Managing Projects with GanttPV

The Complete Guide to GanttPV (in process of construction)

Brian C. Christensen
brian@simpleprojectmanagement.com
11/24/2008
This book is available for purchase at:
www.simpleprojectmanagement.com/ganttpv/

Contents

GanttPV

Introducing GanttPV

GanttPV helps project managers with planning projects and tracking performance. This book explains many of the features of GanttPV and how to use them. More importantly, it provides guidance on whether to use them. Like all application programs, GanttPV provides many features *in case* you need them, not *because* you need them. This book will help you to select the features that are most likely to be of real value.

Of all reports used by project managers, the Gantt report is the one that is most distinctively a project management report. GanttPV was named for the Gantt report in the same way that a major office supply chain was named for one product that is used in almost every office. Every project manager should be familiar with the Gantt report and should know its strengths and limitations. GanttPV is capable of many other kinds of reports. The table of contents lists the reports that are covered here.

While GanttPV can be of great use to project managers, it is intended to supplement other general use software. Most project managers will use word processor, spreadsheet, and email software as much as GanttPV. To make it easier to use GanttPV in conjunction with other programs, GanttPV opens its reports in separate small windows. It normally doesn't take over the whole computer screen.

GanttPV can also be used with GanttPV Server to publish reports on the web.

Project Management Terminology

Like all fields of study, project management has its own terminology. Here we define some of the terms that we'll be using frequently.

Term	Definition
Project	A project is an organized effort to make some change that is important to an organization. The first steps in defining a project are to set a goal, identify an approach to reach the goal, reach agreement that the benefits to be derived from the project are worth the cost of the work required to complete it. Once a project is approved, the team, led by the project manager, identifies the tasks that will have to be completed as part of the overall project.
Task	A task is a part of the project that will be scheduled and assigned to a project team member for completion. The work required to complete the task must be clearly understood by both the project manager and the team member assigned the task. For some tasks it may require only a few words to define the task. For others, it may require pages of documentation. For scheduling purposes small tasks may be grouped together or large tasks broken out into smaller sub tasks. The optimum size of tasks for scheduling purposes depends on the frequency of taking status, and often ranges from a half day to several days duration.
Resource	A resource is a person, tool, or facility that is required to complete a project task. For example, if the task is shooting a scene from a movie, resources may include the

Term	Definition
	actors, crew, cameras, and trucks. Normally resources are only entered into the scheduling tool if they must be managed and the assignment of resources to tasks is not obvious.
Schedule	A schedule identifies when each task must be completed for the overall project to be completed by its target date. The schedule takes into account any dependencies between tasks, resource availability, and team preferences.
Dependency, Prerequisite, Successor	Sometimes tasks must be done in a particular order. If so, the constraints on the order that tasks can be performed are called dependencies. For example, a trench must be dug before pipe can be laid; the walls should be painted before paintings are hung. In these examples, the tasks that must completed first are called prerequisite tasks; the following tasks are called successor tasks.
Start Date	In the schedule, each task has a planned start date. This date is when the task should be started in order for the project to be completed on time. The start date may be specified or calculated.
End Date	Each task is expected to be completed on its end date.
Effort Hours	Effort hours are the number of hours that a person works to complete a task. It comes in three varieties: plan, baseline, and actual. Planned effort hours are the number of hours of work that it is expected will be required to complete a task. If the plan is revised, the original plan is referred to as the baseline. Only after a task is completed can the actual effort hours can be known.
Duration Hours	Duration hours are the number of hours between when a task is scheduled to start and when it is scheduled to finish. It comes in three flavors: the schedule is based on planned duration (what is expected before the task is started); baseline duration refers to the original plan if the plan has been adjusted based on new information; actual duration is only known after the task has been completed. Duration is the number of clock hours that are available for work to be done whether or not work is actually done during those hours. Duration is the difference between the start date and the end date.
Plan	Before starting the work, the team determines what tasks need to be done and when. Tasks are defined, dependencies identified, durations and effort estimated, and a schedule prepared. The plan consists of the actions that the team is expected to take in order to complete the project by the deadline.
Baseline	Sometimes it is necessary to revise plans. The baseline is the version of the plan that represents the commitment that the team has made. The team's actual performance is compared to the baseline.
Actual	Only after a task is completed are the actual duration, effort, and completion dates known. Actual performance is tracked to determine what changes will be required to keep the project on schedule. It is also the basis for improved planning skills.

GanttPV Files

GanttPV database files are identified with the suffix ".ganttpv'. In GanttPV, only one database file can be open at a time. Each database file can contain many projects and each project can contain many reports. Each reports can be opened in its own window. When multiple windows are open, it is easy to return to the Main window -- just type Ctrl-1. Any open report may be directly selected via the Window menu, which contains a list of all of the currently open report windows. When a database is opened, any reports that were opened with the database was last saved are automatically opened in the same

location and size. These small report windows are intended to make is easy for you to open a GanttPV report alongside windows from any other application.

GanttPV keeps track of all of the changes that have been made to the database file from when it was opened. The most recent change can be reversed by typing Ctrl-z or selecting Undo from the Edit menu. If desired, the change can be reapplied by typing Ctrl-Shift-z or selecting Redo from the Edit menu. Any number of changes can be reversed or reapplied in this fashion. Changes do not become permanent until the file is saved. Changes can be saved by typing Ctrl-s or selecting Save from the File menu.

Often it is useful to create an extra copy of the database. For example, when you have a good draft of your project schedule, but have some other ideas that you would like to experiment with. To create a backup copy of the database, type Ctrl-b. A copy of the current database is saved using a name that is formed by adding the current date and time to the end of the file name. The file is saved into the same directory as the original file.

Dates and Hours

Task Start Dates

In GanttPV there are only two ways to specify when a task should start. The most common and most convenient is to specify the prerequisite task. This is done by clicking on the row number to select a task row and then clicking on the Assign Prerequisite toolbar button. The other approach is to enter a suggested Start Date. Both can be entered if required.

Entering Dates in GanttPV

Dates in GanttPV are in this format: 2008-08-28. It consists of a four digit year, a two-digit month, and a two digit day. This is an international date format standard.

GanttPV provides several short cuts for entering dates. If only the day of the month is entered, GanttPV will add in the current month and year. If the month and day are provided, GanttPV will fill in the current year.

How to Plan a Project

In planning a project, the manager must decide what tasks must be completed in order to accomplish the objectives of the project. This book focuses on those tasks where GanttPV can help. For more on the whole task of the project managers, see my book *Simple Project Management: What Everyone Needs to Know About Project Management*.

Where to start? Here is a brief summary of the steps that are identified in *Simple Project Management*

1. Identify and write down the project objectives and scope.
2. Work with the customer to clarify what the project is supposed to accomplish and why it is important.
3. Decide on the approach that will be use.
4. Identify the work that must be done by the project team to reach the objectives.

5. Identify who will be doing the work.
6. Split the work between tasks that will be schedule and assigned to team members and follow up items that will be checked off. Work with the team to create a schedule that will allow the project to be completed on time.
7. Follow the plan. Monitor performance and fix problems as they are identified.
8. Communicate.

In some projects, the schedule is almost determined from dependencies between tasks. If strict dependencies between tasks can be identified, enter those into the scheduling tool. If the order in which tasks can be completed is flexible, sort the task out among the people who will be doing the work. Decide the preferred order of task completion and enter that order using dependencies.

Dependencies, Prerequisites, and Successors

Tasks are said to be dependent if one of the tasks, call it Task B, can't be started until another task, call it Task A has been substantially completed. For example, it makes little sense to hang paintings in a room (Task B) until after the walls have been painted and the paint has dried (Task A). "Hanging the paintings" is said to be dependent on the completion of "Painting the walls". Another way of saying this is to say that "Painting the walls" is a prerequisite to "Hanging the paintings" and that "Hanging the paintings" is the successor to "painting the walls".

In identifying dependencies, only the immediate dependencies are identified. For example, if Task A is a prerequisite of Task B and Task B is a prerequisite to Task C, it would technically be correct to say that Task C is dependent on Task A because Task C cannot be completed until after Task A. But we typically do not specify dependencies that are already implied by other, more immediate dependencies.

Dependencies	Task A -> Task B -> Task C
Entered into tool	Task A -> Task B
	Task B -> Task C
Not entered	Task A -> Task C

Duration vs. Effort

Duration refers to the number of hours that pass regardless of whether work is being done during those hours. Effort refers only to the hours actively spend working on the task. For example, "painting the walls" may only require two hours of effort but it may have a duration of one day to allow time for the paint to dry. The task isn't done until the walls are dry. But after two hours of actual effort, no one is required to work to make the walls dry.

For the task "hang the paintings" effort hours might not be entered because the person working on this task is expected to be working continuously. In other words, the number of hours of duration is expected to equal the number of hours of effort.

The actual calculation of duration hours is complicated a bit by the idea of working hours. Normally, when we say that a task is expected to take one day to complete, we don't mean 24 hours. We mean that it will be completed in the number of hours that are normally worked during a day. Similarly, when

a task is expected to be completed in a week, normally that doesn't mean 24 hours per day for seven days. Rather it is understood to mean during the hours of a normal work week. In GanttPV the number of normal working hours can be managed by editing the work week. The work week should be set to the number of hours that people will normally be working each day of the week. By default, the work week is eight hours per day for Monday through Friday and no hours and the week end. This can be changed using the Edit Work Week script. It is also possible to set a different number of hours for any given day. This is done using the holiday report.

Menus and Tool Bars

GanttPV Menus

The following is a detailed explanation of all of the menus and tool bars in GanttPV.

Main window

File Menu

File Menu is nearly identical for the all windows. Differences are noted on each menu option.

Menu Item	Effect
New	Create a new GanttPV database file. The new file contains only the default minimum set of reports and report types.
Open…	Opens a dialog box that can be used to locate an existing GanttPV database file to open.
Open Selected Report	Opens the selected report in a new window. This is equivalent to double clicking on the report row. (Only applies to the Main window.)
Close Report	Closes the current report window (does not apply to the Main window)
Close All Reports	Closes all open report windows except the Main window.
Save	Save any uncommitted changes replacing the prior save version of the file.
Save As …	Open a dialog box that can be use to specify where the database should be saved and under what name. The new name and location will be used by the save command.
Automatically Save on Quit	Selecting this menu option will toggle the check mark on this item. If the check mark is present, any unsaved changes will automatically be saved when the application is closed. Otherwise, a dialog box will be opened to asked what should be done with any unsaved changes.
Revert to Last Saved	All changes that have been made to the file since the last save will be discarded.
Revert to Last Opened	All changes made to the file since it was opened will be discarded.
Quit	Exit the application.

Edit Menu

Menu Item	Effect
Undo <message>	Reverses the last group of change (that are briefly described by the message). The undo applies to the most recent change regardless of which report or project the change was made in.
Redo <message>	Reapplies the most recently undone change.
Insert New Project	Adds an new project to the current GanttPV database.
Insert New Report …	Used to add new reports to a project. The command opens a window that lists all of the currently installed report types. The user selects the reports to be inserted. Each report has a default set of columns.
Delete Selected Row	Can be used to delete a report or a project. If a project is deleted, all of its reports are deleted also,

Move Selected Row Up **Move Selected Row Down**	Used to rearrange the order of projects and reports. A selected project can be moved up or down the list. Reports can be reordered within a project.
Edit Project Name …	Use to rename a project.
Edit Report Name …	Use to rename a report.

Script Menu

Menu Item	Effect
Choose Scripts Folder …	Use to select the folder (also called a directory) that contains scripts to be displayed in the scripts menu.
Refresh Script Menu	Update the script menu to include any new or renamed scripts.
Repeat Previous Script	Re-run the most recently selected script.

Window Menu

Menu Item	Effect
Main Window	Brings the main report to the front.
<Report Name>	Lists the names of all the open reports. Selecting a report will bring it to the front. (On Windows, if a report has been minimized, this command will not maximize the report. That must be done from the task bar.)

Help Menu

Menu Item	Effect
About GanttPV …	Displays the GanttPV version number.
Quick Start …	Explains the most vital information on how to use GanttPV.
ORM Quick Start …	Explains how to use the Object Role Modeling add-in to GanttPV. The add-in is sometimes called ORM-LITE.
Short Cuts …	Explains many of the most useful keyboard short cuts.
GanttPV Home	Opens the web browser to the GanttPV Home page. Most of the standard GanttPV documentation is located at this web site.
Help Page and Tour	Opens the web browser to the GanttPV Tour. The tour is explains how to use the main features of GanttPV.
Forum	The forum is a good place to search for answers to questions about GanttPV. It is also a good place to ask how to solve a specific problem.

Tool Bar

Menu Item	Effect
New Project	Same as Edit -> Insert New Project
New Report	Same as Edit -> Insert New Report
Duplicate	--
Delete	Same as Edit -> Delete Row
Show Hidden	Toggles the visibility of deleted projects and reports. The deleted projects and reports are colored red. Using the Delete command on a red row will "undelete" it.
Move Row Up **Move Row Down**	Same as Edit -> Move Selected Row Up (or Down)

Grid Report

File, Script, Window, and Help menus are identical to the Main Window menus.

Edit Menu

Menu Item	Effect
Undo <message>	Reverse the effect of the last command that changed the database. Some actions cannot be reversed by undo (for example, moving a window). The message shows the nature of the change that will be undone.
Redo <message>	Reapply the changes from the last Undo command.
Copy	The contents of the selected cells are copied to the clipboard
Paste	The contents of the clipboard are entered in to cells starting with the selected cell, or if more that one is selected, starting with the upper left cell. Protected cells are not changed. Additional rows are added to the end of the report if required in order to accommodate all of the rows of data in the clipboard.
Insert New <type>	Insert a new row after row that current selection (the cell that is outlined in black). If a task report, insert a new task row. If a resource report, insert a new resource row. In two-table reports, the row that is added is of the same type as the selected row.
Insert New <type> Above Selection	Same as Insert New but before the selection
Insert Related <type>	This only applies to two-table reports. An new row of the second type will be inserted after the selected row of the primary type.
Delete Selected Row	Marks the currently selected rows as deleted and hides them. In two-table reports, if a primary table row is deleted, all of its secondary rows are also hidden
Shift Left **Shift Right**	These commands are only available if a hierarchy has been installed for the report type. These commands will increase (or decrease) the indentation of the selected rows. For example, if Task Parenting is installed, the selected rows will become sub-tasks of the prior task.
Move Selected Row Up **Move Selected Row Down**	Move the selected rows towards the top (or bottom) of the report. Child rows are kept with their parents.
Edit Project Name	Same as double-clicking on the project name in the Main window
Edit Report Name	Opens a dialog box that to change the name of the current report.

Action Menu

Menu Item	Effect
Define Prerequisite Task	Open a window that lists all of the other tasks in the report. Selected the prerequisite tasks and click OK. Tasks displayed may be filtered typing in a character string.
Link Selected Tasks as Dependent	Create a dependency between the each pair of selected tasks in the order that they appear in the report.
Assign Resource to Task	Opens a dialog box that lists all of the resources. Select the specific resources to assign to this task. Resources displayed may be filtered by typing in a character string.

View Menu

Menu Item	Effect
Insert Column …	Open a window that lists all of the other tasks in the report. Selected the prerequisite tasks and click OK. Tasks displayed may be filtered typing in a character string.
Delete Selected Column …	Deleted the selected column from the report.
Move Selected Column Left **Move Selected Column Right**	Rearrange the order of the report columns.
Scroll Time Scale Left Fast **Scroll Time Scale Left** **Scroll Time Scale Right** **Scroll Time Scale Right Fast**	Used to change the range of dates that are displayed in a time scale. If no time scale is selected, the left most time scale in the report is affected.
Scroll Time Scale to Task Start Date	Applies only to Gantt charts. This command will scroll the time scale so that bar for the selected task is visible.

Tool Bar

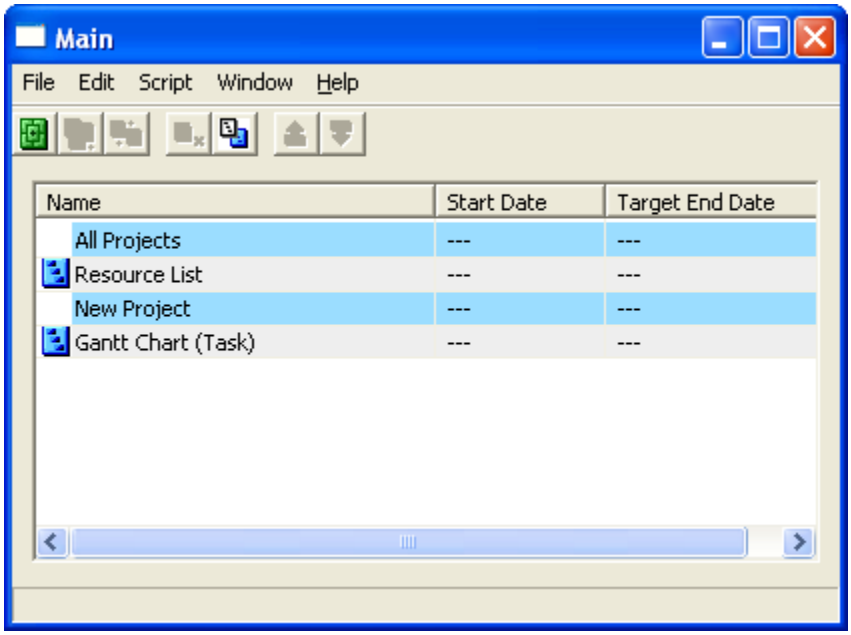
Menu Item	Effect
	All tool bar buttons provide direct access to one of the above menu items.

Reports in GanttPV

GanttPV Reports

Main Window

When you first open GanttPV you'll see the Main window. This window is something of a command center for GanttPV. Use this window to add new projects to a ganttpv database and to add new reports to a project. Closing this window will cause GanttPV to exit.



When the application is first opened, a new database is automatically created. This database includes two projects that each contains a single report.

The first project is named "All Projects" (sometimes referred to as "Project 1") is a special project. It contains reports that apply to all of the other projects. At first this project contains only one report, called "Resource List".

The second project is named "New Project". Double click on the project row to change the project name. Initially the project contains only one report, "Gantt Chart (Task)". Double-click on the report name and a new window will open that displays the Gantt Chart report.

Report Types

Reports can be opened from the main window by double clicking anywhere on the row that contains the report name or single clicking anywhere on the row and selecting "Open Selected Report" from the File menu.

There are two different types of reports. The Gantt Report and the Resource List are examples of Grid Reports. They are similar in appearance to a spreadsheet with rows and columns of data. A new addition to GanttPV is the Diagram report. These reports look contain squares and circles connected by lines. Each type of report has its own menus and toolbar buttons.

Grid Reports

Grid reports can be one-table or two-table reports. In one-table reports, all of the rows are of the same type. For example, every row in a Project report represents a project. Every row in a Task report represents a Task. In two-table reports, two related tables are included. For example, in a Task/Assignment report, all of the primary rows represent the tasks in a project. Beneath each task are listed all of the associated assignments.

By default Grid reports automatically include all of the corresponding objects. For example, if a new Task is added to a project in the database, it is automatically added to that project's Task report.

To rearrange report rows, first select the rows to move by clicking on the row numbers (not the ID column). Use a combination of click, shift-click (to select a range of columns), and ctrl-click. Then use the up and down buttons to move the rows to the desired location (or better, use the keyboard shortcuts Ctrl-9 and Ctrl-0). The first up or down click will bring all of the selected rows together. This feature can be used to bring rows from the bottom of the report in one action. First select the rows to move, add to the selection a row at the top of the report, and click the up bottom. The rows are instantly brought to the top of the report.

Moving report rows in two column reports works the same except that the child rows will always stay with their parent.

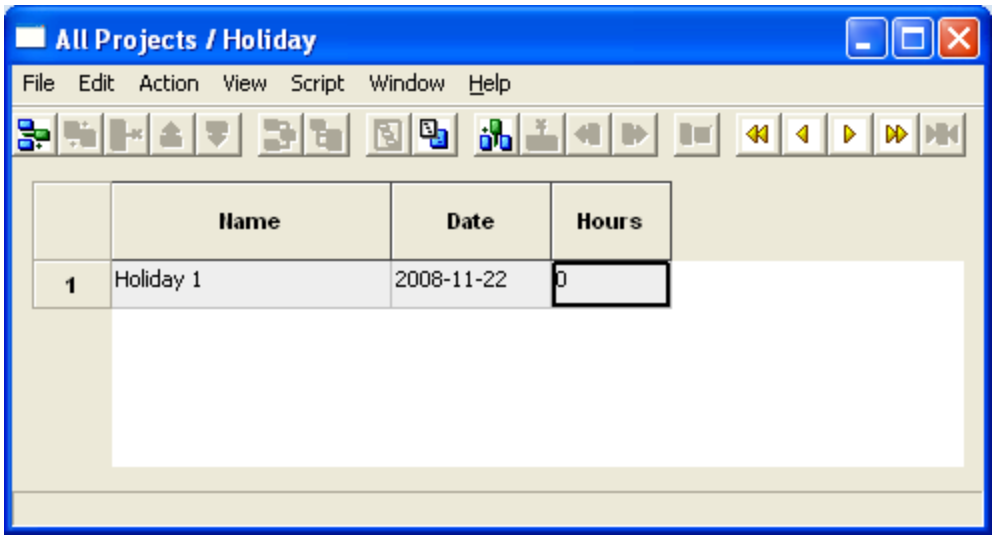
When working with rows in a hierarchy, note that GanttPV will only work with the highest level selected rows. GanttPV will not change parent with the up and down arrow buttons.

In the following sections many of the GanttPV reports are explained in detail.

Holiday Report

In GanttPV, the holiday report is used to change the number of expected working hours during a day. Typically, on holidays the number of working hours is reduced to zero, but it can be set to any number. It is even possible to set "reverse holidays" were the number of work hours for a particular day is increased up to 24 hours. This is useful for a planed "all-nighter".

To set holidays, go to the main report. Select the "All Projects" row. Click the New Report toolbar button and select Holiday. Click OK. Then double-click the holiday report to open it. Type Ctrl-r to add new dates then enter the date and the number of hours. Optionally add a description of the holiday or an explanation for the exception in the normal number of hours that day.



All of the projects in the same GanttPV database share the same work week and holidays. If a project must use a different calendar, it will have to be in a separate database file. Usually it is possible to use a standard calendar for all projects.

Column Name	Description
Name **	Optional. This is a name of the holiday.
Date **	The holiday date in YYYY-MM-DD format.
Hours **	Overrides the default number of hours for the specified day. Normally set to zero, but can be set to any positive value.
ID *	The ID of this Holiday object.

Project Report

Project One only with one row per project. This contains the same information as the primary rows in the Main Window except that these values can be directly edited (the corresponding columns in the main window are locked). Use this report to change project Start Dates and Target End Dates.

Column Name	Description
Name **	Contain a brief description of the project.
Start Date **	Optional. The desired start date. Tasks without prerequisites or start dates will be schedule starting from this date.
Target End Date **	Optional. Desired completion date of the project. Not currently used by GanttPV.
Week/Effort Hours Time Scale **	These are the planned effort hours by project week based on the most recent results from the Calculate Assignment Hours script.
ID *	The unique project ID's.
Actual End Date *	Optional. Actual project end date. Not currently used by GanttPV.

Project/Report Report

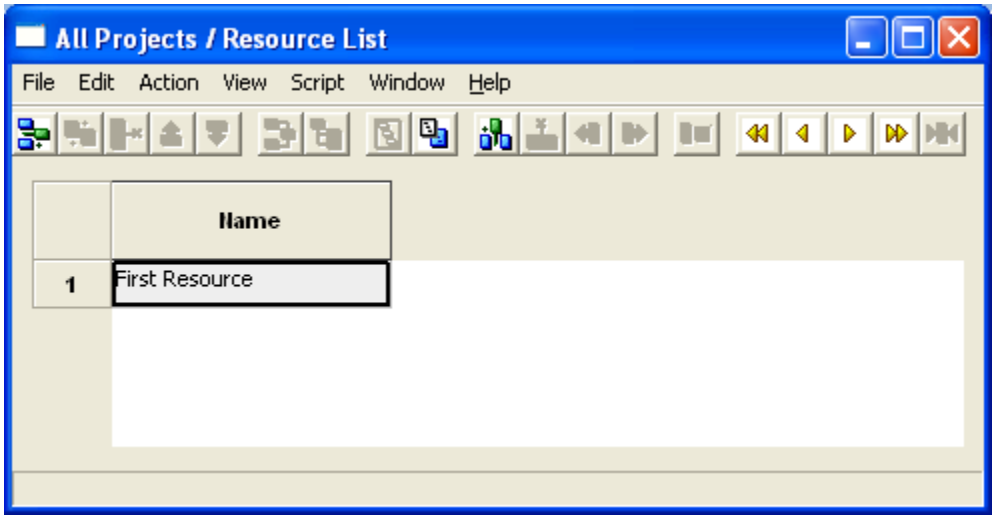
Project One only. Similar to the Main Window except that the values here can be edited (the corresponding columns in the Main Window are locked.

Column Name	Description

ID **	Report ID number
Name **	Report Name

Resource Report

Double click on this report to open it. A new window opens up to display the report. The report consists of one column, "Name", that contains the names of the resources that can be assigned to tasks in projects. An example resource has already been created for you. The resource is temporarily named "First Resource". It was included to make demonstrate what resource rows look like. Feel free to change the name to that of one of the real project resources. To edit the resource name, click into the cell that contains the resource name and type in a new name. Press enter to accept the change or escape ("Esc") to discard the change. To add a new resource press Ctrl-r. Close the Resource List window when you have finished making changes. Note that your changes are not permanent until they saved into the database file via the save menu command (or use the short cut Ctrl-s).



Column Name	Description
Name **	Resource Name
ID *	Unique indentifying number for each resource.
Short Name *	Brief name for resource.
Day/Effort *	Planned effort hours per day for this resource based on the Calculate Assignment Hours script.
Week/Effort *	Planned effort hours per week for this resource based on the Calculate Assignment Hours script.

Resource/Assignment Report

Project One only. The report lists all of the resources in GanttPV and all of their past, current, and future assignments. To hide resources, select the row number in the left margin of the report and click the hide row tool bar button.

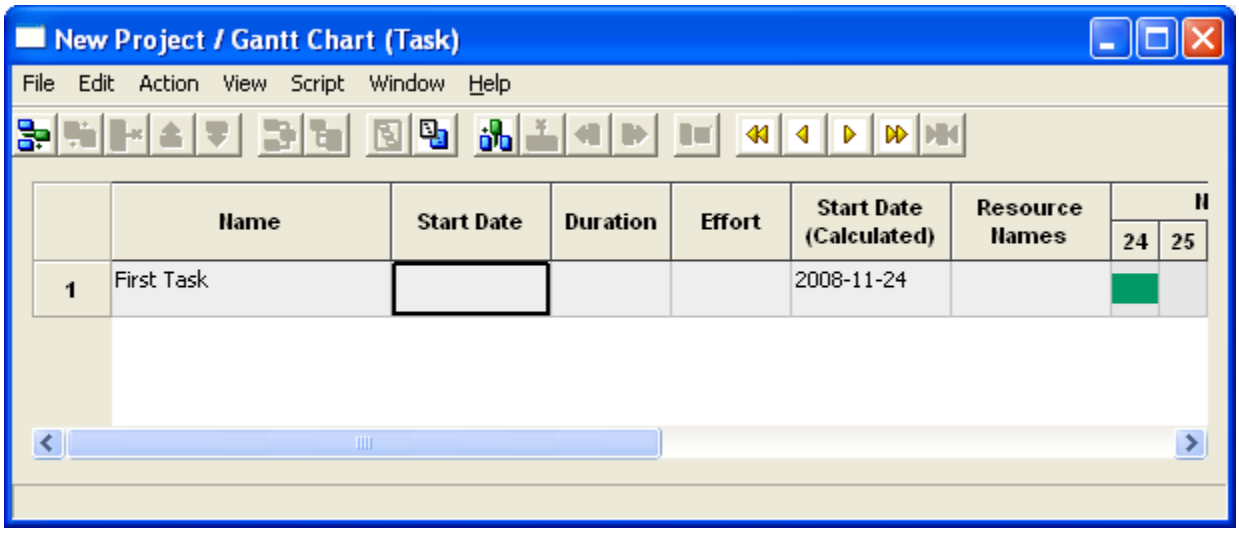
Column Name	Description
Task/Name **	Contain a brief description of the assigned task.
Assignment	Total effort hours from this task that have been specifically allocated to this person.

Effort **	Usually best edit this value in the Task/Assignment report.
Day/Effort **	Planned effort hours per day for this resource assignment based on the Calculate Assignment Hours script.
Week/Effort **	Planned effort hours per week for this resource assignment based on the Calculate Assignment Hours script.

Task Report (Gantt Chart Report)

GanttPV is named for the Gantt chart. Of the many charts and graphs used by project managers the Gantt chart is both widely used and distinctively a project management report. The great strength of the Gantt chart is that it graphically displays each of the major tasks that must be completed in order to reach the goals of the project. It also shows when those tasks must be completed if the project's target date is to be met.

The Gantt chart is named for Gantt. Originally developed to handle project …., the techniques have proven useful and are now applied to projects of any size and shape.



When to Use a Gantt Chart?

The Gantt chart is most useful for projects where the tasks, priorities, and available resources can be identified. In order for a Gantt chart to be created it is important that most of tasks required to complete the project can be identified early on and do not have to be discovered during the project. In other words, if the tasks that must be completed cannot be identified, it is difficult to schedule them. It is also important that the project priority be known. If other higher priority tasks will take precedence over project tasks, then it would be difficult to say with any confidence when the tasks can be completed. If resources are not known, then again it is difficult to say how long a task will take to complete.

GanttPV can be used to plan projects that don't match these criteria, but they must be handled differently. For example, if resource availability is not known or priorities are expected to shift, the project manager will have to work on the project in the gaps between other projects. The best way to track tasks in that case is often by using just a task list.

Other projects consist of tasks that are too small to really need to be scheduled. GanttPV provides a follow up track feature.

Each row in this report corresponds to one task. Let's examine the columns in the Gantt Chart report.

Column Name	Description
Actual End Date *	Date the task was completed.
Day/Effort *	Planned effort hours per day for task calculated by Calculate Assignment Hours script.
Day/Gantt **	A timescale column consists of multiple parts that each represent on period in the time scale. The number of time periods can be changed by selecting the time period columns and running the Edit Column Options script.
Duration **	Should be provided for all tasks. Duration is the number of regular working hours between the time the task begins and when it should be finished.
Effort **	The number of hours of effort should be required to complete the task. It is not necessary to enter Effort unless GanttPV is used to help in resource management. Sometimes GanttPV can infer the effort hours. Because of this, don't enter Effort if the resources assigned to the task are expected to work on the task full time.
End Date (Calculated) *	Planned task end date calculated using the critical path method.
Name **	Contain a brief description of the task.
Project ID *	ID of project that contains this task. (Normally only used in Project One report.)
Project Name *	Name of project that contains this task. (Normally only used in Project One report.)
Resource Names **	Column displays the name of each resource currently assigned to this task. To set resources select the row by clicking on the row number in the left margin of the report and clicking the Assign Resources toolbar button.
Resource Short Names *	Comma separated list of resource names for resources assigned to this task.
Start Date **	The desired start date. This date acts as a "don't start this task before this date". Usually it is better to leave the desired start date blank and specify a prerequisite instead. A prerequisite task can be set by clicking on the row number in the left margin of the report and clicking the Assign Prerequisites toolbar button.
Start Date (Calculated) **	Shows the planned start date for the task that GanttPV has calculated by examining the start dates and dependencies of all of the tasks in the project. This date cannot be edited directly.
Week/Effort *	Planned effort hours per week for task calculated by Calculate Assignment Hours script.

Task/Assignment Report

Includes all of the task report columns plus:

Column Name	Description
Assignment Effort **	Planned effort hours of this task allocated to this resource assignment.
ID *	Unique ID for this specific assignment.
Resource Name	The name of one of the resources that has been assigned to this task.

****	
Role **	Can be used to identify the role played by the resource on this task.
Day/Effort **	Planned effort hours per day for this assignment.
Week/Effort *	Planned effort hours per week for this assignment.

Task/Dependency Report

Includes all of the task report columns plus:

Column Name	Description
Prerequiste/Name **	Contain a brief description of the task.

Scripts in GanttPV

GanttPV Scripts

From the first release, GanttPV has been scriptable. That means that anyone with a little programming knowledge can add new commands to GanttPV. With release v0.10, it has become much easier to write scripts for GanttPV with the introduction of a new object oriented programming feature. This section explains how to write scripts for GanttPV.

Scripts in GanttPV are written using the Python programming language. Fortunately, Python is one of the simplest programming languages to learn and to read. However, even though Python is simple for a programming language, it does take time and motivation to learn.

This section will explain the parts of Python most commonly used for GanttPV scripts and will give many examples of its use. However, it doesn't take the place of spending an hour or two working through the first part of the Python tutorial at: http://docs.python.org/tutorial/

Installing GanttPV Scripts

Scripts available to GanttPV appear in the Scripts menu. New scripts must be placed in the Scripts folder (sometimes referred to as a directory) before they can be used by GanttPV. The menu command "Scripts -> Choose Scripts Folder …" tells GanttPV where the Scripts folder is located. After a new script is added to the Scripts folder, the menu must be refreshed to display the new script by using the "Refresh Script Menu" command. It is also necessary to refresh the menu if a script is renamed, but not if only the contents of a script have been changed. GanttPV always uses the current contents of the script.

Often the easiest way to create a new script is to find an existing script that performs a similar function, copy the script, and then modify it. It is useful to view scripts with an editor and to scan through them. This will, at a minimum, give a sense of the difference between what is simple to do and what is complicated. Usually it is easier to modify shorter scripts.

Script names must end in ".py". Scripts should be edited with a plain text editor, like NotePad or WordPad on Windows, or TextEdit on the Macintosh. If Python is installed on your computer, you can use the IDLE script editor that comes with Python. This has the advantage that the keywords in Python are color coded which makes it easier to visually verify that the script has been coded correctly.

In order for a script to run correctly, it must be free from errors. Unfortunately, a new script rarely works correctly the first time. As the proverb goes, "to err is human". Python will display an error message if identifies an error in the script. In order to see these errors, it is useful to run the script "Script Debugging -> Show Messages". This script will open a window that will display any informational or error messages that the script prints. Error messages will usually display the line in the script where the error occurred and its line number. IDLE has a convenient command that move cursor directly to the

specified line number. While the messages window is a vital debugging aid, it does slow down GanttPV commands and scripts. It is best to close the window when it is not needed.

Common errors include:

- leaving off the colon (":") at the end of "for" or "if" statements,
- misspelling variable names,
- failing to close quotes or to close parentheses.

Sometime Python is able to explain exactly what the error is. Other times it can only indicate approximately were in the script the error appears.

When all of the error messages have be addressed, it is still important to verify that the script produced the correct results. Typically this is done by examining the starting data to determine what result the script should produce and then comparing that with what actually occurs. Always make a backup copy the database before running new scripts.

While the Show Messages window is open, it is possible to observer that GanttPV displays many messages during its normal operation. These messages were added by the GanttPV developers to help assure that the program is working properly.

Export Scripts

What kinds of scripts can be written for GanttPV? The first kind of script covered here is a data export script. These are the easiest to write and probably the most widely useful.

In order to extract information from GanttPV, it is helpful to have some understanding of how the data is organized. We call this the data model. All of the data in GanttPV is part of a database file that has a file name that ends with the ".ganttpv" suffix. When writing scripts this data can be accessed using the GanttPV object model. The object types include: Project, Task, Resource, Assignment, Dependency, and Holiday. A complete description of the GanttPV data model is available at the end of this document. You may wish to refer to that documentation while reading this section.

First Example – Export Project Names

The first step in writing an export script is to get clear on exactly what information we would like to extract. It will be easier to start with an overly simple example. For this example, the objective will be to export a list of project names in the database. While it would be easy to create a Project report in GanttPV, copy the names, and paste them into a text file, we'll create a script. We'll use what is learned as a foundation for more complex scripts later.

The second step is to examine the data model and determine where the desired data is located. In this case, the Project names are available in the Project objects.

The third step is to decide exactly what steps must be taken to extract out that data. Here is what the script needs to do: (1) the script will start by making a list of all the projects in the database, (2) then It will next open a file to store the project names, (3) the script will go through the project list, one project

at a time, and retrieve that project's name, (4) it will write the name to the output file, and (5) finally, it will close the file and end.

The fourth step is to translate those steps into Python. Here is a Python script that writes out a file that contains the names of all the projects in the database:

```
1. projectList = Data.DBObject.GetList('Project')
2. output  = open(r'project list.txt', 'w')
3. for project in projectList:
4.     output.write(project.Name)
5. output.close
```

The line numbers in the examples are just provided for reference. They do not appear in Python scripts. In line #1, projectList is a variable. A variable is a place where information can be stored. In this case, we will use it to store the list of Project objects. The equal sign ("=") means to set the variable on the left to the value of the expression on the right. "Data.DBObject" is a variable that contains a Database Object that includes all of the data from the GanttPV database file. ".GetList" function requests that the Database Object return a list of all of the objects in the database of the requested type, in this case all of the objects of type "Project". To get a list of any other object type, we could use the same ".GetList" command and supply a different object type name.

In line #2, we tell GanttPV to open a new file named "project list.txt". The "w" means that we will be "writing" new data to the file. A new "file" object is created and saved in the variable "output". We will use that "file" object to write data to the file. In practice, the file name in quotes should be changed to include the directory path where the file should be written. The small "r" before the first quote mark means that special characters like back slashes that are sometimes include in file names should be treated as normal characters.

In line #3, we have a "for … in" statement. This statement takes each object out of "projectList" one at a time and puts it into the variable named "project".

Line #4 is indented to show that it is applied to each "project". (Use spaces not tabs to indent in Python. Use four spaces for each level of indentation. The IDLE editor makes this easy.) This line "writes" the project's "Name" to the output file.

Line #5 tells the file that we are done writing data to the file.

When a python script finishes with the last command in the file it stops.

The sixth step in developing a script is to test the script to make sure that it works. Remember to change the file name to include the full directory path.

Debugging Hints

Python can only write data to a file in a directory where it has permission to do so. If the script doesn't have permission, a "permission denied" error may result. Because the directory name is not included with the file name line #2, the script will try to write the file to what is called the "current directory".

Most of the time this works well enough. This behavior can be overridden by including the full directory path in the name in line #2.

Second Example - Export Resources

This second example will show more about how to follow connections in the data model. This script will collect the names and email addresses of all of the shared resources in the database.

Resources that are shared across projects are contained in the Resource List in Project One. Remember that Project One is a special project. In contains information that is shared by all of the other projects. The two main kinds of information shared by all projects are Resources and Holidays. Because it is possible to add resources to other projects, it will be important to make sure the script retrieves only resources that belong to Project One.

In the data model there is a connection between Resources and Projects. This connection can be followed in either direction. That is, from a specific resource it is possible to identify the project to which it belongs by saying "resource.Project". A resource always belongs to exactly one project. It is also possible to go the other way. From an project, it is possible to find all of the resources that belong to it by saying "project.GetList('Resource'). A project may include any number of resources (including zero resources). Note the difference in syntax.

The specific steps the script should take are as follows: (1) the script should identify project one, (2) retrieve all of the resources that are tied to that specific project, (3) open the output file, (4) retrieve the name and email of each project one resource, (5) write them to the file, and (6) close the file and end. Note that the output file can be opened anytime before processing each of the resource in the list.

Here is a Python script that writes out a file that contains the names and email address of all the shared Resources:

```
1. projectOne = Data.DBObject.GetObject('Project', 1)
2. resourceList = projectOne.GetList('Resource')
3. output  = open('resource email addresses.txt', 'w')
4. for resource in resourceList:
5.     name = resource.Name or ''
6.     email = resource.Email or ''
7.     output.write(name + '\t' + email)
8. output.close
```

Line #1 is similar to that of the prior script. The main difference is that instead of requesting all of the Project objects, it is requesting a specific Project, the one with ID number 1.

In Line #2, we request the Resources that belong to Project 1. Given a Resource object we could retrieve its Project by using the phrase "resource.Project".

Lines #3, 4, and 6 are the same as in the prior script but with different file and variable names.

Line #5 retrieves the resource's Name. The phrase " `or ''` " is a Python idiom that means "if the name is missing use an empty string instead". Line#6 retrieves the resources Email address. (The email column can be made available for the resource report by running the install script "Install Resource Tracking".)

Line #7 writes out the Resource name and email address separated by a tab character. This format is convenient for importing into a program like Excel or MS Word.

Variable names in Python can be almost any combinations of letters, numbers, and the underscore character. Variable names must start with an upper or lower case letter. Certain names are reserved by Python and cannot be used as variable names. We have already seen several of them, for example "for", and "in". Following is a complete list of the Python reserved words. (For more information on the see the Python language reference manual at http://docs.python.org/reference/ .) There are also a few additional words that shouldn't be used because they already have a meaning assigned in Python ("close", "float" , "input", "int", "open", "range", "type") or in GanttPV ("Data", "ID").

and	del	from	not	while
as	elif	global	or	with
assert	else	if	pass	yield
break	except	import	print	
class	exec	in	raise	
continue	finally	is	return	
def	for	lambda	try	

Most of the other information in GanttPV is specific to a project. For example, a Project may contain any number of Tasks or Follow Up items. Tasks can be related to each other via a Dependency or to a Resource via an Assignment.

Third Example - Export Tasks

Here is a Python script that writes out Task information for a Project:

```
1.  def doExport(path):
2.      project = Data.DBObject.GetObject('Project', 2)
3.      taskList = project.GetList('Task')
4.      output  = open(path, 'w')
5.      for task in taskList:
6.          name = task.Name or ''
7.          startdate = task.CalculatedStartDate or ''
8.          durationhours = task.DurationHours
9.          if durationhours == None:
10.             durationhours = 8
11.         output.write(name + '\t' + startdate
12.             + '\t' + str(durationhours))
13.     output.close
14.
```

```
15. doExport('task information.txt')
```

Compare this script to the previous one. Most of the script is very similar to the prior one. Only a few things are new.

In line #1 the script defines a function named “doExport”. A function is a series of commands that are given a name. Functions can take paramenters. In this case there is one parameter, named “path”.

The “open” command in line #4 has changed from the prior example. The name of the file has been moved out of the “open” statement and replace with the parameter name “path”.

In lines #8 to 10, the “or 8” idiom isn’t used for hours. The reason this is that it would change zero hours into eight hours. The “or” idiom will use the second value if the first value is one that Python considers to be a “False” value. “False” values include: a missing value, a zero, an empty string, or an empty list.

In line #12, the function str() is used to convert the number to a string so that it can be printed.

Line #15 is new. “doExport” will run the function that was defined in lines #1 to #8. It also provides the output file’s path name as a parameter. The file name is placed inside the “path” parameter in line #1 and used in the “open” command in line #4.

The rest of the script is very similar in structure to the prior example.

Script Techniques

These first few examples have been complete scripts. Most of the other examples will be small excerpts from scripts to illustrate a particular technique.

For example, here is the way to identify the resources that are assigned to a task. In the data model, notice that Tasks are connected to Assignments and Assignments are connected to Resources. This example finds the specific task of interest, retrieves its Assignments, and from each assignment retrieves the Resource.

```
1. task = Data.DBObject.GetObject('Task', 1)
2. resourceList = [ assignment.Resource for assignment
3.                  in task.GetList('Assignment')
4.                  if assignment.Resource ]
```

This uses a powerful feature of Python called the List Comprehension. Let’s consider what it does in pieces.

We start with a Task object. In this example, we request a Task with a specific ID number, but we could have gotten the Task object as in the prior script example. We get the list of all Assignments that refer to that Task. This is done by the phrase “task.GetList('Assignment')”. We take each of those Assignment objects and place them in the variable “assignment” and then retrieve the associated Resource object using the phrase “assignment.Resource”. If the assignment does not refer to a valid resource it is omitted. Finally, these resource objects are made into a list and stored in the variable “resourceList”.

Here is another excerpt that produces the same result, but does not use a List Comprehension.

```
1. task = Data.DBObject.GetObject('Task', 1)
2. resourceList = [ ]
3. for assignment in task.GetList('Assignment'):
4.     if assignment.Resource:
5.         resourceList.append(assignment.Resource)
```

Line #1 is identical to the prior example. It retrieves a specific Task object from the database.

In line #2, we create an empty list using the phrase "[ ]".

In line #3, we make a list of the Task's Assignment objects using the phrase "task.GetList('Ássignment')" and then place each Assignment object, one at a time, into the variable "assignment".

In line #4, check to make sure that the assignment refers to a valid resource, if not skip it.

In line #5, we find the Resource object for each assignment and add it to the Resource list.

Some people find the first version easier to understand. Some prefer the second. It is important to become comfortable with both. Each has its advantages and you will find both in example scripts.

Object Quick Reference

These first few examples have been complete scripts. Most of the other examples will be small excerpts from scripts to illustrate a particular technique.

Using Objects

Purpose	Sample Code
Retrieve an existing object from the database. (The active database object is located at Data.DBObject.)	`project = Data.DBObject.GetObject('Project', 1)`
Retrieve all of an existing object type from the database. (Retrieves objects from every project.)	`projectlist = Data.DBObject.GetList('Project')`
Retrieve data from an object.	`name = task.Name`
Retrieve an object reference from an object.	`project = task.Project`
Retrieve all objects that point to an object.	`tasklist = project.GetList('Task')`
Retrieve all objects that point to an object when the pointer is not the object's table name (i.e. when the pointer is a table alias).	`prereqlist = task.GetList('Task','Prerequisite')`
Update data in an object.	`task.Name = 'New Task Name'`

Update an object reference in an object.	`task.Project = project3`
Clear an object reference. Note the 'ID' that must be added to the object reference name.	`object.Object2ID = None`

Create a new object. Whenever you create a new object, be sure to provide all of the required values. For example, all new objects should have "Project" set.	`task = Data.DBObject.GetObject('Task')`
	`task.Name = 'New Task Name'`
	`task.Project = project`
	`task.DateAdded = Data.TodayString()`
	`Data.SetUndo('Add new Task)`

Some objects use a special object reference, "Target". The "target" may be of any object type.	`object2 = object1.Target`
To set a target, use the same syntax as for other object references.	`object1.Target = object2`
To clear a target, this special syntax must be used.	`object1.TableName = None` `object1.TableID = None`

Working with Reports

Purpose	Sample Code
Retrieve the object for report #2.	`report = Data.DBObject.GetObject('Report', 2)`

Retrieve the object of the currently open report. (When a script starts, the variable 'self' contains a reference to the current report window. 'self.ReportID' is the ID number of the current report. 'self.Report is the report object.)	`report = self.Report`
	
	`or`
	
	`report = Data.DBObject.GetObject('Report',self.ReportID)`

Retrieve all of the reports in a project. (Assumes that 'project' contains the project object.)	`reportList = project.GetList('Report')`

Make a list of only Task reports in a project.	`reportList = project.GetList('Report')`
	`taskReportList = [ report for report in reportList`
	`  if report.ReportType`
	`  and report.ReportType.Name = 'Task' ]`

Working with Tasks

Purpose	Sample Code
Find the project that contains the current report.	`project = self.Report.Project`
Retrieve a list of all tasks in a	`taskList = project.GetList('Task')`

project.	
Retrieve all of the projects in a database.	`projectList = Data.DBObject.GetList('Project')`

Make a list of all unfinished tasks in a project.
(Task completion in GanttPV can be indicated by either an Actual End Date or by a Percent Complete of 100.)
	`taskList = project.GetList('Task')`
	`unfinishedTaskList = [ task for task in taskList`
	`  if not task.ActualEndDate`
	`  and (not task.PercentComplete`
	`  or task.PercentComplete < 100) ]`

List all direct prerequisites for a task.
(The clause 'if depend.Prerequisite != None' will filter out dependencies on inactive tasks.)
	`dependencyList = task.GetList('Dependency')`
	`prerequisiteList = [`
	`  depend.Prequisite for depend in dependencyList`
	`  if depend.Prerequisite != None ]`

List all direct successors for a task.	`dependencyList = task.GetList(`
	`  'Dependency','Prerequisite')`
	`successorList = [ depend.Task for depend in`
	`  dependencyList if depend.Task != None ]`

Import Scripts

The second kind of script we will cover is a data import script. Because of the effort required to write a script, it is sometimes better to copy and paste data into GanttPV or convert the data to match the format that an existing import script can handle. Writing an import script is preferred if the quantity of data is large, the format doesn't match any other import scripts, and the import will be required many times.

Import scripts are little harder to write than export scripts primarily because certain information must be present in order for GanttPV to work properly. For example, the project should always be imported before its tasks and each task must be assigned back to its project. Because import scripts are more difficult to get right, it is recommended that you always make a backup copy of your database before running an import script.

In order to import information into GanttPV you must have a good understanding of how the data is organized. Be sure to review the data model information at the end of this document.

Small Import Script Example

Here is example script that produces import a list of Tasks into GanttPV. This script assumes that the file contains a line for each task and that each line contains the task's name and start date separated by a tab character. The data must be in GanttPV's date format, for example: 2008-01-31 would be for January 31, 2008.

```
1.  project = Data.DBObject.GetObject('Project')
2.  project.Name = 'New Project'
3.  taskData  = open('tasks.txt', 'r')
```

```
4.  for taskData  in input:
5.      fields = taskData.trim().split('\t')
6.      task = Data.DBObject.GetObject('Task')
7.      task.Project = project
8.      task.Name = fields[0]
9.      task.StartDate = fields[1]
10. Data.SetUndo("Import Tasks")
```

Line #1 creates a new Project object.

Line #2 sets the name of the new Project.

Line #3 opens a file that describes the new tasks to be added.

Line #4 reads each line of the input file and places it into the variable name "taskData".

Line #5 trims off any extra leading or trailing white space and then divides the task data by tab characters.

Line #6 creates a new Task object.

Lines #7, 8, and 9 set the Task's project, name, and start date.

Line #10 tells GanttPV that all of the preceding changes should be undoable as a group and sets the message that will be displayed next to the undo command in the menu. It is important to use the command "Data.SetUndo" command after any changes to the database.

After importing the Tasks into GanttPV you will need to add a "Task Report" to display them.

Larger Import Script Example

Here is a more complete example of an import script. Most import scripts will be similar in many ways to this one. This is an updated version of the "Import PMGame File" script that comes with GanttPV. It has been changed to use the new object model. While the script may seem complicated at first, it can be understood if it is examined it one piece at a time.

The first step in creating an import script is to carefully examine the format of the input data. The example file contains five types of record identified by the first word: Format, Skill, Project, Task, and Resource. After that first word the format of each record is different. Each record contains additional fields that are separated by tabs.

The "Format" record contains only one additional field, the file format version number. In this case the format is "1" which means that this is the first (and only format) for PMGame import files. For our purposes we will ignore "Format" records.

The "Skill" record gives an ID number and name for each skill required for this project. In this example, all tasks require the same skill and all of the resources have that skill. Because skill does not impact scheduling for this project, we will ignore the "Skill" format records, too.

The “Project” record contains two additional fields: a project ID number and a project name. We need to assign a new project ID number in GanttPV, but we will also save the file’s project ID number for later reference. The project name will be retained in GanttPV.

The “Task” record is the most complicated. It contains five fields: a task ID number (which should be saved for reference), a task name, the estimated number of effort hours required to complete the task, the ID numbers of the prerequisite tasks, and the ID numbers of the skills required to perform the task. The prerequisite task ID numbers are separated by commas. We will need to convert these ID numbers to the GanttPV equivalent. Also, in this file format, the prerequisite task numbers always refer to tasks that appear earlier in the file. This simplifies matters because we will not have to deal with prerequisite ID numbers that we haven’t already seen. All of the tasks for a particular project appear after the project record. We will have to remember the most recent project record in order to assign the task to the correct project.

Finally, the “Resource” record contains three fields: the resource ID number, the resource name, and the resource’s skills. For GanttPV, we will assign a new resource ID, keep the same name, and ignore the skill.

```
1.  Format   1
2.  Skill    1    nu
3.  Project  1    Grey
4.  Task     1    Interview Quaryman            6                    1
5.  Task     2    Interview Gonzalez            5                    1
6.  Task     3    Use Case 1                    3                    1
7.  Task     4    Documentation Part 1          11                   1
8.  Task     5    Verify Case 1                 5    1,4             1
9.  Task     6    Verify Case 2                 3                    1
10. Task     7    Design Web Page 1             7    2,3             1
11. Task     8    Design Architecture Aspect 1  9    3,4,5           1
12. Task     9    Design Interface 1            7    1,3,5           1
13. Task     10   Define Methods Set 1          8    3,8,9           1
14. Task     11   Define Methods Set 2          8    7,10            1
15. Task     12   Define Methods Set 3          4    5,6,9           1
16. Task     13   Define Methods Set 4          9    2,8,10,12       1
17. Task     14   Define Methods Set 5          9    2,5,8,9,10,11   1
18. Task     15   Write Unit Test Set 1         7    1,12            1
19. Task     16   Code Methods Group 1          8    5,10,15         1
20. Task     17   Code Methods Group 2          12   2,3,4,9,15,16   1
21. Task     18   Code Methods Group 3          13   1,6,10,11,13,15 1
22. Task     19   Integration Task 1            6    4,11,14,17      1
23. Task     20   Integration Task 2            16   7,13,18         1
24.
25. Format   1
26. Skill    1    nu
27. Resource 5    Alice                         1
28. Resource 6    Bob                           1
29. Resource 7    Charles                       1
30. Resource 8    Dana                          1
31. Resource 9    Eric                          1
32. Resource 10   Francis                       1
33. Resource 11   Geoff                         1
34. Resource 12   Hanna                         1
```

```
35.  Resource 13   Iris                            1
36.  Resource 14   John                            1
37.  Resource 1    PM
```

The script comes in three parts. Two parts begin with word “def”. As we have seen, these “def” parts define functions that consist of a series of instructions that can be used later by referring to their names. The third part is a one line command to use one of the functions. A full explanation of the script is given at the end.

Lines #1 to #39 define the function named “DoImport”. This function does most the actual work of the import. When the function is used it must be given the path name of the file that contains the data to be imported.

Lines #41 to #55 define the OpenDialog function that creates a file dialog box that will be used to select the file to be imported.

Line #57 tells the GanttPV to use the OpenDialog function. Line #53 tells GanttPV to use the DoImport function.

```
1.   def DoImport(path):
2.       db = Data.DBObject
3.       taskids = {}  # convert original ID numbers to GanttPV ID numbers
4.
5.       infile = open(path, 'r')
6.       for line in infile:
7.           line = line.strip()  # removes leading or trailing spaces
8.           fields = line.split('\t')
9.           if fields[0] == 'Format':  # ignore
10.              pass
11.          elif fields[0] == 'Project':
12.              project = db.GetObject('Project')  # create a new project
13.              project.PMGameXref = int(fields[1])
14.              project.Name = fields[2]
15.          elif fields[0] == 'Task':
16.              task = db.GetObject('Task')  # create a new task
17.              task.Project = project
18.              task.PMGameXref = int(fields[1])
19.              task.Name = fields[2]
20.              task.EffortHours = int(fields[3])
21.
22.              taskids[task.PMGameXref] = task  # convert prerequisite
23.
24.              if len(fields) > 4:
25.                  preids = fields[4].split(',')
26.                  for prerequisite in preids:
27.                      if prerequisite:
28.                          prereq = db.GetObject('Dependency')
29.                          prereq.Prerequisite =
30.                              taskids[int(prerequisite.strip())]
```

```
31.                         prereq.Task = task
32.         elif fields[0] == 'Resource':
33.             resource = db.GetObject('Resource')  # create new resource
34.             resource.PMGameXref = int(fields[1])
35.             resource.Name = fields[2]
36.         else:
37.             print fields  # ignore other lines
38.
39.     Data.SetUndo("Import PM Game File")
40.
41. def OpenDialog():
42.
43.     wildcardx = "TXT (*.txt)|*.txt|" "All files (*.*)|*.*"
44.     dlg = wx.FileDialog(
45.         self, message="Open PM Game Project File ...",
46.         defaultFile="", wildcard=wildcardx,
47.         style = wx.OPEN
48.         )
49.
50.     dlg.SetFilterIndex(0)
51.     if dlg.ShowModal() == wx.ID_OK:
52.         path = dlg.GetPath()
53.         DoImport(path)
54.
55.     dlg.Destroy()
56.
57. OpenDialog()
```

Line #2 saves the database object into variable “db”. This essentially creates an abbreviated reference to the database. We can say “db” instead of “Data.DBObject”.

GanttPV creates a new ID number for each Task that is imported. This new number will be different than the ID number that the Task had in the source application. Because the import file specifies prerequisite tasks using the ID from the source application, they must be converted to GanttPV ID numbers. Note that in this input file format all the prerequisite tasks are listed before the successor tasks. Line #3 defines a dictionary that will be used to perform this translation.

Line #5 opens the input file. The “r” specifies that the will be read, not written over.

Line #6 reads each line of the input file. Lines #7 and #8 divide the line into separate fields.

Line #9 and #10 identify and then ignore any “Format” lines. “pass” means “do nothing”.

Line #11 and #12 say that if the line begins with “Project” then create a new project.

Line #13 saves the ID number from the source application. Note that this field is not part of the data model. In GanttPV new information can be added to data model simply by specifying its name like this.

The cross reference number is a series of digits in the input file. The “int” command tells GanttPV to treat the digits as a number.

Line #14 saves the project’s name. Note that the first field is referenced by “fields[0]”. It is common for computer languages to start counting from zero.

Line #15 and #16 create a new Task. Line #17 says the Task should be part of the most recently started project.

Lines #18 to #20 save the task’s original ID number (from the source application), name, and effort hours.

Line #22 sets up the cross reference for this task. It says: “whenever I give you the old ID number, return the new Task object.”

Line #24 tests whether prerequisites are specified for the current task. In the file format, prerequisite task ID numbers are separated by commas, for example: “1,3” would specify that tasks #1 and #3 are prerequisite tasks.

Lines #26 takes each prerequisite number and puts it into the variable “prerequisite”.

Line #27 makes sure that a prerequisite number was specified. If input file incorrectly specified the prerequisites as “1,,3”, this line would skip the middle missing prerequisite.

Line #28 creates a new dependency. Line #29 uses the cross reference to find the GanttPV task that corresponds to the prerequisite. Lines #30 and #31 say that the current task is the successor in this dependency.

Lines #32 to #35 import the Resource into GanttPV.

Lines #36 and #37 display and then ignore any other line types.

Line #39 sets up the undo command for the import.

Line#43 identifies which types of files should be visible for selection in the file dialog.

Lines #44 to 50 define the dialog box, give it a title, and specify that it will be used to open a file.

Line #51 displays the dialog.

Lines #52 to #53 determine whether a file was selected or the dialog’s cancel button was pressed. If a file was selected the file’s path name is identified, the “DoImport” function is used to process the file.

Line #55 tells GanttPV that the dialog box isn’t needed anymore.

Install Scripts

The third kind of script we will cover is the install script. Install scripts can be used to define new report types and column types. Be sure to include all of the required information in your install scripts.

Report Types

This is part of the install script that adds the FollowUp item report. Each report type and column type is defined by giving a series of keywords and values. For example, here is the definition for the "FollowUp" report type:

```
1.        rt = {
2.             'Name': 'FollowUp',
3.             'Label': 'Follow Up',
4.             'TableA': 'FollowUp',
5.             'TableB': None,
6.             'Also': None,
7.             'AllOrEach': 'both',
8.             'SuggestedColumns': ',Description;,DateAdded;'
9.             }
```

The report type is defined as a Python "dictionary". The dictionary definition starts with a curly brace ("{") in line #1 and ends with the closing brace in line #9. The format for each entry in the dictionary is a key followed by a value. The values can be a string in quotes, a number, or the word "None". The entries are separated by a comma.

In line #2 the key is "Name" and the value is "FollowUp". This will be the name of the report type. Every report type must have a name. No two report types can share the same name.

Line #3 specifies the "Label". The label is the external name of the report type. It is used in GanttPV menus. The label may be translated to other languages. The label is optional. If omitted, the name is used as the label.

Lines #4 and #5 specify the Object types that are displayed in the report. GanttPV reports can be either "one table" or "two table" reports. "TableA" is required. It can be either an existing object type or a new one. In this example, a new object type is defined named "FollowUp". "TableB" specifies a secondary table or the value "None". If a "TableB" is specified it is defines the child rows. For example, in the "Task/Assignment" report the primary table is defined as "Task" and the secondary table is "Assignment". All tasks would be included in the report. Under each task would be listed all of the Assignments that contain a "TaskID" value that is equal to the task's ID number.

Line #6 shows the "Also" keyword. It is used in a two table report to "copy" the column types from another report one table report. For example, it is used in the Task/Assignment report to copy all of the column type definitions from the "Task" report. The value specified is either the report type name or report type ID number (if known) of the other report.

Line #7 shows the "AllOrEach" keyword. This controls whether reports of this type can be added only to project one (the "All Projects" project), only to other individual projects, or to both kinds. The allowed values are "all", "each", and "both".

Line #8 specifies “SuggestedColumns”. These are the columns that are automatically added to new reports of this type when they are first created. Column definitions are separated by semi colons. Parts of each definition are separated by commas. The format is: table (defaulting to the current table), column type name, column width in pixels (defaults to the column type’s definition), and the number of periods for time scale columns.

Column Types

This is part of the install script that adds the “Description” column type to the report type.

```
1.      { 'Name': 'Description',
2.          'Label': None,
3.          'DataType': 't',
4.          'AccessType': 'd',
5.          'T': 'A',
6.          'Edit': True,
7.          'Width': 180  },
```

Line #1 defines the column type name. The name must be unique within a report type.

Line #2 defines the label or is “None”. If included, the label is displayed instead of the name in report column headings and in menus.

Line #3 controls the type of data that may be entered into the report. Allowed values are: “t” for text, “d” for dates, “I” for integers, “f” for numbers with a decimal place.

Lines #4 and #5 explains where the column will look to find the data to display. “d” stands for “direct”. It will retrieve a value named “Description” (as specified in line #1) from either “TableA” or “TableB” depending on the value of the “T” entry (in line #5). Other possible values will be described later.

Line #6 controls whether the value can be edited in the report or not. “True” means that the value can be edited. “False” means that it cannot.

Line #7 specifies the initial width of columns of this type.

Indirect Access Column Type

Here is an example with a different access type. In this case the access type is “I” which stands for “indirect”. In this example, the column will look in “TableA” (line #5) for the value of “ProjectID” (line #1, the first part of the name with “ID” added). It will find the “Project” (line #1, the first part of the name) with that ID number and then will return that project’s Name (line #1, the last part of the name). The “Edit” value is set to “False” to prevent the user from changing the Project Name through this report.

```
1.      { 'Name': 'Project/Name',
2.          'Label': 'Project\nName',
3.          'DataType': 't',
4.          'AccessType': 'i',
5.          'T': 'A',
```

```
          'Edit': False,
          'Width': 100  },
```

Complete Example

Here is the complete install script:

```
def DoAdd():
      rt = {
          'Name': 'FollowUp',
          'Label': 'Follow Up',
          'TableA': 'FollowUp',
          'TableB': None,
          'Also': None,
          'AllOrEach': 'both',
          'SuggestedColumns': ',Description;,DateAdded; '
          }
      ct = [
      { 'Name': 'ProjectID',
        'Label': 'Project\nID',
        'DataType': 'i',
        'AccessType': 'd',
        'T': 'A',
        'Edit': True,
        'Width': 50  },
      { 'Name': 'Project/Name',
        'Label': 'Project\nName',
        'DataType': 't',
        'AccessType': 'i',
        'T': 'A',
        'Edit': False,
        'Width': 100  },
      { 'Name': 'ID',
        'Label': None,
        'DataType': 'i',
        'AccessType': 'd',
        'T': 'A',
        'Edit': False,
        'Width': 35  },
      { 'Name': 'Description',
        'Label': None,
        'DataType': 't',
        'AccessType': 'd',
        'T': 'A',
        'Edit': True,
        'Width': 180  },
      { 'Name': 'DateAdded',
        'Label': 'Date\nAdded',
        'DataType': 'd',
```

```
43.         'AccessType': 'd',
44.         'T': 'A',
45.         'Edit': True,
46.         'Width': 80  },
47.       ]
48.       Data.AddReportType(rt, ct)
49.
50.       Data.SetUndo("Install Follow Up Tracking")
51.
52.  DoAdd()
```

In the above example, line #48 takes the definitions and applies them. Line #50, sets the description that will appear in the "Undo" menu.

Time Scale Column Types

In order to add column types to a report type that you are certain already exists, only the report type name is required.

```
1.  def DoAdd():
2.      rt = { 'Name': 'Task' }
3.      ct = [
4.      { 'Name': 'Week/Gantt',
5.        'Label': None,
6.        'DataType': 'g',
7.        'AccessType': 's',
8.        'T': 'X',
9.        'Edit': False,
10.       'Width': None  },
11.         ]
12.     Data.AddReportType(rt, ct)
13.     Data.SetUndo("Install Week Gantt Time Scale")
14.
15. DoAdd()
```

For gantt chart column types four values need to be set. Note the "Name" in line #4. It has two parts. The first part is the time frame. It can be "Day", "Week", "Month", "Quarter", or "Year". The second part is"Gantt" for "gantt chart". In line #6, the data type is "g" which also stands for "gantt chart". Line #8 show that for time scales the value of "T" should be "X". In line #9, the "Edit" value should be "False".

Here is another time scale example. This one displays effort hours by day.

```
1.  { 'Name': 'Day/EffortHours',
2.    'Label': 'Day/Effort',
3.    'DataType': 'i',
4.    'AccessType': 's',
5.    'T': 'X',
```

```
6.    'Edit': True,
7.    'Width': 40   },
```

Three keyword values must be set for this to work. In line #1, the name has two parts. The first part is the time period. The second part is the name of the value that should be displayed, “EffortHours”. The “AccessType” (in line #4) must be set as “s” for “time Scale”. Finally, “T” must be “X”, in line #5.

Python and GanttPV Modules

The fourth kind of script we will cover uses Python or GanttPV modules . The library modules are a major strength of Python. In the GanttPV distributions for Windows and Macintosh only a subset of the library modules are available for import by scripts. Some modules are sure to be available because they are imported by one of the GanttPV modules. Others modules may also be included be default. When in doubt, test the available of modules by running a script that imports the module of interest. Here is a list of the modules that are imported by one or more of the GanttPV modules. Documentation on these modules is available at: http://docs.python.org/library/

1. `calendar`
2. `cPickle`
3. `datetime`
4. `gettext`
5. `math`
6. `os`
7. `os.path`
8. `random`
9. `re`
10. `select`
11. `socket`
12. `string`
13. `struct`
14. `sys`
15. `threading`
16. `time`
17. `traceback`
18. `webbrowser`
19. `wx`
20. `wx.grid`
21. `wx.html`
22. `wx.lib.dialogs`
23. `wx.lib.ogl as ogl`
24. `xml.dom.minidom`
25. `xmlrpclib`
26. `Zeroconf`

In addition, the following GanttPV modules are available to be imported by scripts:

1. `Data`

2. GanttReport
3. ID
4. Menu
5. ORM
6. ORMReport
7. QuickStart
8. ReportAids
9. StartupData
10. UI

The following variables are set up before scripts run:

1. 'Data', 'ID' # these module names are already imported
2. 'self' # points to the currently active report window
3. 'thisfile' # contains the script's complete path name
4. 'scriptname' # contains the script's file name
5. 'debug' # contains the value 1

GanttPV Data Model

Data Model

The data in GanttPV is organized as tables with rows and columns. It is similar to the organization of tables in a relational database. GanttPV includes a number of built in tables and reports. More can be added dynamically using “install scripts”. Tables can also be added as needed using the command “Data. AddTable(tablename)” in Python scripts.

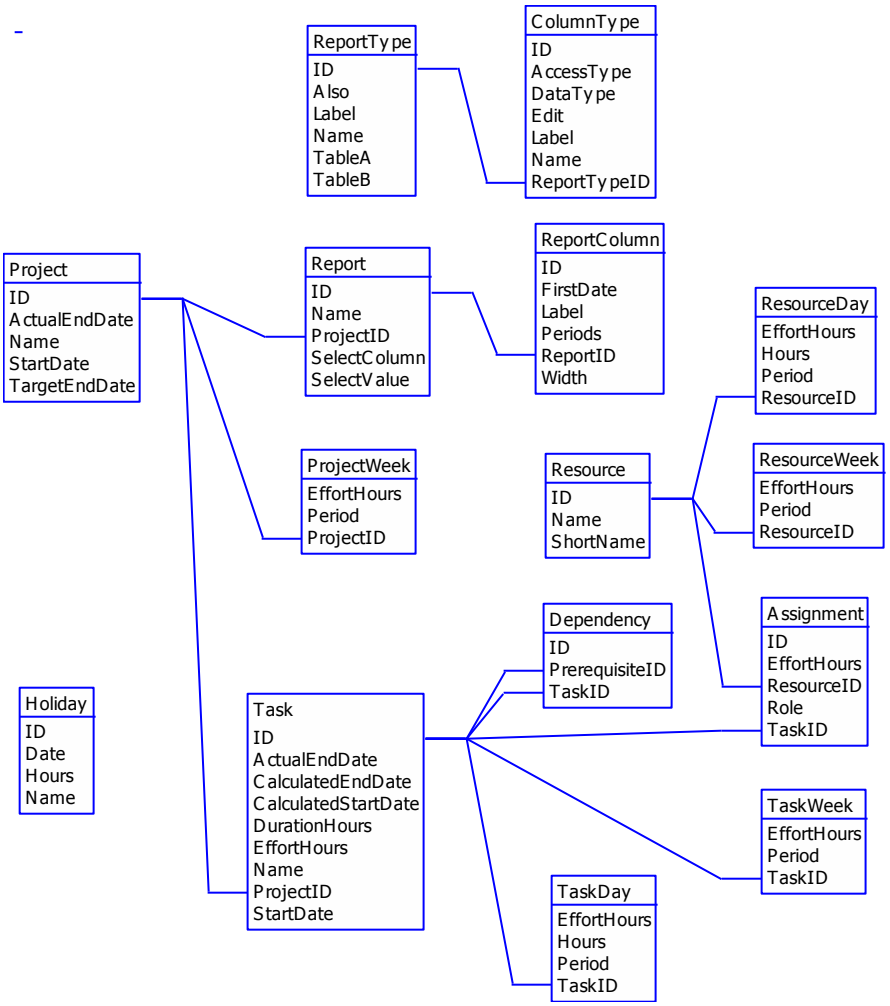
Object Model

An object oriented façade is provided for ease of programming. Each table is represented by a separate object class. Rows are objects. Columns are object data members. Objects are requested via an object factory that can be accessed at “Data.DBObject.GetObject(objecttype, [objected])” to get individual objects or “Data.DBObject.GetList(objecttype)” to retrieve all instances of a particular object type.

The object factory maintains a cache of all of the objects it has returned in order to make sure that a repeated request will return the same object, not just an equivalent object.

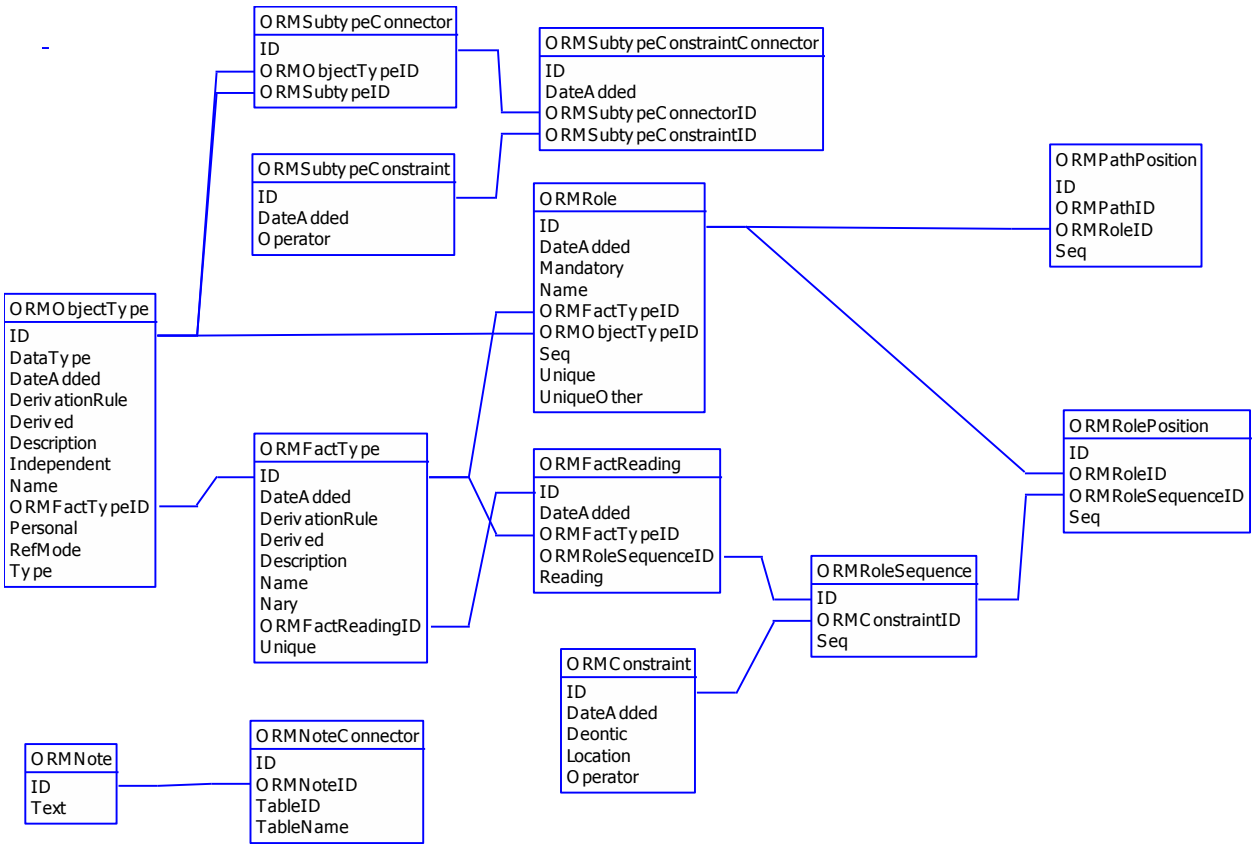
Default Objects

When a new GanttPV database is created, the object model looks like the following. More objects and data are added to the model when install scripts are used.



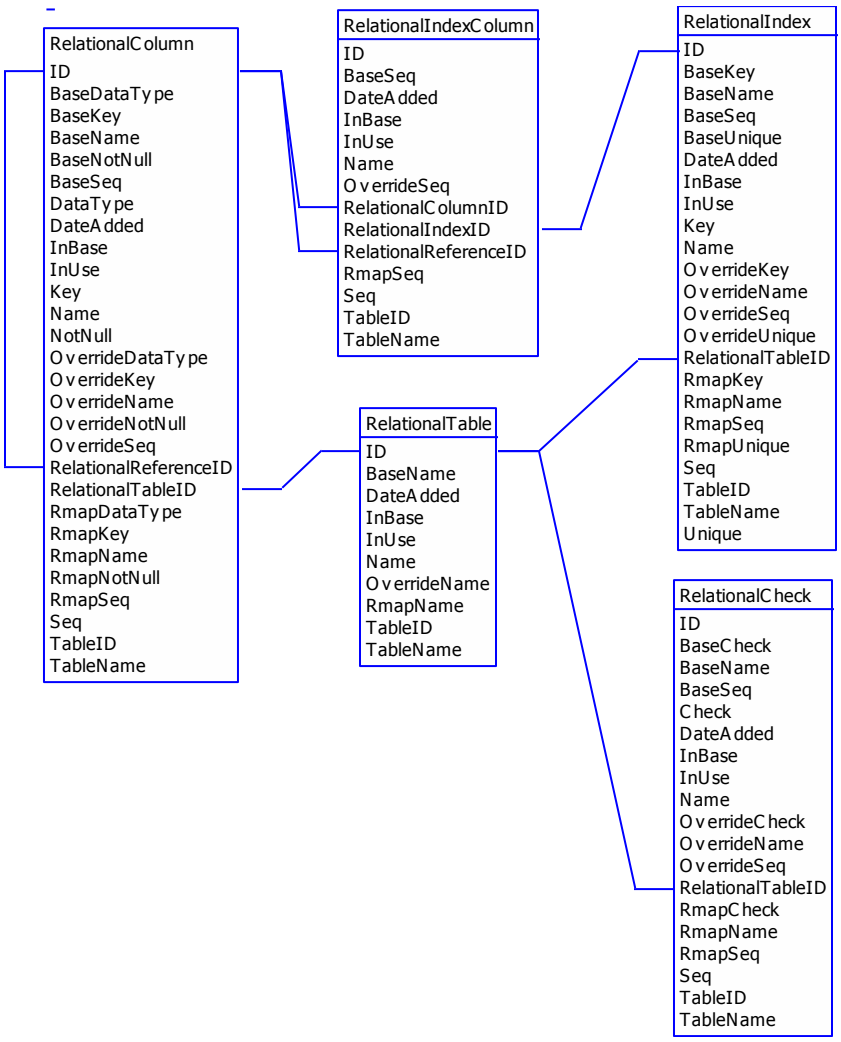
ORM Diagram Data Model

The data model for the ORM Lite information modeling add-in defines the following objects. In addition to the references shown in the diagram, all of the objects also contain a reference to ProgramID.



Relational Tables Data Model

When using Rmap the following objects are added to the model. As of this writing, Rmap creates only two of these tables, Relational Table and Relational Column. The remaining tables will be created to implement additional constraints that can be defined in ORM Lite. The tables are the basis for the SQL generation.



To Do List for this Book

To the reader:

> We want to make sure that this book is useful to you and answers the questions that you have about GanttPV. What sort of things would you like to see in a book on GanttPV? Email us at brian@simpleprojectmanagement.com

There should be reference section that explains all of the menu commands and scripts. [Complete for the commands; still need to do the scripts.]

There should be a section on programming Ganttpv. This section should explain the object model, the programming API, how to read information from a GanttPV database and how to add objects to the GanttPV database. [Still more to do.]

There should be something on how to use the to-do list feature. [Not done yet.]

And something on creating schedules. [Partially done.] Using the color coding. [Not yet.] Using the resource leveling. [Not yet.] Calculating earned value. [Not yet.] Calculating resource work load. [Not yet.]

How to use the GanttPV server? [Not yet.]

Need to give examples of how GanttPV can be used to solve specific problems. [Still needed, what would be most useful?]