**AutoTools HowTo - Persistence**

***Table of contents***

1. ***Generic information - autotools***
2. ***Build pers-common library with autotools***
   1. ***Informative text files***
   2. ***How to write configure.ac + Makefile.am***
      1. ***configure.ac – general information***
      2. ***configure.ac – pers-common***
      3. ***Makefile.am – general information***
      4. ***Makefile.am – pers-common***
   3. ***pkg-config***
      1. ***General information***
      2. ***Integrate pkg support in configure.ac and Makefile.am***
   4. ***autoreconf --force --install***
   5. ***Build and install dependencies: itzam***
   6. ***./configure***
   7. ***make***
   8. ***make install***
3. ***Build persistence-administration-service with autotools***
   1. ***Build & install dependencies: automotive-dlt, libarchive, zlib***
   2. ***Build persistence-administrator***
4. ***Cross compile for arm-montavista***
5. ***Final projects: persistence-administrator & persistence-common***
6. ***References and recommended reading***
7. ***Generic information - autotools***

The **automake** and **autoconf** tools are used to manage C/C++ projects under Unix. They should save a lot of time generating **Makefiles** to build a project compared to writing **Makefiles** and **configure** scripts manually, while ensuring that the project is structured according to [GNU standards](http://www.gnu.org/software/automake/manual/standards/Makefile-Conventions.html).

**autoconf** looks for a file called configure.ac (or configure.in for backward compatibility). It then runs the M4 macro processor to create the configure script.

**automake** looks for a file called Makefile.am. It then creates a Makefile.in, based on the macros which it finds. This is later used by the configure script to create Makefiles.

The **configure** script is used to aid cross-platform compiling. A suitable **configure** script should interpret a **Makefile.in** file and then create a platform-specific **Makefile** file. It will do this after performing several tests to determine the characteristics of the platform.

This allows one to type ./configure and then make to compile a project on one’s system.

The **make** tool is used to manage multi-file projects. **make** uses the **Makefile** in the project folder, which lists the various compiling and linking steps, targets, and dependencies.

In short, the following commands should be the only ones to be run to compile the project using autotools:

**autoreconf --force --install**: runs aclocal, autoconf, autoheader and automake in the right order to create config.h.in, Makefile.in, configure and a number of auxiliary files

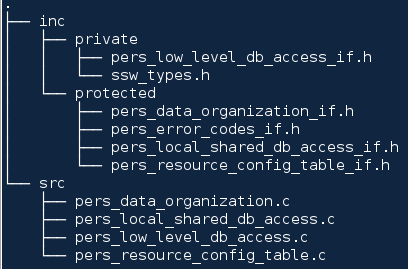
**./configure**: creates Makefile from Makefile.in and config.h from config.h.in

**make**

**make install**

1. ***Build pers-common library with autotools***

Initial structure of persistence-common directory:



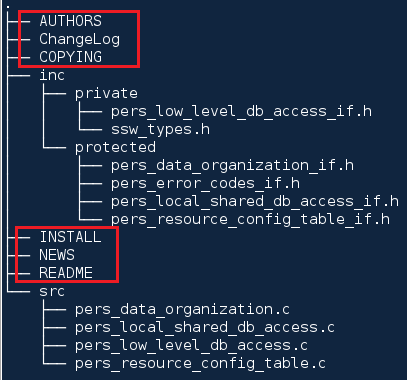
What we want to obtain is a shared library named libperscommon and to install it in /usr/lib to be later used by persistence-administrator application.

In order to achieve this, the following STEPS should be performed to compile the above structure with autotools:

* 1. ***Informative text files***

Because automake tries to make a GNU-style project by default, it will add a COPYING file and complain if some other necessary informative text files are missing (NEWS README AUTHORS ChangeLog). Create the text files in the root of persistence-common directory and name them accordingly.

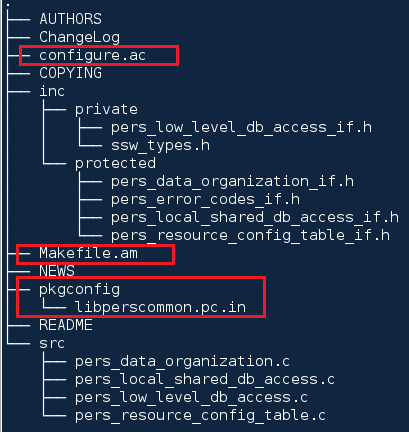
The structure of the persistence-common library after performing this step should look like:



* 1. ***How to write configure.ac + Makefile.am***

Create input files configure.ac + Makefile.am in the root of the persistence-common directory.

Additionally, create a directory named pkgconfig and create an empty file libperscommon.pc.in inside (the need for this file will be explained later on).



* + 1. ***configure.ac – general information***

Every configure.ac should have at least lines like the following:

**AC\_INIT**([Hello], [0.1], [bug-report@hello.example.com], [hello], [http://hello.example.com/])

**AC\_PREREQ**([2.59])

**AM\_INIT\_AUTOMAKE**([1.10 no-define])

**AC\_PROG\_CXX**

**AC\_CONFIG\_FILES**([Makefile])

**AC\_OUTPUT**

The [**AC\_INIT** macro](http://www.gnu.org/software/autoconf/manual/html_node/Initializing-configure.html) initializes autoconf with information about the project, including the project’s name, version number, bug-reporting address, tarball name and the project’s homepage.

The [**AM\_INIT\_AUTOMAKE**](http://www.gnu.org/software/automake/manual/html_node/Public-Macros.html#index-AM_005fINIT_005fAUTOMAKE-284) line adds several standard checks and initializes automake.

[**AC\_PROG\_CXX**](http://www.gnu.org/software/autoconf/manual/html_node/C_002b_002b-Compiler.html#index-AC_005fPROG_005fCXX-914) checks for a C++ compiler. If the project uses C, one can check for a C compiler with [AC\_PROG\_CC](http://www.gnu.org/software/autoconf/manual/html_node/C-Compiler.html#index-AC_005fPROG_005fCC-842).

[**AC\_CONFIG\_FILES**](http://www.gnu.org/software/autoconf/manual/html_node/Configuration-Files.html) lists the files to be generated by configure. By default, each file is generated from a template file of the same name but with an .in extension appended.

[**AC\_OUTPUT**](http://www.gnu.org/software/autoconf/manual/html_node/Output.html) finishes configure processing, and generates the output files.

* + 1. ***configure.ac – pers-common***

First, define all of the version numbers up front. In particular, this allows the version macro to be used in AC\_INIT.

|  |
| --- |
| m4\_define([LIBPERSOCOMMON\_VERSION\_S] |
| m4\_define([LIBPERSOCOMMON\_VERSION\_N] |
|  |

Initialize autoconf

|  |
| --- |
| **AC\_INIT**([persistence-common], LIBPERSOCOMMON\_VERSION\_S()) |
| AC\_COPYRIGHT([Copyright (c) 2012 Continental Automotive GmbH]) |
|  |

The least version autoconf should have

|  |
| --- |
| AC\_PREREQ([2.50]) |
|  |

Initialize automake

|  |
| --- |
| **AM\_INIT\_AUTOMAKE**() |
| AC\_CONFIG\_MACRO\_DIR([m4]) |
| AC\_SUBST([ACLOCAL\_AMFLAGS], ["$ACLOCAL\_FLAGS -I m4"]) |
|  |

Version the shared library

|  |
| --- |
| PERSCOMMON\_MAJOR=$((((LIBPERSOCOMMON\_VERSION\_N() / 1000) / 1000) % 1000 )) |
| PERSCOMMON\_MINOR=$(( (LIBPERSOCOMMON\_VERSION\_N() / 1000) % 1000 )) |
| PERSCOMMON\_MICRO=$(( LIBPERSOCOMMON\_VERSION\_N() % 1000 )) |
| GENERIC\_LIBRARY\_VERSION=$PERSCOMMON\_MAJOR:$PERSCOMMON\_MINOR:$PERSCOMMON\_MICRO |
|  |

The following line defines the GENERIC\_LIBRARY\_VERSION variable, which shall later be used in Makefile.am to version the library (e.g. libperscommon.0.PERSCOMMON\_MINOR. PERSCOMMON\_MICRO)

|  |
| --- |
| AC\_SUBST(GENERIC\_LIBRARY\_VERSION) |
|  |

Check for basic programs

|  |
| --- |
| **AC\_PROG\_CC**() |
| AM\_PROG\_CC\_C\_O() |
| AC\_PROG\_INSTALL() |
|  |

Initialize lib tool – adds libtool support. GNU libtool is a generic library support script. Libtool hides the complexity of using shared libraries behind a consistent, portable interface.

To use libtool, add the new generic library building commands to your Makefile, Makefile.in, or Makefile.am (this shall later be explained when we create the Makefile.am file).

|  |
| --- |
| AC\_PROG\_LIBTOOL() |
|  |

Check for standard header files (listed here only to exemplify)

|  |
| --- |
| AC\_CHECK\_HEADERS([string.h]) |
|  |

Check for standard functions (listed here only to exemplify)

|  |
| --- |
| AC\_CHECK\_FUNCS([memcpy]) |
|  |

Check for typedefs

|  |
| --- |
| AC\_HEADER\_STDBOOL |
| AC\_TYPE\_SIZE\_T |
| AC\_TYPE\_UINT8\_T |
| AC\_TYPE\_UINT16\_T |
| AC\_TYPE\_UINT32\_T |
| AC\_TYPE\_UINT64\_T |
|  |

Generally, the needed headers, functions and types will be reported by autoscan. Autoscan will check your code for C functions and types that you use without referring in configure.ac.

Check for required packages and their corresponding versions & overwrite name of itzam library

|  |
| --- |
| PKG\_CHECK\_MODULES([GLIB], [glib-2.0 >= 2.30.0]) |
| PKG\_CHECK\_MODULES([ITZAM], [libitzam >= 6.0.4]) |
| ITZAM\_LIBS="-litzam" |
|  |

Define configure output

Generate Makefile

|  |
| --- |
| **AC\_CONFIG\_FILES**([Makefile]) |
|  |

Generate libperscommon.pc from libperscommon.pc.in inside pkgconfig directory

|  |
| --- |
| **AC\_CONFIG\_FILES**([pkgconfig/libperscommon.pc]) |
|  |

Finish configure processing

|  |
| --- |
| **AC\_OUTPUT** |
|  |

* + 1. ***Makefile.am – general information***

## *Automake Variables*

Strictly speaking, all Makefile.am files are simply Makefile.in files that follow special formatting rules. automake will copy the entire contents of each Makefile.am file into the corresponding Makefile.in file; the configure script will then translate that Makefile.in into a Makefile file, performing variable substitutions as it finds them.

Of course, if that's all automake did, it would be no more helpful than the cp command. automake adds quite a few makefile targets and rules of its own, based on the syntax of the variable names in Makefile.am. Some of these makefile targets exist to satisfy the GNU standards. Others exist to make the build system more robust and convenient. For example, automake adds dependencies on most of the generated files in the build system, such as the configure script, the Makefile.in and Makefile files, and even config.h. If any of these files have been touched, a simple make command will trigger a rebuild of the affected files. If you tweaked a line in a Makefile.am file in a subdirectory, make will rerun automake for you before continuing.

Part of what allows the automake system to express so much about the make process in so few commands is the elegance of its naming scheme. When automake parses Makefile.am, it looks for a special type of variable name. When it finds the special variables, it creates the necessary makefile targets to handle the contents of that variable properly.

These special variables take the form of target\_PRIMARY. target specifies where the contents are supposed to go; PRIMARY tells automake what to do with those contents. automake supports three general types of primaries, each of which determines what sort of contents the variable can contain: binary files, source files, and linker options.

The first type of primary declares binary targets, such as libraries and executables. A variable with the \_PROGRAMS primary holds a list of one or more executables; the \_LIBRARIES and \_LTLIBRARIES primaries both refer to libraries. Use \_LIBRARIES to create static-only libraries. The \_LTLIBRARIES primary invokes the libtool script to create static and dynamic libraries in a platform-independent manner.

The target name for these primaries should indicate the install location for the binary files listed in the variable's contents. The most common variations you'll use are bin\_PROGRAMS, lib\_LIBRARIES, and lib\_LTLIBRARIES. The bin and lib target names correspond directly to the $(bindir) and $(libdir) variables. Thus with the bin\_PROGRAMS variable you can specify a list of programs you want installed into whichever $(bindir) directory the administrator specifies at configure time. Here are a few examples:

|  |
| --- |
| bin\_PROGRAMS = myfirstapp mysecondapp  lib\_LIBRARIES = libdothis.a libdothat.a  lib\_LTLIBRARIES = libmylibtoollibrary.la |
|  |

You may not always want to install all of your binary files. Sometimes the build process works better if you create transient convenience libraries; if the source code for your target executable is spread across multiple subdirectories, you may have to build a static library for each subdirectory. However, you don't want to install these convenience libraries. They're part of the journey, not the destination. The noinst target exists for this very reason. Any files listed under noinst will be built but not installed. If libdothis.a and libdothat.a in the example here were convenience libraries, we would change the \_LIBRARIES declaration to this:

|  |
| --- |
| noinst\_LIBRARIES = libdothis.a libdothat.a |

The second category of primaries refers to lists of source code files. The \_SOURCES primary, one of the most commonly used primaries, lists source files that must be compiled into object files. These source files are then linked into an executable or library declared in a \_PROGRAMS, \_LIBRARIES, or \_LTLIBRARIES primary. You don't have to include any of your header files in Makefile.am unless you want to install them on the target system. In this case you list the public header files inside a \_HEADERS primary, using the same target convention we used for the \_PROGRAMS primary. So, if we had a myapp.h and an otherstuff.h file that we wanted to install into $(includedir), we would declare the following variable:

|  |
| --- |
| include\_HEADERS = myapp.h otherstuff.h |

The target directory concept is very simple and flexible. automake does not do any unusual processing to the "include" phrase from "include\_HEADERS" when converting it into a reference to $(includedir). It simply wraps it with "$(-dir)". In fact, you can define your own directory variables and use them with a primary, just like the include target. You may need to do this if you want to install header files into a subdirectory of $(includedir). To install the header files we've mentioned here into the $(includedir)/myapp directory, you could do something like this:

|  |
| --- |
| myappdir = $(includedir)/myapp  myapp\_HEADERS = myapp.h otherstuff.h |

So far, we know how to tell automake where to install our files, but how do we tell it which source files belong with which binary? How do we declare that, for example, main1.c and flubber.c should end up in the myfirstapp executable, and main2.c and dingbat.c are for mysecondapp? The \_PROGRAMS primary sets up the target directory for the executable file. We don't need to install the .c files, so it makes no sense to use a directory target for the \_SOURCES primary. Instead we use the name of the corresponding binary as the target for the \_SOURCES primary (you can also do this with \_HEADERS primaries):

|  |
| --- |
| myfirstapp\_SOURCES = main1.c flubber.c  mysecondapp\_SOURCES = main2.c dingbat.c |

The library primaries work similarly, with a small caveat: Because the period is not a legal character for makefile variables, we cannot have a libdothis.a\_SOURCES variable. We must canonicalize it by converting the questionable punctuation into underscores, producing libdothis\_a\_SOURCES instead. automake expects to see underscores, not hyphens or any other canonicalized character, so don't be creative here. There's only one way to canonicalize an automake variable. The libraries we've already mentioned would look like this:

|  |
| --- |
| libdothis\_a\_SOURCES = dothis.c  libdothat\_a\_SOURCES = dothat.c  libmylibtoollibrary\_la\_SOURCES = dotheotherthing.c |

The \_SOURCES and \_HEADERS primaries provide a means for you to inform automake about the important compilable and distributable source code files in your project. automake supports some less frequently used primaries for other types of files. For miscellaneous architecture-independent data files, you can use the \_DATA primary. This works well for HTML files, as well as special configuration files like the GNOME desktop. By default, \_DATA files are not included in a distribution of your project, so you will have to remind automake to include them by listing them in the EXTRA\_DIST variable.

Other related primaries include \_SCRIPTS for shell scripts, Perl scripts, and the like; \_MANS for man pages; \_TEXINFOS for Texinfo documentation; and a few others. Check the automake documentation for more information on how to use these primaries.

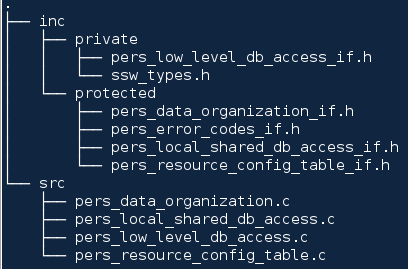
The final category of primaries holds command line options for the linker. The \_LDFLAGS primary lets you define a list of miscellaneous linker flags for a specific target that don't belong with the other library primaries; it's a catchall. Any variables that use the \_LDADD or \_LIBADD primaries should contain only object files, libraries (-l), and library paths (-L) to be linked into a specific ex- ecutable (\_LDADD) or library (\_LIBADD). You cannot mix the contents of any of these three primaries. automake is very particular about what goes into these primaries and will bail out with an error if you misuse them.

Here's what we would do to link our libdothis.a and libdothat.a libraries into our myfirstapp executable:

|  |
| --- |
| myfirstapp\_LDADD = -ldothis -ldothat |

* + 1. ***Makefile.am – pers-common***

Remember the persistence-common structure



Library to install

|  |
| --- |
| lib\_LTLIBRARIES = libperscommon.la |
|  |

Headers to install

If ./configure is run with no overwritten parameters, the headers will be installed/copied in /usr/local/include

|  |
| --- |
| include\_HEADERS= \ |
| inc/protected/pers\_resource\_config\_table\_if.h \ |
| inc/protected/pers\_local\_shared\_db\_access\_if.h \ |
| inc/protected/pers\_data\_organization\_if.h \ |
| inc/protected/pers\_error\_codes\_if.h |
|  |

Compiler flags

Tell the compiler where to take the headers from

$(top\_srcdir) refers to pers-common root directory but it can be omitted (as it is the current directory)

|  |
| --- |
| libperscommon\_la\_CFLAGS = \ |
| -I$(top\_srcdir)/inc/private -I$(top\_srcdir)/inc/protected \ |
| $(ITZAM\_CFLAGS)\ |
| $(PLATFORM\_CFLAGS) |
|  |

Sources to build (relative to the path where Makefile.am resides)

|  |
| --- |
| libperscommon\_la\_SOURCES = \ |
| src/pers\_data\_organization.c\ |
| src/pers\_local\_shared\_db\_access.c\ |
| src/pers\_low\_level\_db\_access.c\ |
| src/pers\_resource\_config\_table.c |
|  |

Remember the GENERIC\_LIBRARY\_VERSION that was defined in configure.ac

This will tell the linker to create a library like for example libperscommon.so.0.1.0, where 0, 1, 0 are the MAJOR, MINOR and MICRO versions specified in configure.ac respectively.

|  |
| --- |
| libperscommon\_la\_LDFLAGS= -version-info $(GENERIC\_LIBRARY\_VERSION) |
|  |

Pkgconfig

|  |
| --- |
| pkgconfigdir = $(libdir)/pkgconfig |
| pkgconfig\_DATA = pkgconfig/libperscommon.pc |
|  |

* 1. ***pkg-config***
     1. ***General information***

In order to understand the last two lines that refer to the pkgconfig the following terms should firstly be explained.

A library can be used if just the headers and the shared library object are provided. However, in an environment where software is usually made available as source packages, authors usually do not provide the compiled binaries for most of their end users. This work is done either by the end user directly, or by packagers of distributions to provide integrated systems build from many independently developed components.

In such an environment, it is very important for your library to integrate well with the platforms it is installed on. If installation paths, features, etc are variable depending on the target platform or build-time settings, there should be a means for an application developer to query configuration information about the library using automated tools. This way, the application developer is able to provide a single source package which will automatically configure itself to build with your library on all supported target platforms.

These days, the best way to make this configuration information available is the [pkg-config](http://pkg-config.freedesktop.org/wiki/) tool. This tool was created to improve upon the old method of manually written shell scripts. It allows you to install details about your library, specifically the linker and include options that should be used with it. Developers can add a line to their configure.ac files that reads this infomation back, along with the options required for your library's dependencies.

##### *The .pc.in file*

Your library should install a .pc file, describing the linker and include options for your library. However, those are dependent on the --prefix given to the configure script, so you will need to create a .pc.in file, which is used to create a .pc file that is updated each time configure is run. For instance:

prefix=@prefix@

exec\_prefix=@exec\_prefix@

libdir=@libdir@

includedir=@includedir@

Name: Something

Description: Some library.

Requires: somethingelse-2.0 somethingmore-1.0

Version: @PACKAGE\_VERSION@

Libs: -L${libdir} -lsomething-1.0

Cflags: -I${includedir}/something-1.0 -I${libdir}/something-1.0/include

You will need to mention this new .pc file that should be created in your configure.ac file, like so:

AC\_CONFIG\_FILES([Makefile something-1.0.pc])

And you will need to mention it in your Makefile.am file, so that it gets installed. For instance:

pkgconfigdir = $(libdir)/pkgconfig

pkgconfig\_DATA = something-1.0.pc

The .pc.in file is automatically marked for distribution by the AC\_CONFIG\_FILES call.

### *CFLAGS and LIBS*

The Makefile needs two pieces of information — how to find the library's header files and how to link to the library itself. These are traditionally stored in variables ending in CFLAGS (for the headers' include argument) and LIBS (for the linker argument). For instance, PROJECT\_CFLAGS and PROJECT\_LIBS. These variables will be set in the configure.ac file.

### *configure.ac*

Your configure.ac script should find the library and set the CFLAGS and LIBS variables:

#### Libraries which have installed a pkg-config .pc file

Most modern libraries use [pkg-config](http://pkg-config.freedesktop.org/wiki/) to provide includes and linker information. In this case, you should use the PKG\_CHECK\_MODULES() macro in your configure.ac file. For instance:

PKG\_CHECK\_MODULES([DEPS], [gtkmm-2.4 >= 2.12.2 somethingelse-1.0 >= 1.0.2])

Of course you must ensure that you have installed [pkg-config](http://pkg-config.freedesktop.org/wiki/).

DEPS\_CFLAGS and DEPS\_LIBS will then include the include and linker options for the library and all of its dependencies, for you to use in your Makefile.am file. The DEPS prefix means 'dependencies', but you can use any prefix that you like. Notice that you can get information about several libraries at once, putting all of the information into one set of \_CFLAGS and \_LIBS variables.

You could also use more than one PKG\_CHECK\_MODULES() line to put information about different sets of libraries in separate \_CFLAGS and \_LIBS variables. However, doing so should be avoided because then pkg-config will not be able to strip redundant flags, nor can it ensure that dependent libraries are specified in the correct order. If you need to build multiple binaries with different sets of dependencies, call PKG\_CHECK\_MODULES() once for each target with the full list of dependencies of that target.

* + 1. ***Integrate pkg support in configure.ac and Makefile.am for pers-common***

*The libperscommon.pc.in file*

|  |
| --- |
| prefix=@prefix@ |
| exec\_prefix=@exec\_prefix@ |
| libdir=@libdir@ |
| includedir=@includedir@ |
| Name: libperscommon |
| Description: library that specifies the organization of Genivi's persistence data and provides access to the resource configuration table and to the local and shared data bases |
| Version: @VERSION@ |
| Cflags: -I${includedir} |
| Libs: -L${libdir} -lperscommon |
| Libs.private: @LIBS@ |
|  |

*Configure.ac*

Generate libperscommon.pc from libperscommon.pc.in inside pkgconfig directory

|  |
| --- |
| **AC\_CONFIG\_FILES**([pkgconfig/libperscommon.pc]) |
|  |

*Makefile.am*

Pkgconfig

pkgconfigdir specifies the final location of the libperscommon.pc (after running make install). The $libdir depends on the parameters given in the ./configure command

pkgconfig\_DATA specifies the file to be copied in pkgconfigdir

|  |
| --- |
| pkgconfigdir = $(libdir)/pkgconfig |
| pkgconfig\_DATA = pkgconfig/libperscommon.pc |
|  |

So having the two input files (configure.ac + Makefile.am) in place, we are finally ready to run the first command to see what happens.

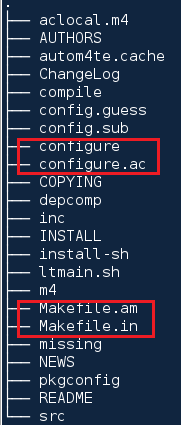
Please proceed to the next paragraph.

* 1. ***autoreconf --force --install***

***autoreconf --force --install***: run this command in the root directory of persistence-common.

If you have already installed the itzam library in your system with the version specified in the configure.ac, then no errors should occur at this step and you can proceed with step 2.6.

Please first check the output of this step against:

****

If however, dependent packages are missing or weren’t properly installed, a suggestive error will be shown.

Please proceed to step 2.5 and follow the instructions to build and install the itzam library in this case, since itzam is the only dependent package for persistence-common library.

* 1. ***Build and install dependencies: itzam***



You can either use the already downloaded version (recommended) or alternatively you could download it from [**http://www.coyotegulch.com/products/itzam/index.html**](http://www.coyotegulch.com/products/itzam/index.html)**.**

Please execute the following commands from the root directory of the uncompressed libitzam-6.0.4 directory:

The special parameters (includedir and libdir) tell the installer to copy the library in /usr/lib and the corresponding headers in /usr/include.

*./configure --includedir=/usr/include --libdir=/usr/lib*

*make*

*make install (you will require root permissions to perform this step)*

Please check the existence of the following files in your system after having performed the steps above:

/usr/include/libitzam/itzam.h

/usr/lib/libitzam-6.0.so.5 -> libitzam-6.0.so.5.1.1

/usr/lib/libitzam-6.0.so.5.1.1

/usr/lib/libitzam.a

/usr/lib/libitzam.la

/usr/lib/libitzam.so -> libitzam-6.0.so.5.1.1

/usr/lib/pkgconfig/libitzam.pc

Manually copy libitzam.pc in /usr/lib/pkgconfig/ if this step has not been performed automatically.

* 1. ***./configure***

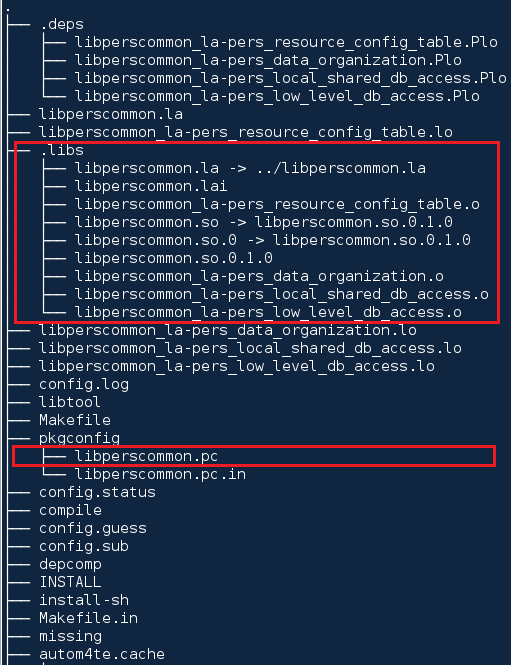
The parameters (libdir and includedir) will tell the installer to copy the library in /usr/lib and the corresponding headers in /usr/include when running make install (step 2.8).

Please run the following command in the root directory of persistence-common:

***./configure --includedir=/usr/include --libdir=/usr/lib***

* 1. ***make***

Please check your output against the following:



* 1. ***make install***

You will need root permissions to perform this step

Please check the existence of the following files after having performed this step:

/usr/lib/libperscommon.so -> libperscommon.so.0.1.0

/usr/lib/libperscommon.so.0 -> libperscommon.so.0.1.0

/usr/lib/libperscommon.so.0.1.0

/usr/include/pers\_data\_organization\_if.h

/usr/include/pers\_error\_codes\_if.h

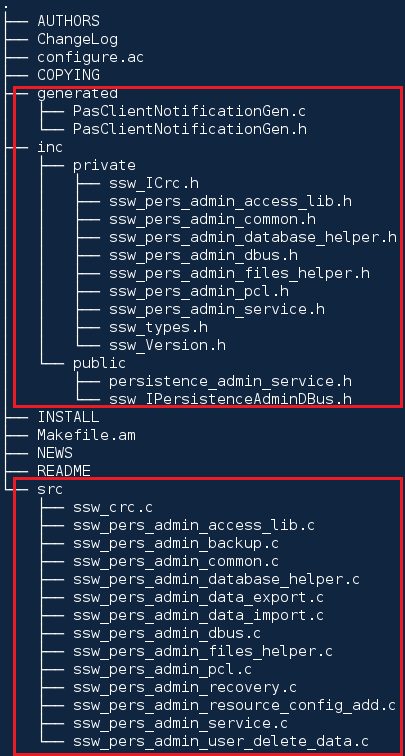
/usr/include/pers\_local\_shared\_db\_access\_if.h

/usr/include/pers\_resource\_config\_table\_if.h

/usr/lib/pkgconfig/libperscommon.pc

1. ***Build persistence-administration-service with autotools***

Initial structure of the persistence-administrator:



What we want to build is a static library named libpersadminaccess from ssw\_pers\_admin\_access\_lib.c and an application named persistenceservice from the other sources provided.

***configure.ac – persistence-administrator***

Initialize autoconf

|  |
| --- |
| AC\_INIT([persistence-administration-service], [1.1.0.0]) |
| AC\_COPYRIGHT([Copyright (c) 2012 Continental Automotive GmbH]) |
| AC\_PREREQ([2.50]) |
|  |

Initialize automake

|  |
| --- |
| AM\_INIT\_AUTOMAKE() |
| AC\_CONFIG\_MACRO\_DIR([m4]) |
| AC\_SUBST([ACLOCAL\_AMFLAGS], ["$ACLOCAL\_FLAGS -I m4"]) |
|  |

Check for basic programs

|  |
| --- |
| AC\_PROG\_CC() |
| AM\_PROG\_CC\_C\_O() |
| AC\_PROG\_INSTALL() |
|  |

Initialize lib tool

|  |
| --- |
| AC\_DISABLE\_STATIC() |
| AC\_PROG\_LIBTOOL() |
|  |

Check for standard header files

|  |
| --- |
| AC\_CHECK\_HEADERS([string.h]) |
|  |

Check for standard functions

|  |
| --- |
| AC\_CHECK\_FUNCS([memcpy]) |
|  |

Check for required packages & overwrite name of itzam library

|  |
| --- |
| PKG\_CHECK\_MODULES([GIO], [gio-2.0 >= 2.30.0]) |
| PKG\_CHECK\_MODULES([GIO\_UNIX], [gio-unix-2.0 >= 2.30.0]) |
| PKG\_CHECK\_MODULES([GLIB], [glib-2.0 >= 2.30.0]) |
| PKG\_CHECK\_MODULES([GOBJECT], [gobject-2.0 >= 2.30.0]) |
| PKG\_CHECK\_MODULES([DBUS], [dbus-1 >= 1.4.10]) |
| PKG\_CHECK\_MODULES([DLT], [automotive-dlt >= 2.2.0]) |
| PKG\_CHECK\_MODULES([ZLIB], [zlib >= 1.2.5]) |
| PKG\_CHECK\_MODULES([ARCHIVELIB], [libarchive >= 3.0.4]) |
| PKG\_CHECK\_MODULES([ITZAM], [libitzam >= 6.0.4]) |
| PKG\_CHECK\_MODULES([PERSCOMMON], [libperscommon >= 1.0.1]) |
| ITZAM\_LIBS="-litzam" |
|  |

Define configure output

|  |
| --- |
| AC\_CONFIG\_FILES([Makefile]) |
| AC\_OUTPUT |
|  |

***Makefile.am – persistence-administrator***

Build static library libpersadminaccess. Please notice the noinst prefix which marks the fact that this library will not be installed/shared with other resources like it was the case for pers-common library.

|  |
| --- |
| noinst\_LTLIBRARIES = libpersadminaccess.la |
|  |

Build application persistenceservice

|  |
| --- |
| bin\_PROGRAMS = $(persistenceservice\_programs) |
|  |

Persistence administration access library compiler flags & sources

|  |
| --- |
| libpersadminaccess\_la\_CFLAGS = \ |
| -I inc/private -I inc/public |
|  |
| libpersadminaccess\_la\_SOURCES = \ |
| src/ssw\_pers\_admin\_access\_lib.c |
|  |

Persistence administration service compiler flags (where headers for this module are to be found) & sources

|  |
| --- |
| persistenceservice\_CFLAGS = \ |
| -I$(top\_srcdir)/inc/private -I$(top\_srcdir)/inc/public -I$(top\_srcdir)/generated \ |
| $(DLT\_CFLAGS) \ |
| $(GIO\_CFLAGS) \ |
| $(GIO\_UNIX\_CFLAGS) \ |
| $(GLIB\_CFLAGS) \ |
| $(GOBJECT\_CFLAGS) \ |
| $(DBUS\_CFLAGS) \ |
| $(ARCHIVELIB\_CFLAGS) \ |
| $(ZLIB\_CFLAGS) \ |
| $(ITZAM\_CFLAGS) \ |
| $(PERSCOMMON\_CFLAGS) |
|  |

Linker flags

|  |
| --- |
| persistenceservice\_LDFLAGS = \ |
| $(PLATFORM\_LDFLAGS) |
|  |

Dependency libraries

|  |
| --- |
| persistenceservice\_LDADD = \ |
| $(DLT\_LIBS) \ |
| $(GIO\_LIBS) \ |
| $(GIO\_UNIX\_LIBS) \ |
| $(GLIB\_LIBS) \ |
| $(DBUS\_LIBS) \ |
| $(GOBJECT\_LIBS) \ |
| $(ARCHIVELIB\_LIBS) \ |
| $(ZLIB\_LIBS) \ |
| $(ITZAM\_LIBS) \ |
| $(PERSCOMMON\_LIBS) |
|  |

Sources to build

|  |
| --- |
| persistenceservice\_SOURCES = \ |
| generated/PasClientNotificationGen.c \ |
| src/ssw\_crc.c \ |
| src/ssw\_pers\_admin\_backup.c \ |
| src/ssw\_pers\_admin\_common.c \ |
| src/ssw\_pers\_admin\_database\_helper.c \ |
| src/ssw\_pers\_admin\_data\_export.c \ |
| src/ssw\_pers\_admin\_data\_import.c \ |
| src/ssw\_pers\_admin\_dbus.c \ |
| src/ssw\_pers\_admin\_files\_helper.c \ |
| src/ssw\_pers\_admin\_pcl.c \ |
| src/ssw\_pers\_admin\_recovery.c \ |
| src/ssw\_pers\_admin\_resource\_config\_add.c \ |
| src/ssw\_pers\_admin\_service.c \ |
| src/ssw\_pers\_admin\_user\_delete\_data.c |
|  |

Output name of the application

|  |
| --- |
| persistenceservice\_programs = persistenceservice |
|  |

* 1. ***Build & install dependencies: automotive-dlt, libarchive, zlib***

*Automotive-dlt [The package was created by Jean-Pierre Bogler]*

**

Installation command:

**rpm -i dlt-daemon-2.5.0-1.i686.rpm**

*Libarchive*

Download latest version of libarchive and install it in /usr/lib with zlib support.

**./configure --includedir=/usr/include --libdir=/usr/lib**

Modify config.h

**#define HAVE\_ZLIB\_H 1**

**make**

**make install**

*Zlib*

Download latest version of zlib and install it in /usr/*lib*.

**./configure --includedir=/usr/include --libdir=/usr/lib**

**make**

**make install**

* 1. ***Build persistence-administrator***

**autoreconf --force --install**

**./configure**

**make**

**[nothing to install]**

1. ***Cross compile for arm-montavista [with help from Mihai Zanoaga]***

*Build all dependencies (pers-common, libarchive, zlib, itzam) for arm-montavista*

**export PATH=$PATH:/PROJ/ODK/20120913\_ODK\_0.8.20/IMX6x/ADK/IMX6q/lnToMV\_Tools/tools/arm-gnueabi/bin**

***./configure --build i686-pc-linux-gnu --host arm-montavista-linux-gnueabi --target arm-montavista-linux-gnueabi --includedir=/PROJ/ADK\_IMX6/freescale-sabrelite-adk/armv7a-mv-linux/usr/include --libdir=/PROJ/ADK\_IMX6/freescale-sabrelite-adk/armv7a-mv-linux/usr/lib/***

-- includedir and libdir are relative to one’s ADK environment.

*--build: the machine you are building on*

*--host: the machine you are building for*

*--target: the machine that GCC will produce code for --target: the machine that GCC will produce code for*

**make**

**make install**

Build persistence-administrator for arm-montavista

Link armv7a-mv-linux to freescale-sabrelite-adk/armv7a-mv-linux.

***ln -s /PROJ/ADK\_IMX6/freescale-sabrelite-adk/armv7a-mv-linux /PROJ/ADK\_IMX6/armv7a-mv-linux***

This is a workaround since all \*.la files (used by the libtool to get compiler & linker flags) in /PROJ/ADK\_IMX6/freescale-sabrelite-adk/armv7a-mv-linux/usr/lib point to includes from /PROJ/ADK\_IMX6/armv7a-mv-linux which does not exist.

For example you can check libmikmod.la and several other libraries:

# Libraries that this one depends upon.

dependency\_libs=' -L/PROJ/ADK\_IMX6/armv7a-mv-linux/usr/lib -ldl'

**export PATH=$PATH:/PROJ/ODK/20120913\_ODK\_0.8.20/IMX6x/ADK/IMX6q/lnToMV\_Tools/tools/arm-gnueabi/bin**

***PKG\_CONFIG\_ALLOW\_SYSTEM\_LIBS=1 PKG\_CONFIG\_SYSROOT\_DIR=/PROJ/ADK\_IMX6/freescale-sabrelite-adk/armv7a-mv-linux PKG\_CONFIG\_LIBDIR=/PROJ/ADK\_IMX6/freescale-sabrelite-adk/armv7a-mv-linux/usr/lib PKG\_CONFIG\_PATH=/PROJ/ADK\_IMX6/freescale-sabrelite-adk/armv7a-mv-linux/usr/lib/pkgconfig ./configure --build=i686-pc-linux-gnu --host=arm-montavista-linux-gnueabi --target=arm-montavista-linux-gnueabi***

***make***

1. ***Final projects: persistence-administrator & persistence-common***



1. ***References and recommended reading***

[**http://openbooks.sourceforge.net/books/wga/generating-makefiles.html**](http://openbooks.sourceforge.net/books/wga/generating-makefiles.html)

[**http://www.openismus.com/documents/linux/automake/automake.shtml**](http://www.openismus.com/documents/linux/automake/automake.shtml)