I'm not a bot

Collection of resources used to develop a computer program "Software library" redirects here and is not to be confused with library software. This article is about a software development concept. For a repository of digital assets, see Digital library. Illustration of an application which uses libvorbisfile to play an Ogg Vorbis file In computing, a library is a collection of resources that can be leveraged during software development to implement a computer program. Commonly, a library consists of executable code such as compiled functions and classes, or a library can be a collection of source code. A resource library may contain data such as images and text. A library can be used by multiple, independent consumers (programs and other libraries). This differs from resources defined in a program which can usually only be used by that program. When a consumer uses a library resource, it gains the value of the library without having to implement it itself. Libraries encourage software reuse in a modular fashion. Libraries can use other libraries resulting in a hierarchy of libraries in a program. When writing code that uses a library, a programmer only needs to know how to use it – not its internal details. For example, a program could use a library that abstracts a complicated system call so that the programmer can use the system feature without spending time to learn the intricacies of the system function. The idea of a computer library dates back to the first computers created by Charles Babbage. An 1888 paper on his Analytical Engine suggested that computer operations could be punched on separate cards from numerical input. If these operation punch cards were saved for reuse then "by degrees the engine would have a library of its own."[1] A woman working next to a filing cabinet containing the subroutine library on reels of punched tape for the EDSAC computer. In 1947 Goldstine and von Neumann speculated that it would be useful to create a "library" of subroutines for their work on the IAS machine, an early computer that was not yet operational at that time.[2] They envisioned a physical library of magnetic wire recordings, with each wire storing reusable computer code.[3] Inspired by von Neumann, Wilkes and his team constructed EDSAC. A filing cabinet of punched tape held the subroutine library for this computer.[4] Programs for EDSAC consisted of a main program and a sequence of subroutines copied from the subroutine library.[5] In 1951 the team published the first textbook on programming, The Preparation of Programs for an Electronic Digital Computer, which detailed the creation and the purpose of the library.[6] COBOL included "primitive capabilities for a library system" in 1959,[7] but Jean Sammet described them as "inadequate library facilities" in retrospect.[8] JOVIAL has a Communication Pool (COMPOOL), roughly a library of header files. Another major contributor to the modern library concept came in the form of the subprogram innovation of FORTRAN. FORTRAN subprograms can be compiled independently of each other, but the compiler lacked a linker. So prior to the introduction of modules in Fortran-90, type checking between FORTRAN[NB 1] subprograms was impossible.[9] By the mid 1960s, copy and macro libraries for assemblers were common. Starting with the popularity of the IBM System/360, libraries containing other types of text elements, e.g., system parameters, also became common. In IBM's OS/360 and its successors this is called a partitioned data set. The first object-oriented programming language, Simula, developed in 1965, supported adding classes to libraries via its compiler.[10][11] The linking (or binding) process resolves references known as symbols (or links) by searching for them in various locations including configured libraries. If a linker (or binder) does not find a symbol, then it fails, but multiple matches may or may not cause failure. Static linking is linking at build time, such that the library executable code is included in the program. Dynamic linking is linking at run time; it involves building the program with information that supports run-time linking to a dynamic link library (DLL). For dynamic linking, a compatible DLL file must be available to the program at run time, but for static linking, the program is standalone. Smart linking is performed by a build tool that excludes unused code in the linking process. For example, a program that only uses integers for arithmetic, or does no arithmetic operations at all, can exclude floating-point library routines. This can lead to smaller program file size and reduced memory usage. Some references in a program or library module are stored in a relative or symbolic form which cannot be resolved until all code and libraries are assigned final static addresses. Relocation is the process of adjusting these references, and is done either by the linker or the loader. In general, relocation cannot be done to individual libraries themselves because the addresses in memory may vary depending on the program using them and other libraries they are combined with. Position-independent code avoids references to absolute addresses and therefore does not require relocation. An executable library consists of code that has been converted from source code into machine code or an intermediate form such as bytecode. A linker allows for using library objects by associating each reference with an address at which the object is located. For example, in C, a library function is invoked via C's normal function call syntax and semantics.[12] A variant is a library containing compiled code (object code in IBM's nomenclature) in a form that cannot be loaded by the OS but that can be read by the linker. A static library is an executable library that is linked into a program at build-time by a linker (or whatever the build tool is called that does linking).[13][14] This process, and the resulting stand-alone file, is known as a static build of the program. A static build may not need any further relocation if virtual memory is used and no address space layout randomization is desired.[15] A static library is sometimes called an archive on Unix-like systems. A dynamic library is linked when the program is run – either at load-time or runtime. The dynamic library was intended after the static library to support additional software deployment flexibility. A source library consists of source code; not compiled code. A shared library is a library that contains executable code designed to be used by multiple computer programs or other libraries at runtime, with only one copy of that code in memory, shared by all programs using the code.[16][17][18] Although generally an obsolete technology today, an object library exposes resources for object-oriented programming (OOP) and a distributed object is a remote object library. Examples include: COM/DCOM, SOM/DSOM, DOE, PDO and various CORBA-based systems. The object library technology was developed since as OOP became popular, it became apparent that OOP runtime binding required information than contemporary libraries did not provide. In addition to the names and entry points of the code located within, due to inheritance, OOP binding also requires a list of dependencies – since the full definition of a method may be in different places. Further, this requires more than listing that one library requires the services of another. In OOP, the libraries themselves may not be known at compile time, and vary from system to system. The remote object technology was developed in parallel to support multi-tier programs with a user interface application running on a personal computer (PC) using services of a mainframe or minicomputer such as data storage and processing. For instance, a program on a PC would send messages to a minicomputer via remote procedure call (RPC) to retrieve relatively small samples from a relatively large dataset. In response, distrubuted object technology was developed. A class library contains classes that can be used to create objects. In Java, for example, classes are contained in JAR files and objects are created at runtime from the classes. However, in Smalltalk, a class library is the starting point for a system image that includes the entire state of the environment, classes and all instantiated objects. Most class libraries are stored in a package repository (such as Maven Central for Java). Client code explicitly specifies dependencies to external libraries in build configuration files (such as a Maven Pom in Java). A remote library runs on another computer and its assets are accessed via remote procedure call (RPC) over a network. This distributed architecture allows for minimizing installation of the library and support for it on each consuming system and ensuring consistent versioning. A significant downside is that each library call entails significantly more overhead than for a local library. A runtime library provides access to the runtime environment that is available to a program – tailored to the host platform. Many modern programming languages specify a standard library that provides a base level of functionality for the language environment. A code generation library has a high-level API generating or transforming byte code for Java. They are used by aspect-oriented programming, some data access frameworks, and for testing to generate dynamic proxy objects. They also are used to intercept field access.[19] On most modern Unix-like systems, library files are stored in directories such as /lib, /usr/lib and /usr/local/lib. A filename typically starts with lib, and ends with .a for a static library (archive) or .so for a shared object (dynamically linked library). For example, libfoo.a and libfoo.so. Often, symbolic link files are used to manage versioning of a library by providing a link file named without a version that links to a file named with a version. For example, libfoo.so.2 might be version 2 of library foo and a link file named libfoo.so provides a version independent name to that file that programs link to. The link file could be changed to a refer to a version 3 (libfoo.so.3) such that consuming programs will then use version 3 without having to change the program. Files with extension .la are libtool archives; not usable by the system. See also: Dynamic linker § macOS and iOS The macOS system inherits static library conventions from BSD, with the library stored in a .a file. It uses either .so or .dylib for dynamic libraries. Most libraries in macOS, however, consist of "frameworks", placed inside special directories called "bundles" which wrap the library's required files and metadata. For example, a framework called Abc would be implemented in a bundle called Abc.framework, with Abc.framework/Abc being either the dynamically linked library file or a symlink to the dynamically linked library file in Abc.framework/Versions/Current/Abc. Often, a Windows dynamic-link library (DLL) has the file extension .dll,[20] although sometimes different extensions are used to indicate general content, e.g. .ocx for a OLE library. A .lib file can be either a static library or contain the information needed to build an application that consumes the associated DLL. In the latter case, the associated DLL file must be present at runtime. Code reuse – Using existing code in new software Object file – File containing relocatable format machine code Plug-in – Software component that extends the functionality of existing software Prelink, also known as prebinding Runtime library – Access to a program's runtime environment Visual Component Library – Object Pascal framework for Windows (VCL) Component Library for Cross Platform (CLX) C standard library – Standard library for the C programming language Java Class Library – Core Java libraries Framework Class Library – Standard library of Microsoft's .NET Framework Generic programming – Style of computer programming (used by the C++ Standard Library) soname – Field of data in a shared object file Method stub – Short and simple version of a method List of open source code libraries ^ It was possible earlier between, e.g., Ada subprograms. ^ Babbage, H. P. (1888-09-12). "The Analytical Engine". Proceedings of the British Association. Bath. ^ Goldstine, Herman H. (2008-12-31). The Computer from Pascal to von Neumann. Princeton: Princeton University Press. doi:10.1515/9781400820139. ISBN 978-1-4008-2013-9. ^ Goldstine, Herman; von Neumann, John (1947). Planning and coding of problems for an electronic computing instrument (Report). Institute for Advanced Study. pp. 3, 21–22. OCLC 26239859. it will probably be very important to develop an extensive "library" of subroutines ^ Wilkes, M. V. (1951). "The EDSAC Computer". 1951 International Workshop on Managing Requirements Knowledge. 1951 International Workshop on Managing Requirements Knowledge. IEEE. p. 79. doi:10.1109/afips.1951.13. ^ Campbell-Kelly, Martin (September 2011). "In Praise of 'Wilkes, Wheeler, and Gill'". Communications of the ACM. 54 (9): 25–27. doi:10.1145/1995376.1995386. S2CID 20261972. ^ Wilkes, Maurice; Wheeler, David; Gill, Stanley (1951). The Preparation of Programs for an Electronic Digital Computer. Addison-Wesley. pp. 45, 80–91, 100. OCLC 641145988. ^ Wexelblat, Richard (1981). History of Programming Languages. ACM Monograph Series. New York, NY: Academic Press (A subsidiary of Harcourt Brace). p. 274. ISBN 0-12-745040-8. ^ Wexelblat, op. cit., p. 258 ^ Wilson, Leslie B.; Clark, Robert G. (1988). Comparative Programming Languages. Wokingham, England: Addison-Wesley. p. 126. ISBN 0-201-18483-4. ^ Wilson and Clark, op. cit., p. 52 ^ Wexelblat, op. cit., p. 716 ^ Deshpande, Prasad (2013). Metamorphic Detection Using Function Call Graph Analysis (Thesis). San Jose State University Library. doi:10.31979/etd.t9xm-ahsc. ^ "Static Libraries". TLDP. Archived from the original on 2013-07-03. Retrieved 2013-10-03. ^ Kaminsky, Dan (2008). "Chapter 3 - Portable Executable and Executable and Linking Formats". Reverse Engineering Code with IDA Pro. Elsevier. pp. 37–66. doi:10.1016/b978-1-59749-237-9.00003-x. ISBN 978-1-59749-237-9. Retrieved 2021-05-27. ^ Collberg, Christian; Hartman, John H.; Babu, Sridivya; Udupa, Sharath K. (2003). SLINKY: Static Linking Reloaded. USENIX '05. Department of Computer Science, University of Arizona. Archived from the original on 2016-03-23. Retrieved 2016-03-17. ^ Levine, John R. (2000). "9. Shared Libraries". Linkers and Loaders. ISBN 1-55860-496-0. ^ UNIX System V/386 Release 3.2 Programmers Guide, Vol. 1 (PDF). Prentice Hall. 1989. p. 8-2. ISBN 0-13-944877-2. ^ "Shared Libraries in SunOS" (PDF). pp. 1, 3. ^ "Code Generation Library". Source Forge. Archived from the original on 2010-01-12. Retrieved 2010-03-03. Byte Code Generation Library is high level API to generate and transform JAVA byte code. It is used by AOP, testing, data access frameworks to generate dynamic proxy objects and intercept field access. ^ Bresnahan, Christine; Blum, Richard (2015-04-27). LPIC-1 Linux Professional Institute Certification Study Guide: Exam 101-400 and Exam 102-400. John Wiley & Sons (published 2015). p. 82. ISBN 9781119021186. Archived from the original on 2015-09-24. Retrieved 2015-09-03. Linux shared libraries are similar to the dynamic link libraries (DLLs) of Windows. Windows DLLs are usually identified by .dll filename extensions. Levine, John R. (2000) [October 1999]. "Chapter 9: Shared Libraries & Chapter 10: Dynamic Linking and Loading". Linkers and Loaders. The Morgan Kaufmann Series in Software Engineering and Programming (1 ed.). San Francisco, USA: Morgan Kaufmann. ISBN 1-55860-496-0. OCLC 42413382. Archived from the original on 2012-12-05. Retrieved 2020-01-12. Code: [1][2][dead link] Errata: [3] Article Beginner's Guide to Linkers by David Drysdale Article Faster C++ program startups by improving runtime linking efficiency by Léon Bottou and John Ryland How to Create Program Libraries[usurped] by Baris Simsek BFD - the Binary File Descriptor Library 1st Library-Centric Software Design Workshop LCSD'05 Archived 2019-08-28 at the Wayback Machine at OOPSLA'05 2nd Library-Centric Software Design Workshop LCSD'06 at OOPSLA'06 How to create shared library by Ulrich Drepper (with much background info) Anatomy of Linux dynamic libraries at IBM.com Retrieved from " In this article i will illustrate how to generate pdf files from html in PHP using the open source TCPDF library. There are a lot of open source libraries in php that can be used to generate pdf files from html, but most of them lack some pdf features such as how to manipulate text written in Right to Left languages, also handling the different font types. TCPDF in the other hand is a powerful open source library that has almost all the features needed to generate a well working pdf files, from these features: all standard page formats, custom page formats, custom margins and units of measure. UTF-8 Unicode and Right-To-Left languages TrueTypeUnicode, OpenTypeUnicode, TrueType, OpenType, Type1 and CID-0 fonts. methods to publish some XHTML + CSS code, Javascript and Forms. images (PNG, JPG, GIF), graphic (geometric figures) and transformation methods. automatic page header and footer management. document encryption up to 256 bit and digital signature certifications. Check out TCPDF official website to figure the rest of the available features. Download & Usage Go to the github repository of TCPDF and clone or download it. You need to add the library in your project folder and then include it like this: include_once('TCPDF/tcpdf.php'); By including the main tcpdf.php you have access to all classes and methods. Configuration The main configuration file located in TCPDF/config/tcpdf_config.php. This file contains the general configuration that applies as per document basis when generating pdf files. Try to uncomment and change the configuration values to suite your needs. Adjust these configurations values: // Installation path (/var/www/html/project folder/TCPDF/) define ('K_PATH_MAIN', '/var/www/html/pdf-tutorial/TCPDF/'); // URL path to tcpdf installation folder (. define ('K_PATH_URL', '); // Path for PDF fonts. define ('K_PATH_FONTS', K_PATH_MAIN.'fonts/'); // Default images directory for example used for logos define ('K_PATH_IMAGES', K_PATH_MAIN.'../images/'); // Default image logo used be the default Header() method. define ('PDF_HEADER_LOGO', 'my_logo.png'); // Header logo image width in user units. define ('PDF_HEADER_LOGO_WIDTH', 30); // Cache directory for temporary files (full path). define ('K_PATH_CACHE', '/tmp/'); The above configurations is the basic configurations needed but there more you can explore them by yourself as we don't cover them in this tutorial. As shown i have created a new folder for images in the project root folder, this will be used to store the header logo: define ('K_PATH_IMAGES', K_PATH_MAIN.'../images/'); define ('PDF_HEADER_LOGO', 'my_logo.png'); Example: Let's create a new file in the project root directory called index.php with the following contents: index.php include_once('TCPDF/tcpdf.php'); // create new PDF document $pdf = new TCPDF(PDF_PAGE_ORIENTATION, PDF_UNIT, PDF_PAGE_FORMAT, true, 'UTF-8', false); $pdf->SetCreator("Wael Salah"); $pdf->SetAuthor('Wael Salah'); $pdf->SetTitle('Demonstrating pdf with php'); $pdf->setHeaderData(PDF_HEADER_LOGO, PDF_HEADER_LOGO_WIDTH, PDF_HEADER_TITLE, PDF_HEADER_STRING, array(0, 6, 255), array(0, 64, 128)); $pdf->setFooterData(array(0,64,0), array(0,64,128)); $pdf->setHeaderFont(array(PDF_FONT_NAME_MAIN, '', PDF_FONT_SIZE_MAIN)); $pdf->setFooterFont(Array(PDF_FONT_NAME_DATA, '', PDF_FONT_SIZE_DATA)); // set default monospaced font $pdf->SetDefaultMonospacedFont(PDF_FONT_MONOSPACED); $pdf->SetMargins(PDF_MARGIN_LEFT, PDF_MARGIN_TOP, PDF_MARGIN_RIGHT); $pdf->SetHeaderMargin(PDF_MARGIN_HEADER); $pdf->SetFooterMargin(PDF_MARGIN_FOOTER); // set auto page breaks $pdf->SetAutoPageBreak(TRUE, PDF_MARGIN_BOTTOM); // set image scale factor $pdf->setImageScale(PDF_IMAGE_SCALE_RATIO); // ---------- // set default font subsetting mode $pdf->setFont('dejavusans', '', 14, '', true); $pdf->AddPage(); $html = SetCreator("Wael Salah"); $pdf->SetAuthor('Wael Salah'); $pdf->SetTitle('Demonstrating pdf with php'); Then i set the header and footer data: $pdf->setHeaderData(PDF_HEADER_LOGO, PDF_HEADER_LOGO_WIDTH, PDF_HEADER_TITLE, PDF_HEADER_STRING, array(0, 6, 255), array(0, 64, 128)); $pdf->setFooterData(array(0,64,0), array(0,64,128)); The setHeaderData() method accepts the header logo, the logo width, the header title, the header string text, all of these constants defined in the configuration file we described above. The other two arguments represents the RGB color for the text and line. After that i set the header and footer fonts as shown: $pdf->setHeaderFont(array(PDF_FONT_NAME_MAIN, '', PDF_FONT_SIZE_MAIN)); $pdf->setFooterFont(Array(PDF_FONT_NAME_DATA, '', PDF_FONT_SIZE_DATA)); Those two methods accepts an array with the font family and font size. It's important also to set margins for the document, this achieved by calling those methods: $pdf->SetMargins(PDF_MARGIN_LEFT, PDF_MARGIN_TOP, PDF_MARGIN_RIGHT); $pdf->SetHeaderMargin(PDF_MARGIN_HEADER); $pdf->SetFooterMargin(PDF_MARGIN_FOOTER); Then i enabled the option of auto page break by calling SetAutoPageBreak(). Also for the images i set the scale ratio to be 1.25 by calling setImageScale(). Finally i set the font for the document text by calling setFont() method which takes the font family, style, size, and the font file. The second step i added a new page like this: $pdf->AddPage(); When ever you need to add a new page you must call this method. Next i specified the html text to be written to pdf file then we called writeHtml() method to write the html to the pdf page: $html = writeHTML($html): $pdf->AddPage(); $html =