phpGroupWare Setup III June 18, 2001 MilesLott A developer introduction to using the next generation setup application for phpgroupware. Introduction Welcome Thanks for taking the time to look over this document. If you are a developer who is new to phpgroupware, this document will be invaluable to your success during the life of your application. This is in addition to the other fine documentation available in the phpgwapi/doc directory in your install. Even long-time phpgw developers should benefit this document. Herein, I will attempt to outline the critical steps required in order to get along with setup3, setup-TNG, or whatever we end up calling it (Hey, how about 'setup'?) Overview With setup3, we introduce several new capabilities and technologies for the developer and end user alike. Michael Dean was kind enough to offer up schema_proc to form the core of an abstracted and database-independent upgrade process. This enables developers to write a single set of upgrades and table definitions, which should then work on MySQL and PostgreSQL, or any other database type we might add in the future. Adding to this to control the process was a good chunk of the old setup program, written by Dan Kuykendall (Seek3r). Dan had everything to do with the new dependencies support and with the format of the $setup_info array in setup3. Setup3 adds multi-language support for the setup application, a long missed feature, I would imagine. Setup3 gives each application developer control over their application install and upgrade processes, while giving them access to work within a realm formerly owned by only the former core phpgroupware applications. Yes, this is extra work for the developer. But it is hoped that setup3 is also viewed as a tool that can truly enhance the development process. OK. Let's dive right in... Application setup files The files in this section are contained within each application/setup directory. Every app will some of these files in order to operate with setup3. setup.inc.php (Required) Basic information The values in this section must be used by all applications. The first section of setup.inc.php defines the very basic and yet critical information about the application. Take a look at the following section: $setup_info['addressbook']['name'] = 'addressbook'; $setup_info['addressbook']['title'] = 'Addressbook'; $setup_info['addressbook']['version'] = '0.9.13.002'; $setup_info['addressbook']['app_order'] = 4; $setup_info['addressbook']['enable'] = 1; 'name' is used throughout phpgroupware, typically in $phpgw_info flags such as 'currentapp' or as the 'app_name' almost everywhere else. 'title' would be used in the navbar, admin, preferences, as well as in the application itself. The 'version' string defines the version of the application and table code. This would be incremented whenever you create a new upgrade function, and typically only for table modifications. If the change is significant from the last code update, you could increment this here also. Incrementing this version string is not trivial, so please do read the rest of this document for more information about that. 'app_order' determines the order of applications in the navbar. If the number you set here is the same as is set for another app, the app whose 'name' is first in the English alphabet would appear first. Smaller numbers show closer to the top or left end of the navbar, depending upon the layout. The 'enable' string is used by the phpgroupware API to determine whether an application is disabled, enabled, or enabled but hidden from the navbar. Most applications will want this set to a value of 1 (enabled). The notifywindow app sets this to 2, which keeps it off the navbar. An enable of 0 would disable the app by default. There is one other special case, 3, which is used primarily by the API itself. From the perspective of setup3, the API is an application just like any other application. By setting the 'enable' flag to 3, the API is still enabled, but will not be assignable to a user as a real application. It will thereby be hidden from the admin for application and user/group editing. Table info Only applications with database tables will use entries in this section. The next section of $setup_info values is an array defining all of the application's database tables: $setup_info['addressbook']['tables'] = array('phpgw_addressbook', 'phpgw_addressbook_extra'); This is a simple array, and must list accurately the current table names you are using in your application. This list will match a much more complex array of table specifications, as you will see below. Hooks Some applications will use this section. The hooks array part of $setup_info contains a simple list of hooks the application will use: $setup_info['addressbook']['hooks'][] = 'preferences'; $setup_info['addressbook']['hooks'][] = 'admin'; Here we also note a different method of 'stuffing the array.' In any case, this list of hooks will be required soon in order for your hook_admin.inc.php and other files to work. This is being done to cut down on the manual directory listing and file_exists loops done currently to discover hook files. Other than 'preferences' and 'admin', 'home', 'manual', 'after_navbar' and 'navbar_end' are all valid hook entries. Dependencies All applications will have at least one entry here. The final section, or array of data, is a listing of the other applications your application requires in order to function: $setup_info['addressbook']['depends'][] = array('appname' => 'phpgwapi', 'versions' => Array('0.9.10', '0.9.11', '0.9.12', '0.9.13')); This is the standard dependency array for all phpgroupware applications. It states that this application requires the phpgwapi, and lists the versions with which versions this app is compatible. This list would need to be appended upon each new API release, assuming your application is compatible with this new API version. You may list other applications here, e.g. your app might depend upon 'email' in order to work properly. Do NOT list applications here without considering this: If you do list an application here, and your app does not really require it, your application will not install unless that other application is already installed. This is handled normally within the install/upgrade process loops, which will install only applications whose dependencies are satisfied. Using a multipass function, the applications are installed in the correct order to ensure that dependencies are resolved. In all cases, the API would be installed first in every new install or upgrade, since all applications depend on the API. tables_baseline.inc.php (Recommended) Any application that has at least one upgrade routine will have this file. The tables_baseline file represents the earliest supported version of an application's tables. This file is used only in the upgrade process, and is critical to its success. It contains an array of database-independent table, field, key and index definitions. This array is formatted for use by the class.schema_proc_array.inc.php file in setup3. See the tables_update section below for more detail about schema_proc, but for now, here is a simple table definition in this format: $phpgw_baseline = array('skel' => array('fd' => array('skel_id' => array('type' => 'auto','nullable' => false), 'skel_owner' => array('type' => 'varchar','precision' => 25), 'skel_access' => array('type' => 'varchar','precision' => 10), 'skel_cat' => array('type' => 'int','precision' => 4), 'skel_des' => array('type' => 'text'), 'skel_pri' => array('type' => 'int','precision' => 4)), 'pk' => array('skel_id'), 'fk' => array(), 'ix' => array(), 'uc' => array())); This multi-dimensional array contains 1 subarray with 5 subs of its own. The first array ('skel' above) defines the table name. Below that are 5 sections, 'fd' for field definitions, 'pk' to define primary keys, 'fk' to define foreign keys, 'ix' to define indexed fields, and 'uc' to define columns that require unique values. In the above example, the table 'skel' has 6 fields (skel_id, skel_owner, skel_access, skel_cat, skel_des, skel_pri), and 'skel_id' is defined also as the primary key for this table. More information on this array is below. But, this format was chosen as an available solution for defining tables and fields without having to maintain seperate files for different databases. tables_current.inc.php (Recommended) All applications with tables will need this file. The tables_current file defines the current table definition that matches the 'version' string in $setup_info as well as the current code. This file is used only for new installs, or whenever the application is removed and reinstalled. The format and name of the array in this file is the same as for the tables_baseline file listed above. In fact, whenever it is required to change your table definitions, you would start by copying the current file over to become the tables_baseline file. After having created your upgrade routines, you would then recreate the current file to match the new table definitions. tables_update.inc.php (Recommended) Any application which requires an upgrade to a previous version's tables will need this file. This file will be the most complex of all setup-oriented files with which you will be working. It will contain all upgrade functions capable of upgrading any possible version of your phpgroupware app. These upgrade routines roughly match the old setup program's upgrade functions, but the use of objects and the methods have changed dramatically. The simplest version upgrade routine would look like: $test[] = "0.9.3pre10"; function addressbook_upgrade0_9_3pre10() { global $setup_info; $setup_info['addressbook']['currentver'] = '0.9.3'; return $setup_info['addressbook']['currentver']; } This upgrade function merely updates the current version number. Note that there is not only an upgrade function, but also the setting of a value in the $test array. The name 'test' is a holdover from the old setup program, and is an arbitrary choice. However, this name must be used for the upgrade process to work. Prior to each of your upgrade functions, add the value of the previous version to $test. Now look at the function name. The name is important and should be structured as the application name and the version from which you are intending to upgrade. The '.'s in the version string are replaced with '_'. Inside the function, we global the $setup_info array. Next, we alter the version number in that array, for our application. Please be careful to specify YOUR application name here. The very last thing we do is to return this new version to the calling function. The upgrade process relies on the value returned, since it uses this directly to determine the new version. This may appear illogical on some level, but it does work. The reason for returning this value instead of a True or 1, etc. has to do with variable scope and lifetime. In this way, even the globaling of $setup_info inside the function may have little effect on the upgrade process. But, there may be values in this array you would want to use within the function. More on that later. There is one other variable you would need if doing any database operations here. If you global $phpgw_setup, you will then have access to db and schema_proc objects and functions. The objects of interest here are: $phpgw_setup->oProc $phpgw_setup->db. For most database work you should use the oProc object. This also has a db object that should be used for most standard phpgw API db class functions, including $db->query, next_record, num_rows, and f. The use of these for standard db operations is critical to the upgrade process. Schema_proc has a flag that can be set to determine what mode of upgrade we are in. This flag is set in the setup class during the upgrade process, and should not be altered locally. This flag is a decision on whether to alter the database or the schema_proc array. The tables_baseline file above is loaded by setup prior to running your upgrade routines. If the current installed version is greater than the current upgrade routine, we don't need to alter the database yet. But schema_proc instead alters the $phpgw_baseline array in memory. The maintenance of this array is done even when we do alter the database. Once our version number in the test array matches the currently installed version of an application, real work on the tables begins. 'Why bother modifying this array at all', you may ask. The array must be maintained in order to keep current table definition status. This is used in some schema_proc functions when altering columns and tables. This is especially critical for pgsql schema_proc functions. By using the $phpgw_setup->oProc object for basic inserts and queries, we acheive the ability to run all upgrade functions in every upgrade cycle without actually altering the database until we reach the current version we actually want to upgrade. For example: $sql = "SELECT * FROM phpgw_addressbook_extra WHERE contact_name='notes'"; $phpgw_setup->oProc->query($sql,__LINE__,__FILE__); while($phpgw_setup->oProc->next_record()) { We could have used $phpgw_setup->db or even a copy for the above activity. However, using the above method ensures that an array only upgrade does just that. If the flag was set in setup telling schema_proc to alter the array only, we do not want to touch the tables for inserts or selects yet. In this case, $phpgw_setup->oProc->next_record() returns False, and the loop is skipped. The $phpgw_baseline array does not know about table content, only table and field definitions. If the upgrade function containing this method is actually working on the tables (currentver <= the upgrade function), then next_record() is returned as the expected action of pulling the next row of data. Inside of this while loop, you can safely use $phpgw_setup->db, or preferably a copy, to do the insert/delete, etc you want to have happen here. $cid = $phpgw_setup->oProc->f('contact_id'); $cvalu = $phpgw_setup->oProc->f('contact_value'); $update = "UPDATE phpgw_addressbook set note='" . $cvalu . "' WHERE id=" . $cid; $db1->query($update); $delete = "DELETE FROM phpgw_addressbook_extra WHERE contact_id=" . $cid . " AND contact_name='notes'"; $db1->query($delete); } $db1 is a copy of $phpgw_setup->db, to avoid potential conflicts with the rest of setup's db activities. In addition to the basic API db class functions, schema_proc introduces the following special functions: function DropTable($sTableName) function DropColumn($sTableName, $aTableDef, $sColumnName) function RenameTable($sOldTableName, $sNewTableName) function RenameColumn($sTableName, $sOldColumnName, $sNewColumnName) function AlterColumn($sTableName, $sColumnName, $aColumnDef) function AddColumn($sTableName, $sColumnName, $aColumnDef) function CreateTable($sTableName, $aTableDef) Please use these functions where appropriate in place of standard SQL CREATE, DROP, and ALTER TABLE commands. This will ensure that your upgrade script works for all supported databases. Of these functions, DropTable, RenameTable, and RenameColumn are pretty straightforward. Pass these the table names you wish to Drop/Rename, and schema_proc will handle the rest, including indexes and sequences, where applicable. The remaining functions require some explanation: CreateTable: $phpgw_setup->oProc->CreateTable('categories', array('fd' => array('cat_id' => array('type' => 'auto','nullable' => false), 'account_id' => array('type' => 'int','precision' => 4,'nullable' => false, 'default' => 0), 'app_name' => array('type' => 'varchar','precision' => 25,'nullable' => false), 'cat_name' => array('type' => 'varchar', 'precision' => 150, 'nullable' => false), 'cat_description' => array('type' => 'text', 'nullable' => false)), 'pk' => array('cat_id'), 'ix' => array(), 'fk' => array(), 'uc' => array())); Does this look familiar? The array passed to CreateTable is in the format used also in tables_baseline and tables_current. Note a slight difference where the table name is being passed as a seperate argument. The second argument to the function is the table definition array, starting with 'fd'. AddColumn: $phpgw_setup->oProc->AddColumn('phpgw_categories','cat_access',array('type' => 'varchar', 'precision' => 25)); Here we pass the table name of an existing table, the new column name, and a field definition. This definition is merely a slice of the table arrays found earlier in this document. AlterColumn: $phpgw_setup->oProc->AlterColumn('phpgw_sessions','session_action',array('type' => 'varchar', 'precision' => '255')); The format of this function matches AddColumn. It is also a simple case of passing the table name, field name, and field definition. DropColumn: $newtbldef = array("fd" => array('acl_appname' => array('type' => 'varchar', 'precision' => 50), 'acl_location' => array('type' => 'varchar', 'precision' => 255), 'acl_account' => array('type' => 'int', 'precision' => 4), 'acl_rights' => array('type' => 'int', 'precision' => 4)), 'pk' => array(), 'ix' => array(), 'fk' => array(), 'uc' => array()); $phpgw_setup->oProc->DropColumn('phpgw_acl',$newtbldef,'acl_account_type'); This is the most complicated function in schema_proc, from the user's perspective. Its complexity is necessitated by the requirement of some databases to recreate a table in the case of dropping a column. Note that the table definition array is being used yet again. The array defined here should match the table definition you want after this function has completed. Here, we are dropping the column 'acl_account_type' from the table 'phpgw_acl', and the table definition does not have this column defined. You could copy information from your tables_current file here and edit it to match the desired new table spec, less the column you wish to drop. There are additional functions within schema_proc, the majority of which are not to be called directly. They are used internally. If you do wish to investigate further, use class.schema_proc.inc.php as your guide. This master file includes the class.schema_proc_DBMS.inc.php and class.schema_proc_array.inc.php files. The DBMS files should not be used as a guide, since their functions are called from the master class, and the parameters are different from what you might expect relative to the master. PLEASE, DO NOT WRITE TO OR ALTER ANOTHER APPLICATION'S TABLES OR THE API TABLES IN YOUR APPLICATION UPGRADE FUNCTIONS! default_records.inc.php (Optional) Any application with tables that wants to load some default data will need this file. The default_records file consists of a list of SQL INSERTs using the $oProc object directly: $oProc->query("INSERT INTO phpgw_inv_statuslist (status_name) VALUES ('available')"); $oProc->query("INSERT INTO phpgw_inv_statuslist (status_name) VALUES ('no longer available')"); $oProc->query("INSERT INTO phpgw_inv_statuslist (status_name) VALUES ('back order')"); In this case, the developer wanted to insert some status information, which was then used in a select box on an html form. Using the default_records file, every new install will have this data included. This file should consist of queries applicable to the tables defined in setup.inc.php and tables_current.inc.php. test_data.inc.php (Optional) Any developer wanting to test the full list of upgrade routines can use this file. test_data.inc.php is similar to default_records above. It is called only by schematoy.php and is never installed with a new install or upgrade. This is a developer-only file. The INSERTs here should be applicable to the tables_baseline table definitions. language files (Required) All applications should have at least a file of English translations, used for their application lang() calls. Format of a lang file: {phrase}{TAB}{appname}{TAB}{LANG_CODE}{TAB}{translation} e.g: first name common en First Name first name common de Vorname Filenames: phpgw_{LANG_CODE}.lang e.g. English: phpgw_en.lang German: phpgw_de.lang Please see the contents of the API 'languages' table for the correct setting of the LANG_CODE. Developer Tools sqltoarray.php Displays the current schema_proc array defining an application's tables. This web application reads the current table status live from the database. It then parses this information into a hopefully correct table definition array for schema_proc. Upon visiting this app, you are shown a list of currently installed applications with defined tables. You may then select one app or all apps, and then submit the form. From this form you may then download a tables_current file, suitable for commission to cvs. Please do check the format to make sure the definitions are correct. schematoy.php Runs the full cycle of upgrades, including optional test_data. This app is not beautiful, may bomb on you, and will definitely drop your application's tables. The display is similar to the user/admin tool, applications.php. You are shown a list of apps with tables. Select one app, and enter a target version. Upon submission of the form: All application tables are dropped. tables_baseline.inc.php is loaded. test_data.inc.php is loaded tables_update.inc.php is loaded. a full application upgrade test begins. This will give a LOT of debugging output. Depending on your database, the process may take quite awhile. This tool should be considered as a destructive test of the full upgrade cycle. If the upgrade process is successful, you can then check the loaded test_data to see that it is still in place as expected after all the table modifications, etc. If not, it should be clear where the error has occurred. Look for the usual INVALID SQL warnings, among others. tools subdirectory some utilities for sql file conversion, etc. In the tools directory under setup3, there should be at least a couple of hopefully handy perl or shell scripts. These are for running on the commandline only, and might apply to converting SQL files into lang files, etc. They are not expected to be perfect, but might offer some assistance or ideas for additional utilities. Use these at your own risk or benefit. The install/upgrade process Overview Setup internal upgrade functions Setup uses a common set of functions for new installs and upgrades. These are implemented as multi-pass loops. For a single application install or upgrade, a single pass is done. For multiple application installs or upgrades, multiple passes are done automatically. The order of install in a mass install or upgrade is determined by application dependencies. The other determining factor is the order in which the application directories and setup.inc.php files are read from the filesystem. New installs Detection Each run of index.php or applications.php in setup3 first runs a set of detection routines. These read the data from each setup.inc.php file, and from the 'applications' or 'phpgw_applications' table as appropriate, and only if one of these tables exists. This data is parsed into the $setup_info array. In this case, this array contains information about all applications. Based on the information gathered, a status flag is set to one of the following values: U - Upgrade required/available R - upgrade in pRogress C - upgrade Completed successfully D - Dependency failure F - upgrade Failed V - Version mismatch at end of upgrade M - Missing files at start of upgrade (Not used, proposed only) Using this information, the setup logic in index.php determines what mode we are in. index.php is not capable of being selective about which application it found as being out of sync. It is designed only for 'Simple Application Management', which is Step 1 of the setup process. For more selective application manipulation, use applications.php. index.php then tells the user that 1) their applications are current 2) some of their applications are out of sync 3) no db exists, etc. For a new install, all applications will be out of sync, since there is not even an 'phpgw_applications' table in the database to tell setup what the status is for any application. Selection There is no selection for application installs in 'new install' mode. All physically present applications will be installed, or at least attempted. Installation Once the setup user clicks the magic button to install all applications, the following occurs: The setup_info array is passed to the process_pass() function, using a method='new' Applications whose status flag='U' (API on first pass) are then handed off to the process_current() function. This handles inclusion and installation of the application's tables_current.inc.php file. The application is registered as a new application in the 'phpgw_applications' table. If for some reason there is old data in this table for this application, it will be updated instead. Its hooks, if any, are registered in the 'phpgw_hooks' table. Next, this array is passed to the process_default_records() function. If this file is present in the current application's setup directory, the queries here are run to install the data to the application's table(s). The above is repeated until all application status flags equal 'C'. However, if an application install failed for some reason, it will then be skipped on the next pass. This keeps the loop from running away. Upgrades Detection Only an API version mismatch will trigger an automated request for the user to upgrade their install. Once the api is current, they can move on to applications.php for more 'Advanced Application Management', which is Step 4 of the setup process. However, if the API is out of sync, clicking 'Upgrade' in index.php will also attempt to upgrade other applications which may be out of sync, as well. As the phpgwapi continues to stabilize, it is felt that this method of upgrading will become less and less common. Selection Within applications.php, a color-coded matrix of application status and actions is displayed. Depending on the status flag of each application, certain actions will be either enabled or disabled. These actions include 'install', 'upgrade', 'remove'. If something is very wrong with previous attempts to install or upgrade an application, another column called 'resolution' will then display a link. This link will display additional information which would be helpful for determining how to resolve the problem. Assuming all is well, the user can select applications requiring upgrade from this list. Once selected, they submit the form. This runs the follow three routines in order: remove install upgrade Upgrade The idea here is that multiple actions can be selected and run in order in one click. In any case, once they select an application for upgrade, the following occurs: A stripped down version of the setup_info array is passed to the process_upgrade() function. This array contains only the information for the selected application Within process_upgrade(), the tables_baseline.inc.php file for the application is loaded. The tables_update.inc.php file for the application is loaded The contents of the test array is used to loop through the entire list of upgrade functions for the application. The application's unique function names are rebuilt, then run. When the currentver (installed) matches the version (available), process_upgrade() exits, setting the status flag for the app to 'C'. Just prior to exiting, the application and its hooks are updated into the 'phpgw_applications' and 'phpgw_hooks' tables. Uninstallation/Removal Selection Selective removal of an application is done via applications.php, in a manner similar to the method above for upgrades. Uninstallation Once an application is selected for removal: A stripped down version of the setup_info array is passed to the process_droptables() function. This function removes all of the application's defined tables, but only after first checking to see if the tables are there. In this way, we attempt to cut down on the number of errors sent to the browser. The application's hooks are deregistered (removed from 'phpgw_hooks'). The application itself is deregistered (removed from 'phpgw_applications'). Caveats Must see info Auto fields For auto type fields, schema_proc creates a sequence automatically based on the table name for databases that require sequences. In the case of postgresql, the limit for this name based on our tests is 31 characters. The schema_proc format is: $sSequenceSQL = sprintf("CREATE SEQUENCE seq_%s", $sTableName); This limits the maximum length for a tablename to 27 characters. Based on the tablename standard in phpgw of 'phpgw_tablename', you are further limited to 21 characters in which to describe your table. You will need to be less descriptive in some cases, e.g. use 'phpgw_widget_cats' instead of 'phpgw_widget_info_categories'. Default 0 For int fields, a default of 0 is not assumed. Only some databases will set this default for you, MySQL being one. You will need to explicitly define this default in the table definition. Also, for auto fields, do not enter a default, since the resulting SQL query would fail on many RDBMS.