

# Introducing LXD Cross Documenter

# About LXD

- What does it intend to be?
  - Documentation system for open source projects
- Main idea
  - Mix low and high level information
    - Low level information can be obtained automatically from the source code
    - High level information is only known by human designers or developers

# Motivation

- Documentation status in open source projects
  - There is little documentation for huge projects
  - Available documentation is not always up-to-date
  - Even so, when it is up-to-date, it is not very specific in some cases
- Minimize the learning time for new developers

# Objectives

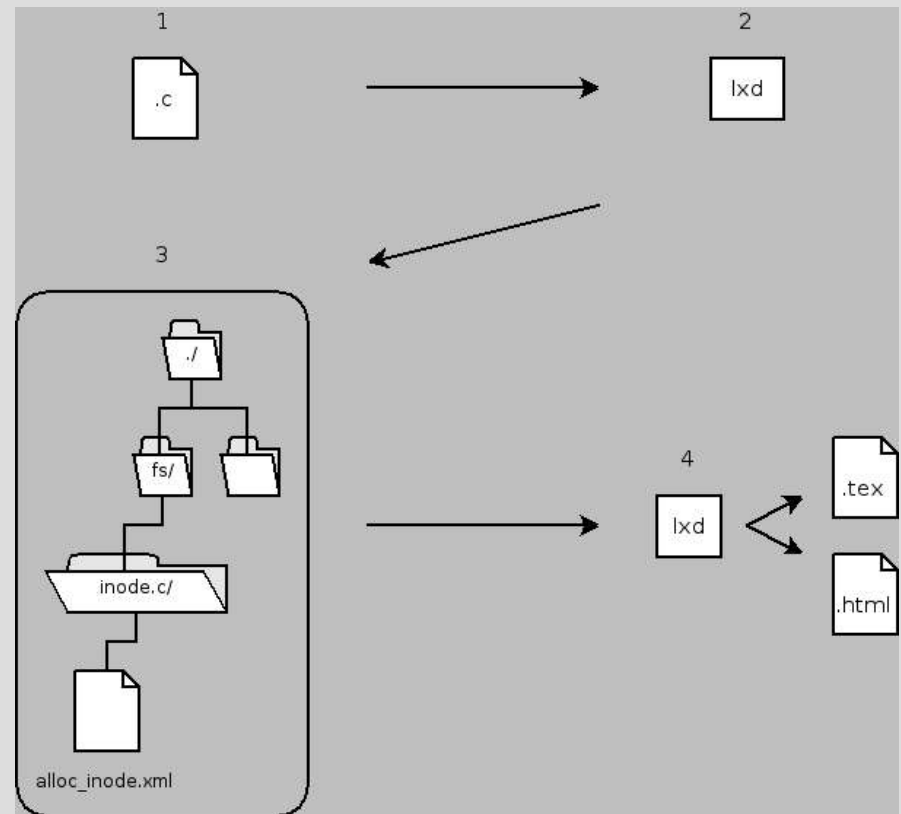
- Formalize the implementation details of open source projects
- Establish the way in which developers write documentation for other developers
- Tie the design and coding phases of open source projects with the documentation task, thus fresh documentation is always available
- To make the most of the distributed work capacity that characterizes open source projects

# High level requirements

- The system shall be usable under a distributed environment
- The system shall allow the user to specify functions, types and variables.
- The system shall allow the user to specify interfaces, data structures, and components
- The system shall allow the user to specify the implementation of interfaces, data structures and components

# How does it work?

1. Write some code
2. Use LXD to create / update the *documentation tree*
3. Fill in the *XML cards*
4. Generate human-readable documentation using LXD



# What is the *documentation tree* about?

- The *documentation tree* is a directory structure
- The *documentation tree* has the same structure as the directory tree of the project code
- Source files in the code tree of the project are directories in the *documentation tree*
- Documentation files are distributed through the *documentation tree* (the *documentation tree* is intended to be a separate copy of the directory structure of the project code to avoid documentation files of being mixed with source code)

# What are the *XML cards* about?

- The *XML cards* are the documentation files that are distributed through the *documentation tree*
- Most of the *XML cards* are the leaves of the *documentation tree*
- It has been identified nine types of *XML cards* with well defined structure:
  - Function, Type, Macro, Variable, Static Value
  - Interface, Data Structure, Component
  - Project

# Examples (I)

## High level *XML card*

<Interface>

<Path "directory1/subdirectory1.1/"\>

<FileName "examplef.xml">

<Description "Specifies the way a set of elements (functions, variables, etc...) should work in order to extend the functionality of some component." \>

<FullFunctionSpecification name="funcSpec1">

    <Responsibility "Function implementing this specification shall take care of ..." \>

    <ReturnValue "void" \>

    <ParameterList>

        <Parameter type="int" description="Number of elements to process" \>

        <Parameter type="\*\*char" description="Elements to process" \>

    <\ParameterList>

<\FullFunctionSpecification>

<\Interface>

# Examples (II)

## Low level *XML card*

<InterfaceImplementationFunction>

<Path "directory1/subdirectory1.1/sourcefileX.c"\>

<FileName "ifImplementationFunction1.xml"\>

<Identifier "ifImplementationFunction1"\>

<Implements "directory1/subdirectory1.1/exampleIf.xml/funcSpec1"\>

<\InterfaceImplementationFunction>

# About the author

- Name: Guillermo López Alejos
- Age: 23 years
- Technical Engineer in Information Management Systems
- Student of Computer Sciences (fifth course, last year)
  - Developing LXD as final year project
- Worked recently in the analysis of *expandfs*
- Worked before in the *FreeCiv* project (development of AI players)

# Some information about LXD internals

- The work of parsing the source code and indexing symbols is already done by LXR
  - LXD is intended to use index files generated by LXR