Moto Extention Compiler Upgrade Design

Design Goals

1) mxc should be installed and usable from the install directory in a fairly simple way
*** 2) we should know how to find all the includes, archive files an extension depends on from the extension itself**
3) moto extension building should no longer depend on make.vars
*** 4) mmc compilation of modules that depend on extensions should not need any information not derivable from the extension itself**

Design Questions

Q Why does mmc currently need the MySQL library information ?
A Its complicated. The reason we have .a files at all is because gcc when linking against shared objects will not link statically. Objects in .a files however are not linked to dependent 3rd party libraries whereas objects in .so files are, this is because the .a/.o files they are linked against may not be in LDPATH.

Design Implications

mxc should be installed and usable from the install directory

For some reason mxc doesn't work properly from the install directory. We need to find out why.

1) Part of the reason it hasn't needed to is because mxc gets executed prior to being installed since it gets run as part of the install process. What this really means is that it *must* be runnable from the src dir. However it *should* also be runnable from the install directory.

We should know how to find all the includes, archive files an extension depends on from the extension itself

The overarching process for any compilation of moto pages should occur roughly as follows

1) Generate C code for the set of input pages by calling 'moto -c' on the individual input pages
2) Discover the moto extensions used by these pages by calling moto -u on each page and unioning the output.
3) For each moto extension the information needed at compile time must be discovered. This will be done by calling mxdeps
    3.1) call mxdeps -I to get retrieve the include paths necessary for compilation
    3.2) call mxdeps -a to retrieve the archives necessary for compilation
4) In addition to these include paths and archives we also need to include the extension archive and and extension folder as an include path. When generating the compilation string the extension archive should appear prior to dependant archives.

For mmc style compilation to work properly with extensions that depend on external libraries and headers the .a files and necessary includes and include paths must be listed in the relevant .i files. Listing them in make.vars alone is not good enough because mmc and any future compiler scripts do not have access to the necessary information

This means

1) We need new mxc keywords
    - `IncludePath` : The path to a directory containing header files needed for compilation of moto code that uses the extension
    - `LibraryPath` : The path to a directory containing external libraries needed to compile the extension
    - `Archive` : The full path to a .a file needed at compilation time

2) We need new mxc generated functions (generated per extensions)
    - `__MOTO_<entention>_get_includePaths` - This function should return the specified external include paths
    - `__MOTO_<entention>_get_libraryPaths` - This function should return the specified external library paths
    - `__MOTO_<entention>_get_archives` - This function should return the specified external archive paths

The moto binary is already used to the get the relevant .so dependencies . It could possibly be used to generate -I and -L / -l dependencies directly

We must keep in mind that order sometimes matters with regard to archive inclusion in the gcc command. The basic ordering we want is

1) extesnion .a files followed
2) libmotom.a
3) libcodex.a
4) external archives

Right now the tools **mototest** and **mmc** need to generate compiled binaries from moto code. They currently have at least some relevant library information hard coded into them which is a bad thing

In the future we may want to consider:

1) a moto runtime library
2) generating cgis / fast cgis
3) apache 2 modules

Moto extension building should no longer depend on make.vars

Now that the make vars formerly present in make.vars are all replicated in .i files

EXINCL -> IncludePath
EXLDPATH -> LibraryPath
EXLIBS -> Archive or Library

The generated mx.mk file can be modified to take advantage of substitutions performed when mxc runs

mxc_emitBuildScript in mxcg.c can be modified to substitute in LIBRARYPATHS INCLUDEPATHS and ARCHIVES

What do we do in cases where no .a file for a library a moto extension depends on exists Or in cases where the library itself rely's on symbols in other shared

libraries ?

It seems that when there is a reliance on shared libraries that cannot be worked around those libraries MUST be in the LD_PATH of the apache user. APXS will not copy in symbols from shared libraries when a module is being built.

Need to add the following new tokens

%token ARCHIVE
%token INCLUDEPATH
%token LIBRARYPATH

Need to add / modify the following parser rules

```
extension_header_qualifier
      :     extension_file_qualifier { $$ = $1; }
      |     extension_includepath_qualifier { $$ = $1; }
      |     extension_librarypath_qualifier { $$ = $1; }
      |     extension_archive_qualifier { $$ = $1; }
      |     extension_include_qualifier { $$ = $1; }
      |     extension_library_qualifier { $$ = $1; }
      ;

extension_archive_qualifier
      :  ARCHIVE  local_file_specification
            { $$ = op(ARCHIVE, 1, $2); }
      ;

extension_includepath_qualifier
      :  INCLUDEPATH  local_file_specification
            { $$ = op(INCLUDEPATH, 1, $2); }
      ;

extension_librarypath_qualifier
      :  LIBRARYPATH  local_file_specification
            { $$ = op(LIBRARYPATH, 1, $2); }
      ;
```

Need to add the following new tokens to the lexer

```
LIBRARYPATH    LibraryPath:
ARCHIVE        Archive:
INCLUDEPATH    IncludePath:
```

And the following new nonterminals to the parser

```
extension_archive_qualifier
extension_includepath_qualifier
extension_librarypath_qualifier
```