# Reltool application

**version 0.2**

Typeset in LaTeX from SGML source using the DocBuilder-0.9.8.5 Document System.

# Contents

# Chapter 1

# Reltool Users Guide

`Reltool` is a release management tool. It analyses a given Erlang/OTP installation and determines various dependencies between applications. The `graphical` frontend depicts the dependencies and enables interactive customization of a target system. The backend provides a `batch` interface for generation of customized target systems.

## 1.1 Introduction

`Reltool` is a release management tool. It analyses a given Erlang/OTP installation and determines various dependencies between applications. The `graphical` frontend depicts the dependencies and enables interactive customization of a target system. The backend provides a `batch` interface for generation of customized target systems.

### 1.1.1 Scope and Purpose

This manual describes the Reltool application, as a component of the Erlang/Open Telecom Platform development environment. It is assumed that the reader is familiar with the Erlang Development Environment, which is described in a separate User's Guide.

### 1.1.2 Prerequisites

The following prerequisites is required for understanding the material in the Reltool User's Guide:

- familiarity with Erlang/OTP system principles and Erlang/OTP design principles

The application requires Erlang/OTP release R13A or later.

### 1.1.3 About This Manual

In addition to this introductory chapter, the Reltool User's Guide contains the following chapters:

- Chapter 2: "Usage" describes the architecture and typical usage of the application.
- Chapter 3: "Examples" gives some usage examples

### 1.1.4 Where to Find More Information

Refer to the following documentation for more information about Reltool and about the Erlang/OTP development system:

- the Reference Manual of `Reltool`
- the Erlang/OTP `System Principles`
- the Erlang/OTP `Design Principles`
- Programming Erlang: Software for a Concurrent World (2007), Pragmatic Bookshelf, ISBN13: 9781934356005.

## 1.2 Usage

### 1.2.1 Overview

This document focuses on the graphical parts of the tool. The concepts are explained in the reference manual for the module `reltool`.

### 1.2.2 System window

The system window is started with the function `reltool:start/1`. At startup the tool will process the all `beam` files and `app` files in order to find out dependencies between applications and their modules. Once all this information has been derived, it will be possible to explore the tool.

The system window consists of four main pages (tabs):

- Libraries
- System settings
- Applications
- Releases

Click on a name tag to display its page.

#### Libraries

On the library page it is possible to control which sources that the tool will use. The page is organized as a tree which can be expanded and collapsed by clicking on the little symbol in the beginning of the expandable/collapsable lines.

The `Root directory` can be edited by selecting the line where the path of the root directory is displayed and clicking with the right mouse button. Choose edit in the menu that pops up.

Library directories can be added, edited or deleted. This is done by selecting the line where the path to a library directory is displayed and clicking with the right mouse button. Choose add, edit or delete in the menu that pops up. New library directories can also be added by selecting the line `Library directories` and clicking with the right mouse button. Choose add in the menu that pops up.

Escript files can be added, edited or deleted. This is done by selecting the line where the path to an escript file is displayed and clicking with the right mouse button. Choose add, edit or delete in the menu that pops up. New escripts can also be added by selecting the line `Escript files` and clicking with the right mouse button. Choose add in the menu that pops up.

When libraries and escripts are expanded, the names of their contained applications will be displayed. Double click on an application name to launch an application window.

### System settings

On the system settings page it is possible to control some global settings that are used as defaults for all applications. Set the `Application inclusion policy` to `include` to include all applications that not are explicitly excluded. See `incl_cond` (application inclusion) and `mod_cond` (module inclusion) in the reference manual for the module `reltool` for more info.

The system settings page is rather incomplete.

### Applications

There are four categories of applications on the applications page. `Included` contains applications that are explicitly included. `Excluded` contains applications that are explicitly excluded. `Derived` contains applications that either are used directly by excplicitly included applications or by other derived applications. `Available` contains the remaining applications.

Select one or more applications and click on a button directly below the application column to change application category. For example, select an available application and click on its tick button to move the application to the included category. Clicking on the tick symbol for included applications will move the application back to the available category. The tick is undone.

The symbols in front of the application names are intended to describe the status of the application. There are error symbols and warning symbols that means that there are something that needs attention. The tick symbol means that the application is included or derived and no problem has been detected. The cross symbol means that the application is excluded or available and no problem has been detected. Applications with error symbols are listed first in each category, then comes the warnings and the normal ones (ticks and crosses) are found at the end.

Double click on an application to launch its application window.

### Releases

The releases page is incomplete and very experimental.

### File menu

- `Display application dependency graph` - Launches an application force graph window. All included and derived applications and their dependencies will be shown in a graph.
- `Display module dependency graph` - Launch a module force graph window. All included and derived modules and their dependencies will be shown in a graph.
- `Reset configuration to default`
- `Undo configuration (toggle)`
- `Load configuration` - Loads a new configuration from file.
- `Save configuration` - Saves the current configuration to file. Only configuration parameters with values that differs from their defaults are saved.
- `Generate rel, script and boot files`
- `Generate target system`
- `Close` - Close the system window and all its subwindows.

#### Dependencies between applications or modules displayed as a graph

The dependency graph windows are launched from the file menu in the system window. The graph depicts all included and derived applications/modules and their dependencies.

It is possible to perform some limited manipulations of the graph. Nodes can be moved, selected, locked or deleted. Move a single node or the entire graph by moving the mouse while the left mouse button is pressed. A node is can be locked into a fix position by holding down the shift button when the left mouse button is released. Select several nodes by moving the mouse while the control key and the left mouse button i pressed. Selected nodes can be locked, unlocked or deleted by klicking on a suitable button.

The algorithm that is used to draw a graph with as few crossed links as possible is called force graph. A force graph consists of nodes and directed link between nodes. Each node is associated with a repulsive force that pushes nodes away from each other. This force can be adjusted with the left slider or with the mouse wheel. Each link is associated with an attractive force that pulls the nodes nearer each other. This force can be adjusted with the right slider. If this force becomes to strong, the graph will be unstable. The third parameter that can be adjusted is the length of the links. It is adjusted with the middle slider.

The `Freeze` button starts/stops the redrawing of the graph. `Reset` moves the graph to the middle of the window and resets all graph settings to default, with the exception of deleted nodes.

### 1.2.3 Application window

The application window is started by double clicking on an application name. The application window consists of four pages (tabs):

- Application settings
- Modules
- Application dependencies
- Module dependencies

Click on a name tag to display its page.

#### Application settings

Select version of the application in the `Source selection policy` part of the page. By default the latest version of the application is selected, but it is possible to override this by explicitly select another version.

By default the `Application inclusion policy` on system level is used for all applications. Set the value to `include` if you want to excplicitly include one particular application. Set it to `exclude` if you want to exclude the application despite that it is used by another (explicitly or implicitly) included application. `derived` means that the application automatically will be included if some other (explicitly or implicitly) included application uses it.

By default the `Module inclusion policy` on system level is used for all applications. Set it to `derived` if you only want actually used modules to be included. Set it to `app` if you, besides derived modules, also want the modules listed in the app file to be included. Set it to `ebin` if you, besides derived modules, also want the modules that exists as beam files on the ebin directory to be included. Set it to `all` if you want all modules to be included, that is the union of modules found on the ebin directory and listed in the app file.

The application settings page is rather incomplete.

### Modules

There are four categories of modules on the modules page. `Included` contains modules that are explicitly included. `Excluded` contains modules that are explicitly excluded. `Derived` contains modules that either are used directly by excplicitly included modules or by other derived modules. `Available` contains the remaining modules.

Select one or more modules and click on a button directly below the module column to change module category. For example, select an available module and click on its tick button to move the module to the included category. Clicking on the tick symbol for included modules will move the module back to the available category. The tick is undone.

The symbola in front of the module names are intended to describe the status of the module. There are error symbols and warning symbols that means that there are something that needs attention. The tick symbol means that the module is included or derived and no problem has been detected. The cross symbol means that the module is excluded or available and no problem has been detected. Modules with error symbols are listed first in each category, then comes the warnings and the normal ones (ticks and crosses) are found at the end.

Double click on an module to launch its module window.

### Application dependencies

There are four categories of applications on the `Application dependencies` page. If the application is used by other applications, these are listed under `Used by`. If the application requires other applications be started before it can be started, these are listed under `Required`. These applications are listed in the `applications` part of the `app` file. If the application includes other applications, these are listed under `Included`. These applications are listed in the `included_applications` part of the `app` file. If the application uses modules other applications, these are listed under `Uses`.

Double click on an application name to launch an application window.

### Module dependencies

There are two categories of modules on the `Module dependencies` page. If the module is used by other modules, these are listed under `Modules used by others`. If the module uses modules other modules, these are listed under `Used modules`.

Double click on an module name to launch a module window.

## 1.2.4 Module window

The module window is started by double clicking on an module name. The module window consists initially of two pages (tabs):

- Dependencies
- Code

Click on a name tag to display its page.

#### Dependencies

There are two categories of modules on the Dependencies page. If the module is used by other modules, these are listed under Modules used by others. If the module uses modules other modules, these are listed under Used modules.

Double click on an module name to launch a module window.

#### Code

On the Code page the Erlang source code is displayed. It is possible to search forwards and backwards for text in the module. Enter a regular expression in the Find field and press enter. It is also possible to goto a certain line on the module. The Back button can be used to go back to the previous position.

Put the marker on a function name and double click to go to the definition of the function. If the function is defined in another module, that module will be loaded and added to the page list.

## 1.3 Examples

### 1.3.1 Start and stop windows and servers

```
Erlang R13A (erts-5.7) [source] [64-bit] [smp:4:4] [rq:4] [async-threads:0] [kernel-poll:false]

Eshell V5.7  (abort with ^G)
1> reltool:start_server([]).
{ok,<0.35.0>}
2> reltool:get_config_server(Server).
{ok,{sys,[]}}
3>  reltool:stop(Server).
ok
```

### 1.3.2 Generate release and script files

```
5> {ok, Server} = reltool:start_server([{config, {sys, [{boot_rel, "NAME"}, {rel, "NAME", "VSN", [kernel
{ok,<0.1288.0>}
6>  reltool:get_config(Server).
{ok,{sys,[{boot_rel,"NAME"},
          {rel,"NAME","VSN",[kernel,stdlib,sasl]}]}}
7>  reltool:get_rel(Server, "NAME").
{ok,{release,{"NAME","VSN"},
             {erts,"5.7"},
             [{kernel,"2.13"},{stdlib,"1.16"},{sasl,"2.1.6"}]}}
8>  reltool:get_script(Server, "NAME").
{ok,{script,{"NAME","VSN"},
            [{preLoaded,[erl_prim_loader,erlang,init,otp_ring0,
                         prim_file,prim_inet,prim_zip,zlib]},
             {progress,preloaded},
             {path,["$ROOT/lib/kernel-2.13/ebin",
                    "$ROOT/lib/stdlib-1.16/ebin"]},
             {primLoad,[error_handler]},
             {kernel_load_completed},
```

```
             {progress,kernel_load_completed},
             {path,["$ROOT/lib/kernel-2.13/ebin"]},
             {primLoad,[application,application_controller,
                        application_master,application_starter,auth,code,
                        code_server,disk_log,disk_log_1,disk_log_server,
                        disk_log_sup,dist_ac,dist_util,erl_boot_server|...]},
             {path,["$ROOT/lib/stdlib-1.16/ebin"]},
             {primLoad,[array,base64,beam_lib,c,calendar,dets,
                        dets_server,dets_sup,dets_utils,dets_v8,dets_v9,dict|...]},
             {path,["$ROOT/lib/sasl-2.1.6/ebin"]},
             {primLoad,[alarm_handler,erlsrv,format_lib_supp,misc_supp,
                        overload,rb,rb_format_supp,release_handler,
                        release_handler_1,sasl|...]},
             {progress,modules_loaded},
             {path,["$ROOT/lib/kernel-2.13/ebin",
                    "$ROOT/lib/stdlib-1.16/ebin","$ROOT/lib/sasl-2.1.6/ebin"]},
             {kernelProcess,heart,{heart,start,[]}},
             {kernelProcess,error_logger,{error_logger,start_link,[]}},
             {kernelProcess,application_controller,
                          {application_controller,start,[{...}]}},
             {progress,init_kernel_started},
             {apply,{application,load,[...]}},
             {apply,{application,load,...}},
             {progress,applications_loaded},
             {apply,{...}},
             {apply,...},
             {...}|...]}}
9>  reltool:stop(Server).
ok
```

### 1.3.3 Create a target system

```
11> file:list_dir("TARGET_DIR").
{error,enoent}
12>  reltool:create_target([{config, {sys, [{app, sasl, [{incl_cond, include}]},
                                           {boot_rel, "NAME"},
                                           {rel, "NAME", "VSN", [kernel, stdlib, sasl]}]}}],
                          "TARGET_DIR").
ok
13> file:list_dir("TARGET_DIR").
{ok,["bin","erts-5.7","lib","releases"]}
14> file:list_dir("TARGET_DIR/lib").
{ok,["erts-5.7","tools-2.6.3","syntax_tools-1.6",
     "stdlib-1.16","sasl-2.1.6","kernel-2.13","hipe-3.7",
     "compiler-4.6"]}
15> file:list_dir("TARGET_DIR/erts-5.7").
{ok,["bin"]}
16> file:list_dir("TARGET_DIR/releases").
{ok,["VSN","start_erl.data"]}
```

# Reltool Reference Manual

## Short Summaries



### reltool

The following functions are exported:

# reltool

Erlang Module

This is an interface module for the Reltool application

*Reltool* is a release management tool. It analyses a given Erlang/OTP installation and determines various dependencies between applications. The `graphical` frontend depicts the dependencies and enables interactive customization of a target system. The backend provides a `batch` interface for generation of customized target systems.

The tool uses an installed Erlang/OTP system as input. `root_dir` is the root directory of the analysed system and it defaults to the system executing `reltool`. Applications may also be located outside `root_dir`. `lib_dirs` defines additional library directories where applications additional may reside and it defaults to the the directories listed by the operating system environment variable `ERL_LIBS`. See the module `code` for more info. Finally single modules and entire applications may be read from Escripts. The names of the Escripts are given in the configuration parameter `escripts`. By default, no Escripts are included.

Some configuration parameters control the behavior of Reltool on system (`sys`) level. Others provide control on application (`app`) level and yet others are on module (`mod`) level. Module level parameters overrides application level parameters and application level parameters overrides system level parameters.

The following top level `options` are supported:

`config` This is the main option and it controls the configuration of `reltool`. It can either be a `sys` tuple or a name of a `file` containing a sys tuple.

`trap_exit` This option controls the error handling behavior of `reltool`. By default the window processes traps exit, but this behavior can altered by setting `trap_exit` to `false`.

`wx_debug` This option controls the debug level of `wx`. As its name indicates it is only useful for debugging. See `wx:debug/1` for more info.

Besides the already mentioned source parameters `root_dir`, `lib_dirs` and `escripts`, the following system (`sys`) level options are supported:

`erts` Erts specific configuration. See application level options below.

`app` Application specific configuration. An application has a mandatory name and application level options that are described below.

`mod_cond` This parameter controls the module inclusion policy. It defaults to `all` which means that if an application is included (either explicitly or implicitly) all modules in that application will be included. This implies that both modules that exists on the `ebin` directory of the application, as well as modules that are named in the `app` file will be included. If the parameter is set to `ebin`, both modules on the `ebin` directory and derived modules are included. If the parameter is set to `app`, both modules in the `app` file and derived modules are included. `derived` means that only modules that are used by other included modules are included. The `mod_cond` setting on system level is used as default for all applications.

`incl_cond` This parameter controls the application inclusion policy. It defaults to `derived` which means that the applications that not have any explicit `incl_cond` setting, will only be included if any other (explicitly or implicitly included) application uses it. The value `include` implies that all applications that that not have any explicit `incl_cond` setting will be included. `exclude` implies that all applications that that not have any explicit `incl_cond` setting will be excluded.

`boot_rel` A target system may have several releases but the one given as `boot_rel` will be used as default when the system is booting up.

`rel` Release specific configuration. Each release maps to a `rel`, `script` and `boot` file. See the module `systools` for more info about the details. Each release has a name, a version and a set of applications with a few release specific parameters such as type and included applications.

`app_file` This parameter controls the default handling of the `app` files when a target system is generated. It defaults to `keep` which means that `app` files are copied to the target system and their contents are kept as they are. `strip` means that a new `app` file is generated from the contents of the original `app` file where the non included modules are removed from the file. `all` does also imply that a new `app` file is generated from the contents of the original `app` file, with the difference that all included modules are added to the file. If the application does not have any `app` file a file will be created for `all` but not for `keep` and `strip`.

`debug_info` The `debug_info` parameter controls whether the debug information in the beam file should be kept (`keep`) or stripped `strip` when the file is copied to the target system.

`incl_erts_dirs` By default only the `bin` directory is copied to the target system for erts. This parameter controls if other directories should be copied. erts may optionally have an application directory containing `erl` and `beam` files for preloaded code. Which application directories that shall be copied are controlled with `incl_app_dirs` and `excl_app_dirs`.

`excl_erts_dirs` This parameter controls which erts directories that not should be copied to the target system. In order to be copied, a directory must be included in `incl_erts_dirs` and NOT be included in `excl_erts_dirs`.

`incl_app_dirs` By default only the application directories `ebin` and `priv` are copied to the target system. This parameter controls if other directories should be copied.

`excl_app_dirs` This parameter controls which applications directories that not should be copied to the target system. In order to be copied, a directory must be included in `incl_app_dirs` and NOT be included in `excl_app_dirs`.

On application (`app`) level,the following options are supported:

`vsn` The version of the application. In an installed system there may exist several versions of an application. The `vsn` parameter controls which version of the application that will be choosen. If it is omitted, the latest version will be choosen.

`mod` Module specific configuration. A module has a mandatory name and module level options that are described below.

`mod_cond` The value of this parameter overrides the parameter with the same name on system level.

`incl_cond` The value of this parameter overrides the parameter with the same name on system level.

`app_file` The value of this parameter overrides the parameter with the same name on system level.

`debug_info` The value of this parameter overrides the parameter with the same name on system level.

`incl_app_dirs` The value of this parameter overrides the parameter with the same name on system level.

`excl_app_dirs` The value of this parameter overrides the parameter with the same name on system level.

On module (`mod`) level,the following options are supported:

`incl_cond` This parameter controls whether the module is included or not. By default the `mod_incl` parameter on application and system level will be used to control whether the module is included or not. The value of `incl_cond` overrides the module inclusion policy. `include` implies that the module is included, while `exclude` implies that the module not is included. `derived` implies that the is included if any included uses the module.

`debug_info` The value of this parameter overrides the parameter with the same name on application level.

## DATA TYPES

```
options()        = [option()]
option()         = {config, config() | file()}
                 | {trap_exit, bool()}
                 | {wx_debug, term()}
config()         = {sys, [sys()]}
sys()            = {root_dir, root_dir()}
                 | {lib_dirs, [lib_dir()]}
                 | {escripts, [escript()]}
                 | {erts, app()}
                 | {app, app_name(), [app()]}
                 | {mod_cond, mod_cond()}
                 | {incl_cond, incl_cond()}
                 | {boot_rel, boot_rel()}
                 | {rel, rel_name(), rel_vsn(), [rel_app()]}
                 | {app_file, app_file()}
                 | {debug_info, debug_info()}
                 | {incl_erts_dirs, [incl_erts_dir()]}
                 | {excl_erts_dirs, [excl_erts_dir()]}
                 | {incl_app_dirs, [incl_app_dir()]}
                 | {excl_app_dirs, [excl_app_dir()]}
app()            = {vsn, app_vsn()}
                 | {mod, mod_name(), mod()}
```

```
                        | {mod_cond, mod_cond()}
                        | {incl_cond, incl_cond()}
                        | {debug_info, debug_info()}
                        | {app_file, app_file()}
                        | {incl_app_dirs, [incl_app_dir()]}
                        | {excl_app_dirs, [excl_app_dir()]}
              mod()     = {vsn, app_vsn()}
                        | {incl_cond, incl_cond()}
                        | {debug_info, debug_info()}
              rel_app() = app_name()
                        | {app_name(), app_type()}
                        | {app_name(), [incl_app()]}
                        | {app_name(), app_type(), [incl_app()]}
              app_name()       = atom()
              app_type()       = permanent | transient | temporary | load | none
              app_vsn()        = string()
              boot_rel()       = rel_name()
              app_file()       = keep | strip | all
              debug_info()     = keep | strip
              dir()            = string()
              escript()        = file()
              escript_arg()    = string()
              excl_app_dir()   = dir()
              excl_erts_dir()  = dir()
              file()           = string()
              incl_app()       = app_name()
              incl_app_dir()   = dir()
              incl_cond()      = include | exclude | derived
              incl_erts_dir()  = dir()
              lib_dir()        = dir()
              mod_cond()       = all | app | ebin | derived | none
              mod_name()       = atom()
              reason()         = string()
              rel_file()       = term()
              rel_name()       = string()
              rel_vsn()        = string()
              root_dir()       = dir()
              script_file()    = term()
              server()         = server_pid() | options()
              server_pid()     = pid()
              window_pid()     = pid()
```

## Exports

`create_target(Server, TargetDir) -> ok | {error, Reason}`

Types:

- Server = server()
- TargetDir = dir()
- Reason = reason()

Create a target system. In order to be able to run the target system, it must be installed first.

`get_config(Server) -> {ok, Config} | {error, Reason}`

Types:

- Server = server()
- Config = config()
- Reason = reason()

Get reltool configuration. Only configuration parameters with values that differs from their defaults are returned.

`get_rel(Server, Relname) -> {ok, RelFile} | {error, Reason}`

Types:

- Server = server()
- RelName = rel_name()
- RelFile = rel_file()
- Reason = reason()

Get contents of a release file. See `rel(4)` for more details.

`get_script(Server, Relname) -> {ok, ScriptFile | {error, Reason}`

Types:

- Server = server()
- RelName = rel_name()
- ScriptFile = script_file()
- Reason = reason()

Get contents of a boot script file. See `script(4)` for more details.

`install(Server, TargetDir) -> ok | {error, Reason}`

Types:

- Server = server()
- TargetDir = dir()
- Reason = reason()

Install a created target system

`start(Options) -> {ok, WindowPid} | {error, Reason}`

Types:

- Options = options()
- WindowPid = window_pid()
- Reason = reason()

Start a main window process with options

`start_server(Options) -> {ok, ServerPid} | {error, Reason}`

Types:

- Options = options()
- ServerPid = server_pid()
- Reason = reason()

Start a server process with options. The server process identity can be given as argument to several other functions in the API.

`stop(Pid) -> ok | {error, Reason}`

Types:

- Pid = server_pid() | window_pid()()
- Reason = reason()

Stop a server or window process

# Index of Modules and Functions

Modules are typed in *this way*.
Functions are typed in `this way`.