does not specify a home directory, and `transport_home_directory` is not set for the router, the home directory for the transport is taken from the password data if `check_local_user` is set for the router. Otherwise it is taken from `router_home_directory` if that option is set; if not, no home directory is set for the transport.

See chapter 23 for further details of the local delivery environment.

`unseenUse: routersType: boolean†Default: false`

The result of string expansion for this option must be a valid boolean value, that is, one of the strings `“yes”`, `“no”`, `&ldquo>true”`, or `&ldquo>false”`. Any other result causes an error, and delivery is deferred. If the expansion is forced to fail, the default value for the option (`false`) is used. Other failures cause delivery to be deferred.

When this option is set true, routing does not cease if the router accepts the address. Instead, a copy of the incoming address is passed to the next router, overriding a false setting of `more`. There is little point in setting `more` false if `unseen` is always true, but it may be useful in cases when the value of `unseen` contains expansion items (and therefore, presumably, is sometimes true and sometimes false).

The `unseen` option can be used to cause copies of messages to be delivered to some other destination, while also carrying out a normal delivery. In effect, the current address is made into a `“parent”` that has two children `–` one that is delivered as specified by this router, and a clone that goes on to be routed further. For this reason, `unseen` may not be combined with the `one_time` option in a redirect router.

Warning: Header lines added to the address (or specified for removal) by this router or by previous routers affect the `“unseen”` copy of the message only. The clone that continues to be processed by further routers starts with no added headers and none specified for removal. However, any data that was set by the `address_data` option in the current or previous routers is passed on. Setting the `unseen` option has a similar effect to the `unseen` command qualifier in filter files.

`userUse: routersType: string†Default: see below`

When a router queues an address for a transport, and the transport does not specify a user, the user given here is used when running the delivery process. The user may be specified numerically or by name. If expansion fails, the error is logged and delivery is deferred. This user is also used by the redirect router when running a filter file. The default is unset, except when `check_local_user` is set. In this case, the default is taken from the password information. If the user is specified as a name, and `group` is not set, the group associated with the user is used. See also `initgroups` and `group` and the discussion in chapter 23.

`verifyUse: routers‡Type: booleanDefault: true`

Setting this option has the effect of setting `verify_sender` and `verify_recipient` to the same value.

`verify_onlyUse: routers‡Type: booleanDefault: false`

If this option is set, the router is used only when verifying an address or testing with the `-bv` option, not when actually doing a delivery, testing with the `-bt` option, or running the SMTP EXPN command. It can be further restricted to verifying only senders or recipients by means of `verify_sender` and `verify_recipient`.

Warning: When the router is being run to verify addresses for an incoming SMTP message, Exim is not running as root, but under its own uid. If the router accesses any files, you need to make sure that they are accessible to the Exim user or group.

`verify_recipient`Use: routers‡Type: booleanDefault: true

If this option is false, the router is skipped when verifying recipient addresses or testing recipient verification using `-bv`. See section 3.12 for a list of the order in which preconditions are evaluated.

`verify_sender`Use: routers‡Type: booleanDefault: true If this option is false, the router is skipped when verifying sender addresses or testing sender verification using `-bvs`. See section 3.12 for a list of the order in which preconditions are evaluated.