

# How to create an application for RaidSonic's NAS-2000

Philipp Wehrheim

27.11.2007

Version 1.0

## Abstract

This Howto explains how to create an optional software package for the NAS-1000, NAS-2000 and NAS-4220.

This document was created for RaidSonic GmbH.  
Copyright © 2007 by RaidSonic GmbH.

All rights reserved.

## Contents

<b>1</b>	<b>About this document</b>	<b>3</b>
<b>2</b>	<b>Walk through the package installation</b>	<b>4</b>
<b>3</b>	<b>Package structure</b>	<b>5</b>
<b>4</b>	<b>Streamripper</b>	<b>9</b>
<b>5</b>	<b>Creating the Package</b>	<b>13</b>

# 1 About this document

The NAS-2000 firmware version 2.3.2.IB.2.RS.1 and later support the installation of additional software packages created either by third party companies or by end users.

This Howto shows you how to create such a software package. The document is divided into 5 sections which will explain how the structure of a package is supposed to look like and what files are expected by the firmware.

We will create a package that will add a streamripper application to the NAS. By adding a streamripper you are able to record music or (web)radio shows and store them on the NAS. But as this is only an example of what is possible with optional packages you could create different packages in order to make the NAS your homeserver and customise it to meet your needs.

Please note that it is assumed that the reader has some knowledge about compiling C programmes and how to apply a patch to sourcecode.

## 2 Walk through the package installation

We will begin with the install-procedure of the example-application on the NAS. This way it is easy to understand how the structure of a package is supposed to look like and what is happening with (new) packages at the system startup.

```
/mnt/IDE1/public
|
|-- applications
|
|-- new_software
```

Listing 1: The relevant parts of the filesystem (on the NAS) for our package.

As you can see there is an "application"-directory below the public directory. It will be the place where all the optional applications are located.

In order to install new applications the user has to copy the application archive "streamripper.tgz" into the new\_software directory.

```
/mnt/IDE1/public
|
|-- applications
|
|-- new_software
|
|-- streamripper.tgz
```

Listing 2: Installing a new package

The next thing in order to activate the new app is to restart the NAS.

During the bootprocess the firmware checks if the new\_software directory contains any archives.

Note: It is very important that the archive's name ends with ".tgz" !

It is also recommended that the archive-name contains no whitespaces.

If the bootscript finds a valid archive it will extract it into the application directory and delete the archive from the new\_software directory.

```
/mnt/IDE1/public
|
|-- applications
|
|-- streamripper
|-- new_software
```

Listing 3: New directory created after a reboot

After that, the bootscript iterates through all directories in the application-directory and searches for an init script/program. The init script is responsible for setting up the environment the application needs to work correct, like: create symlinks, copy shared libraies, etc.

Note: The init script/program should have the executable bit set and should be owned by the user admin! Once all the init scripts/programs are executed the bootscript continues with the system start.

### 3 Package structure

The next step is to understand how the structure of a package should look like. Our example application streamripper will have the following directory structure:

```
/mnt/IDE1/public/applications/streamripper/  
|  
|-- bin  
|-- lib  
`-- webroot  
    |  
    |-- cgi  
    |   `-- streamripper  
    |-- html  
    |   `-- streamripper  
    `-- nav  
        `-- streamripper
```

Listing 4: Structur of the example package

You may ask now: "What are all the different directories for?"

- The bin directory contains all the executable files.
- The lib directory contains shared libraries that are needed by the executibles.
- The webroot and its subdiretories are containing the files that are needed to register in the WebGUI in order to give the user the possiblility to start/stop, activate/deactive the new application.

In this example the html directory is not needed and will be left empty.

Note: It is very important that all the directories below the webroot have the same names for one application and that there is no other application that is using the same namespace as the application we are just creating.

WebGUI structure:

The WebGUI is generated by a software called "Sausalito". Sausalito comes with a special directory structure that makes it very easy to add or remove entries in the WebGUI.

We will begin with adding a new element to the menu on the left side of the WebGUI. For archiving this, by a simple xml file is used.

```
<item id="streamripper" label="Streamripper" \
description="" \
url="cgi/streamripper/streamripper.cgi" \
parent="network_service" order="42"/>
```

Listing 5: webroot/nav/streamripper/streamripper.xml

The `item` element will be displayed in the WebGUI menu. The url `cgi/streamripper/streamripper.cgi` is the file that will be called if the user clicks on our new menu entry. The parent element will tell Sausalito who is the parent of the menu entry and where to create the subentry. Finally the order value is responsible for the order of all menu entries. The lowest number will be the first entry in the menu and the highest number will be the last.

Then you have to create the CGI-script that is called whenever an user pushes the menu button.

In this example the CGI-script is a simple Shell-script. The CGI-script is responsible for reading the user input, starting and stopping streamripper and displaying the progress of the recording.

```
#!/bin/sh

# process post data
if [ "$REQUEST_METHOD" = "POST" ]; then
  $(dd count=$CONTENT_LENGTH bs=1 2>/dev/null of=/tmp/sr_post_data)
  . /tmp/sr_post_data
  if [ $url ]; then
    $(echo "s/%3F/?/g;s/%40/@/g;s/%24/$/g;s/%25/%/g;s/%26/\&/g;s/%3D/=/g; \
s/%2F///g;s/%3A/:/g" > /tmp/sr_sed;
    url=`echo $url | sed -f /tmp/sr_sed`;
    echo "url=\"$url\"" > /tmp/sr_url;
    (0<&- 2<&- 1<&- /mnt/IDE1/public/applications/streamripper-app/bin/
    streamripper \
    "`echo $url`" -d /mnt/IDE1/public/music/ &))
    # the last line tells streamripper the hard way to
    # disconnect from std{in,out and error}
  else
    $(killall streamripper; \
    sleep 1; \
    killall streamripper; \
    sleep 1; \
    killall streamripper; \
    rm -f /tmp/sr_post_data; \
    rm -f /tmp/sr_status_data; \
    rm -f /tmp/sr_url)
  fi
fi

#
# HTML start
echo -e "Content-type: text/html\n"
echo "<html><head><title>Streamripper Settings</title><link rel=\"StyleSheet\"
type=\"text/css\" href=\"/css/storlib.css\">
<script language=\"JavaScript\" type=\"text/javascript\">
<!-- Start Script
function checkurl () {
```

```

if (document.form.url.value == "") {
    alert( "Please enter a valide URL." );
    return false;
}
form.url.replace(/;/g)
return true;
}"
#
# test if streamripper is running
sr_on=`ps -eaf | grep "streamripper" | grep -v grep | grep -v streamripper.cgi`
#
# source /tmp/sr_post_data if exists
[ -e /tmp/sr_post_data ] && . /tmp/sr_url
#
# source the status file
[ -e /tmp/sr_status ] && . /tmp/sr_status
#
# if streamripper is not active let the user enter the url
if [ ${#sr_on} -eq 0 ]; then
    unset $sr_on
    echo "//End Script --></script></head>"
    echo "<body><form name=\"form\" method=\"post\" action=\"streamripper.cgi\" \
        onsubmit=\"return checkurl()\">"
    echo "<table border=\"0\" width=\"600\" cellpadding=\"0\" \
        cellspacing=\"2\">"
    echo "<tr><td colspan=\"2\" class=\"TITLE\">Streamripper Settings</td></tr>"
    echo "<tr><td class=\"HEAD\">Streamserver URL:</td>"
    echo "<td>"
    echo "<input type=\"text\" name=\"url\" size=\"70\">"
    echo "</td></tr>"
    echo "<tr><td colspan=\"2\"><p align=\"center\"><input \
        type=\"submit\" value=\"Record\" \
        >&nbsp;  </td></tr>"
#
# if streamripper is active disable the textarea and show the stop-button
else
    echo "//End Script --></script>"
    echo "<meta http-equiv=\"refresh\" content=\"7\" URL=\"streamripper.cgi\"></ \
        head>"
    echo "<body><form name=\"form\" method=\"post\" action=\"streamripper.cgi\">"
    echo "<table border=\"0\" width=\"600\" cellpadding=\"0\" \
        cellspacing=\"2\">"
    echo "<tr><td colspan=\"2\" class=\"TITLE\">Streamripper Settings</td></tr>"
    echo "<tr><td class=\"HEAD\">Streamserver URL:</td>"
    echo "<td>"
    echo "<input type=\"text\" name=\"url\" disabled size=\"70\" value=\"\$url\"> \
        </td></tr>"
    <tr><td colspan=\"2\"><p align=\"center\"><input \
        type=\"submit\" value=\"Stop\" \
        >&nbsp;  </td></tr>"
    echo "<tr><td class=\"TITLE1\" colspan=\"2\"><b>Status:</b> $STATUS</td></tr> \
        <tr><td class=\"TITLE1\" colspan=\"2\"><b>Server:</b> $SERVER</td></tr> \
        <tr><td class=\"TITLE1\" colspan=\"2\"><b>Trackname:</b> $TRACKNAME <b> \
        Bitrate:</b> \
        $BITRATE kbit/s</td></tr>"

```

```

fi
echo "</table></form>"
echo "</body></html>"

```

Listing 6: webroot/cgi/streamripper/streamripper.cgi

When the CGI-script is called it checks, if the the REQUEST\_METHOD contains the value POST. If this is true, the user must have clicked on the record-button and the script will store the submitted values in /tmp/sr\_post\_data. After the file is written, it is sourced (read into the script). Moreover, it tests if the url-variable is defined. If so, all the special characters (% , @ , & , ? , \$ , etc.) will be restored into the original characters (%40 into @).

The execution of echo for every line is not a performace boost, but makes it easy to understand what the cgi script is doing. For faster execution the "Here Document" technic is recommended.

Note: Because the NAS is not using the fullblown GNU-commands, but stripped down busybox commands, not all features and flags can be used on the NAS.

For this example we don't need additional html files, so the html directory will be empty.

As mentioned earlier, every package should have an init script in its root directory that will be executed during the NAS bootup.

Since most of the filesystem is located on a ramdisk, all changes made will be lost when the NAS is rebooted. Therefore we will create symlinks from the packages directory to the filesystem at every boot.

Note: As the webserver will refuse to read files and follow symlinks outside of the webroot, we have to copy these files into the webroot.

For the streamripper example the init script looks like this:

```

#!/bin/sh
#
# This init script will create all symlinks
# needed by streamripper to be noticed from sausalito.
#
#
# Written by Philipp Wehrheim <flip at flipstar dot net> 20070808
# for RaidSonic GmbH
#
# This file is licenced under the GLP.
#
#
# get the hd mountpoint
HD_MNT_POINT=$(cat /usr/sausalito/codb/objects/1/Disk.rootdir 2> /dev/null)
#
# if $HD_MNT_POINT is empty we are in trouble -> exit
if [ -z $HD_MNT_POINT ]; then
    exit -1
fi
#
#
CWD=$HD_MNT_POINT/public/applications/streamripper-app
#
# create symlinks for sharedlibs
#

```

```

# libpthread
ln -s $CWD/lib/libpthread.so /lib/libpthread.so
# libogg
ln -s $CWD/lib/libogg.so.0.5.3 /lib/libogg.so.0
# libvorbis
ln -s $CWD/lib/libvorbis.so.0.3.1 /lib/libvorbis.so.0
# libm
ln -s $CWD/lib/libm.so.6 /lib/libm.so.6

# register at the webroot
#
# NOTE: The webserver will refuse to read files outside of
#       the Webroot so we have to copy our cgi and nav files into the webroot.
#
if [ ! -e /usr/webroot/cgi/streamripper ]; then
    cp -a $CWD/webroot/cgi/streamripper /usr/webroot/cgi/streamripper
fi
if [ ! -e /usr/webroot/nav/streamripper ]; then
    cp -a $CWD/webroot/nav/streamripper /usr/webroot/nav/streamripper
fi

```

Listing 7: streamripper init-script

The init-script is executed when the NAS is booting. It will first create symlinks for every shared library the binary needs. After that the package's "HTML- and CGI-files" are copied into the webserver's webroot.

## 4 Streamripper

The streamripper project was started by Jon Clegg. The aim of streamripper is to record a lifestream (like your favourite radio show) to the local harddisk. So you can listen to it whenever you want to.

In order to make streamripper work with the CGI-script some little changes have to be made in the sourcecode.

The following diff shows the changes that are needed for streamripper version 1.62.1. After downloading the archive we have to apply the patches and compile the streamripper. Since the NAS has an ARM-CPU streamripper has to be compiled with a crosscompiler. A complete build environment can be downloaded from the RaidSonic website.

```

--- console/cstreamripper.c 2007-08-13 20:58:15.000000000 +0200
+++ console/cstreamripper.c 2007-08-13 21:00:11.000000000 +0200
@@ -47,7 +47,8 @@
 static BOOL
 static BOOL
 static BOOL
-static BOOL
+static BOOL
+static BOOL
 RIP_MANAGER_OPTIONS
 time_t
         m_started = FALSE;
         m_alldone = FALSE;
         m_got_sig = FALSE;
         m_dont_print = FALSE;
         m_file_print = TRUE;
         m_dont_print = TRUE;
         m_opt;
         m_stop_time = 0;

@@ -73,6 +74,18 @@
     parse_arguments(argc, argv);
     if (!m_dont_print)
         fprintf(stderr, "Connecting...\n");

```

```

+
+   if (m_file_print) {
+       FILE *fp;
+       if ((fp = fopen("/tmp/sr_status", "w")) != NULL) {
+           fprintf(fp, "SERVER=\"%-\"\\n"
+                   "STATUS=\"connecting\"\\n"
+                   "TRACKNAME=\"%-\"\\n"
+                   "BITRATE=\"%-\"\\n");
+           fclose(fp);
+       }
+   }
+
+   if ((ret = rip_manager_start(rip_callback, &m_opt)) != SR_SUCCESS) {
+       fprintf(stderr, "Couldn't connect to %s\\n", m_opt.url);
+       exit(1);
@@ -98,6 +111,18 @@
+   if (!m_dont_print) {
+       fprintf(stderr, "shutting down\\n");
+   }
+
+   if (m_file_print) {
+       FILE *fp;
+       if ((fp = fopen("/tmp/sr_status", "w")) != NULL) {
+           fprintf(fp, "SERVER=\"%-\"\\n"
+                   "STATUS=\"shutting down\"\\n"
+                   "TRACKNAME=\"%-\"\\n"
+                   "BITRATE=\"%-\"\\n");
+           fclose(fp);
+       }
+   }
+
+   /* GCS: Why? */
+   #if defined (commentout)
+   m_dont_print = TRUE;
@@ -130,6 +155,36 @@
+   static int buffering_tick = 0;
+   BOOL static printed_fullinfo = FALSE;
+
+   if (m_file_print) {
+       /* muss nach oben define */
+       FILE *fp;
+       switch(m_curinfo.status)
+       {
+       case RM_STATUS_BUFFERING:
+           if ((fp = fopen("/tmp/sr_status", "w")) != NULL) {
+               fprintf(fp, "SERVER=\"%s\"\\n"
+                       "STATUS=\"buffering\"\\n"
+                       "TRACKNAME=\"%-\"\\n"
+                       "BITRATE=\"%-\"\\n"
+                       ,m_curinfo.streamname);
+               fclose(fp);
+           }
+           break;
+
+       case RM_STATUS_RIPPING:

```

```

+         if ((fp = fopen("/tmp/sr_status", "w")) != NULL) {
+             fprintf(fp, "SERVER=\"%s\"\n"
+                 "STATUS=\"ripping\"\n"
+                 "TRACKNAME=\"%s\"\n"
+                 "BITRATE=\"%d\"\n"
+                 ,m_curinfo.streamname, m_curinfo.filename,
+m_curinfo.bitrate);
+             fclose(fp);
+         }
+         break;
+     }
+ }
+
+ if (m_dont_print)
+     return;

```

Listing 8: streamripper diff needed for the CGI-support

It is quite easy to see that a status file (/tmp/sr\_status) is created which contains the actual streamripper status. It is not possible to simply redirect the output of streamripper to get the status, since the filedescriptors are never closed, as long as streamripper is running. The CGI-script would hang and not update until streamripper ends.

Another important thing to note is that the user that manages the NAS is usually the user admin and not the user root. Consequently we have to make sure that the files created by streamripper have proper permissions. Because the NAS got no command to set, the umask another small patch is affordable.

```

--- lib/filelib.c 2007-08-13 21:12:56.000000000 +0200
+++ lib/filelib.c 2007-08-13 21:12:50.000000000 +0200
@@ -119,6 +119,7 @@
     mkdir (s);
 #else
     mkdir (s, 0777);
+   chmod(s, S_IRWXU | S_IRWXG | S_IRWXO );
 #endif
     return SR_SUCCESS;
 }
@@ -1032,6 +1033,8 @@
     // printf ("ERROR creating file: %s\n",filename);
     return SR_ERROR_CANT_CREATE_FILE;
 }
+ /* we always set mod 666 -> this is insecure !!! */
+ chmod(fn, S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH);
 #endif
     return SR_SUCCESS;
 }

```

Listing 9: streamripper diff to set the right permissions

With this patch all directories created by streamripper will have mode "777" aka readable, writeable and executable by everybody which is NOT secure. Also the files stored by streamripper can be read and written by everybody.

Note: If a crosscompiler is used, you have to make sure that the right compiler is used. This is usually done by exporting some environment variables.

After streamripper is compiled, we have to make sure that not only the compiled program is copied into our package, but also the shared libraries. The easiest way to find out which libraries are used by streamripper is to run either *ldd* or *readelf*.

```

streamripper-1.62.1 # ldd streamripper
libpthread.so.0 => /lib/libpthread.so.0 (0x3fac9000)
libogg.so.0 => /usr/lib/libogg.so.0 (0x3fb23000)
libvorbis.so.0 => /usr/lib/libvorbis.so.0 (0x3fb30000)
libm.so.6 => /lib/libm.so.6 (0x3fb57000)
libc.so.6 => /lib/libc.so.6 (0x3fbd2000)
/lib/ld-linux.so.2 (0x3faab000)

```

Listing 10: list of shared libraries needed by streamripper

By running *ldd* or *readelf* all dependencies of streamripper are listed. So the next step is to locate the libraries. There is one library that we don't have to care about because it is already on the NAS. It is the *libc.so.6* library.

The relevant libraries are:

- libpthread.so.0
- libogg.so.0
- libvorbis.so.0
- libm.so.6

Another thing that can be done in order to save space on the harddisk and on the ram is to strip the binary program we have just compiled. Stripping will remove global symbols which aren't involved in relocation.

```
strip --strip-unnneeded streamripper
```

Listing 11: strip streamripper to save space

## 5 Creating the Package

Now that we know all the steps needed to construct a package, let's put them all together and build the streamripper package. We will begin with creating all the directories. After that, we start to copy the relevant files into the different directories.

```
mkdir -p streamripper-app/{bin,lib,webroot}
```

Listing 12: package directory structure

After we have created the basic packet structure, we can begin with populating the directories with our files. Copy the streamripper binary into the bin directory and the shared libraries into lib. It is important to note that libraries are often symlinks to another file. Before you start to copy the libraries, make sure that you don't copy a link but, the file the symlink is pointing to!

```
cp <path>/streamripper streamripper-app/bin/streamripper  
cp <path>/<lib name> streamripper-app/lib/<lib name>
```

Listing 13: copy the binary and libraries into the package structur

Optionally you can create a VERSION-file that contains the version of the package.

```
VERSION="1.0"
```

Listing 14: version file

The next step is to make the directories for the webcontent.

```
mkdir -p streamripper-app/webroot/cgi/streamripper  
mkdir -p streamripper-app/webroot/nav/streamripper
```

Listing 15: package directory structur

Once the directories are created the streamripper.xml file and the CGI-script have to be copied into place.

```
cp <path>/streamripper.xml streamripper-app/webroot/nav/streamripper/  
cp <path>/streamripper.cgi streamripper-app/webroot/cgi/streamripper/
```

Listing 16: populate the webroot directory

Note: It is very important that the streamripper binary and the CGI-script have the executable bit set!

After all files are copied into the streamripper package, structure we have to make sure that all files of the package are owned by the user admin (ID=500).

```
chown -R 500 ./streamripper-app
```

Listing 17: make user admin the owner of all files

Ok, that was it. We have just created an (optional) application for the RaidSonic's NAS 2000. The very last thing to do is to build a tar-archive that will be copied onto the NAS. The archive will be created by the following command.

```
tar -czvf ./streamripper-app`date +%Y%m%d`.tgz ./streamripper-app
```

Listing 18: create the streamripper tar-archive

As explained at the beginning of this document the archive can now be copied onto the NAS and saved in the new \_application directory. During the next boot the firmware will extract and install the application.

I hope this Howto helps you to develop applications for the NAS in order to make the NAS meet your needs.

Have fun.