Modular Builds for ROOT

Final presentation for the Google Summer of Code 2024 project

Pavlo Svirin, 6/11/2024

ROOT: modular builds

- ROOT is a very big software (~730 MB of source code, 5.5 mln lines of code)
  - Around 130 components with inter-dependencies (ROOT_STANDARD_LIBRARY_PACKAGEs)
- ROOT needs lots of time to compile and lots of resources
- User not all of the libraries or utilities which are compiled
  - some components can be omitted by “-D…”
- Practical use case: users might need only a fragment of ROOT’s components (e.g. libCling+libCore or libMathCore)

ROOT: modular builds

- Goal:
  - to allow to select the components to be built
  - to allow to identify the components which are already built and installed to target directory and offer to skip their compilation
  - to easily add new components to existing installation
  - in case of admin-only rights to write into ROOT's installation directory: to install new components together with their modulemap files to different directory and then on ROOT's start combine all of the necessary modulemaps into one
- GSoC'24 proposal: https://docs.google.com/document/d/1UN5jMcm2w83KbnObxsukPdMRzAcnQ0HOKfVPXmizmbE/edit?usp=sharing
- First presentation for this project: https://indico.cern.ch/event/1430315/ (24/06/2024)

ROOT: modular builds

- The idea is to specify which components have to be compiled during configuration time
- We managed to get an absolutely minimal set of components to be compiled to run ROOT:
  - Core, IO, CLING interpreter, MathCore, some basic binaries like root.exe and rootcling
- Other components can be compiled if specified (currently we take into account only top-level like net, graph, etc.)
  - they are specified as “external projects” and can be build separately

Distributed modulemap files

- Modulemap in ROOT is a file which defines available components in the installation directory, their headers and shared libraries
- Currently include/module.modulemap a file of several hundreds lines
- We managed to split it into multiple files:
  - each file defines one component
  - main modulemap file just includes all of these files
- Benefits:
  - easy to add new components
  - easy to identify which components are already installed
- Pull request ready: #16211

$ROOT_BASE/include/ROOT.modulemap :

```
.....
module "XMLParser" {
  requires cplusplus
  module "TDOMParser.h" { header "TDOMParser.h" export * }
  module "TSAXParser.h" { header "TSAXParser.h" export * }
  module "TXMLAttr.h" { header "TXMLAttr.h" export * }
  module "TXMLDocument.h" { header "TXMLDocument.h" export * }
  module "TXMLNode.h" { header "TXMLNode.h" export * }
  module "TXMLParser.h" { header "TXMLParser.h" export * }
  link "libXMLParser.so"
  export *
}

extern module Net “Net.modulemap"
extern module Graf2D “Graf2D.modulemap”
.....
```

$ROOT_BASE/include/ROOT.modulemap.d/Net.modulemap :

```
module "Net" {
  requires cplusplus
  module "NetErrors.h" { header "NetErrors.h" export * }
  module "RRemoteProtocol.h" { header "RRemoteProtocol.h" export * }
   module "TApplicationRemote.h" { header "TApplicationRemote.h" export
* }
  module "TApplicationServer.h" { header "TApplicationServer.h" export * }
  module "TFileStager.h" { header "TFileStager.h" export * }
  module "TFTP.h" { header "TFTP.h" export * }
  module "TGrid.h" { header "TGrid.h" export * }
....
  link "libNet.so"
  export *
}
```

CMake/make example calls

```
>> cmake  -DROOT_ENABLE_PROJECTS="net;graf2d" \
    -DROOT_BASE="/usr"  -DCMAKE_INSTALL_PREFIX=/opt/root3
    \
     -DROOT_SRC_DIR=$HOME/devel/root      .....
>> cmake -DROOT_ENABLE_PROJECT_SET="[All | Essentials]" \
    -DCMAKE_INSTALL_PREFIX=/opt/root3     …
>> make all install
>> make package
```

Component search

- In on of the ROOT_BASE_DIR: already compiled libraries
  - if exists(${ROOT_BASE_DIR}/lib/lib${COMPONENT}.so)
  - then create a pseudo-target for a found library
- In ROOT_SRC_DIR directory as an external project which is to be compiled:
  - enabled as an external project and a dependency of the project which is processed
  - dependencies tree in ROOT source dir can be build using cmake's graphviz feature
  - dependencies in the ROOT's compiled libraries can be tracked using mechanisms in CMake
- Built-ins or external libraries:
  - in progress

Packaging and installation for components

- Installation process copies into destination directory necessary files and puts modulemap include line into ROOT.modulemap file
- Packaging to RPM and installation/uninstallation for external projects works good
- Packaging to RPM and installation for essential part: in progress
- Other formats can be considered

Current status and closest tasks

- Several presentations have been done for the ROOT team regarding the project
- All of the components of the projects implemented
  - https://github.com/pavlo-svirin/root/tree/superbuilds
- A CMake-only solution, around 800 lines of code changed
- Pull requests to ROOT's main repository:
  - distributed modulemaps: #16211
  - superbuilds: #16751
- Current actions in progress:
  - testing
  - study how multiple base directories can be combined in order to create a viable ROOT installation which includes components installed by superuser and by regular users

Closest plans

- Implement relevant test for the ROOT's CI/CD system
- Documentation
- Implement execution of ROOT with distributed base directories
- Gather and analyze other proposals from ROOT users and developers, which are relevant to this project

https://docs.google.com/document/d/1EQ-z5upcEF1sYejKXQlljLD146iVaZom_ZN5NKfDorE/edit?usp=sharing