4D Job Wallet

Version 1.0

4D Job Wallet
for 4D Developers

Documentation Revision Nº: 1.0.0

Acknowledgements

4D (previously "4th Dimension") has been and continues to be one of the more outstanding innovations of the late 20th century software revolution, uniquely liberating the dreams of thousands of its customers in a way that competing platforms could only do with a mountain of venture capital and an army of programmers. It is one of the tiny number of endeavours of that era which took up the challenge of doing justice to the term "4th Generation Language" and emerged as both a conceptual and commercial success.

Ultimately, the fortunes of such a venture are dictated by an equally visionary customer base which identifies distinct opportunities higher up the risk/reward ladder than its competitors in their own commercial sectors.

This project owes its origins to those customers of 4D who have seen fit to place their impassioned technological aspirations in my hands and those of my industry peers, whose requirements this component attempts to address.

Finally, the following parties made valued contributions to this product's completion, variously directly....

- **The Cancer Research UK Formulation Unit** *at the* **University of Strathclyde** *in* **Glasgow** *who sponsored the 'Snapshot' feature*
- **Ortwin Zilgen***, who provided significant and authoratitive critical appraisals during the final production phase resulting in several revisions to the functional presentation*
- **Peter Schumaker***, whose experience and wisdom in any quantity is of immeasurable value to a project such as this*

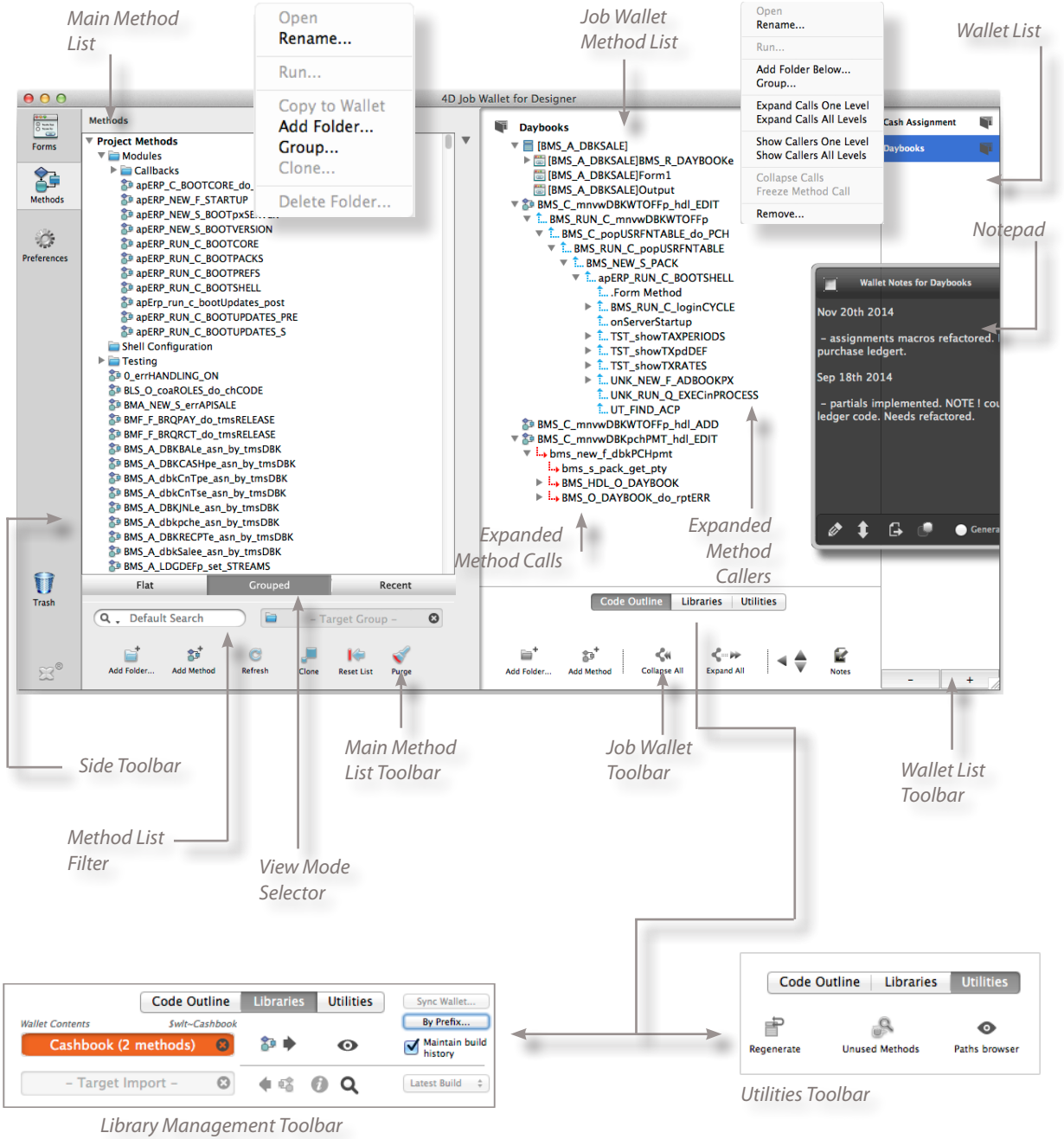
...and indirectly:

- **Keisuke Miyako** *on whose advanced programming tutorials some features of this product are based. His prolific contributions to 4D life form a rich resource that establishes best practice approaches to many difficult coding scenarios*
- **Nicholas Butler** *of nickbutler.net who's .NET interpretation of Myer's Greedy Diff algorithm formed the basis for my 4D implementation*

It goes without saying that any shortfalls in adequacy are mine, not theirs.

Peter Jakobsson
Creator, 4D Job Wallet

Terminology



Contents

1 What is 4D Job Wallet ?

1.1 Going from 1 to 2 Architectural Layers

When 4D introduced its first component architecture in version 6.7 and substantially enhanced it with the v11 rewrite, it set in motion a new phase of modular development opportunities for business applications. At the same time, it diversified the code base constituting a 4D application. We now had 2 distinct layers to consider and maintain:

- *the component layer - well delineated, throuroughly re-useable code units supported by a flotilla of memory management, portability, error handling and process management features*
- *the host layer - by contrast having limited support for code reusability, library management and general complexity in general*

4D Job Wallet was created to address some of those challenges in host layer.

It provides an environment in which to break down a complex application into manageable units to optimise code development, auditability and portability. In particular, 4D Job Wallet lets you organise your structure objects as "mini mind maps" which allow you to quickly re-focus on a particular task while at the same time keeping all the relevant methods handy for immediate access.

Additionally, the component comes with a bundle of utility features which further support the code management process and together represent a powerful aid to development. A single method call invokes the unified editor who's highlights include:

Primary Features

- *an* **enhanced 4D method browser** *supporting project, form and object methods, hierarchically grouped views, flat views, filtering by method name or date, and "Find in Design" type content searches*
- *the* **new wallet environment** *which lets you create focused, hierarchical method groups, inspect calls and callers and generally "mind-map" specific parts of your application as you see fit*

Supporting Features

- *a ground breaking new* **text based code archiver** *which supports basic but powerful version control, user-friendly module delineation via the wallet metaphor and fine-grained* **native diffing** *support for imports*
- *a module* **cloning tool** *which accelerates productivity by letting you duplicate methods and method groups while automatically mapping old name segments to new ones*
- *a* **coding notes** *feature which lets you create wallet-specific or application general code annotations*
- *an* **unused methods** *inspector which lets you keep track of un-called methods*
- *a* **missing tables** *recovery feature to lever the datafile's ability to migrate table definitions between code forks*
- *a* **paths browser** *which presents instant access to all the key 4D and 4D Job Wallet folders in your desktop environment*
- *a* **structure snapshot** *feature which supports wholescale dumping of every structure object's meta data including table definitions, form object properties method code for parsing with 3rd party diffing tools*

2 Getting up and Running

Most of the functions in 4D Job Wallet are intuitively obvious in their behaviour. This short section will get you started using the basic feature sets of the method browser and wallet. An in depth discussion of all tools is provided in later sections.

2.1 Installing

4D Job Wallet is supplied as a cross platform component in ".4dbase" form. Simply drop this package into your "Components" folder for each structure that requires it. Alternatively you can store the component in a common location, create aliases and drop the aliases into each structure's component folder.

2.2 Invoking 4D Job Wallet

To bring up the 4D Job Wallet editor, simply execute the method **4wallet_doLaunch** from the Run menu in 4D's design environment.

[Tip: A quick way to invoke this command is to use the keyboard shortcut for Run -> Method and then simply type a 4 on the keyboard followed by Enter.]

4D Job Wallet will parse your structure the first time it is launched. (Parsing will be much quicker on subsequent launches as much of the parse is cached between restarts).

2.3 Using 4D Job Wallet

The method browser works largely as in 4D's native explorer but with some additional functions. Use the view-mode selectors at the bottom of the browser to switch between flat, grouped and recent views. Recent is simply a flat view sorted in date-time modified order with the latest methods at the top.

2.4 Grouping Methods

Most of the interactive method browser functions can be found by right-clicking on

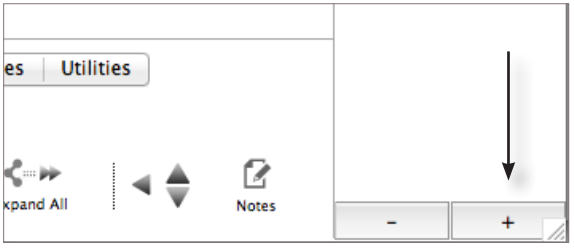
the method list to display the contextual menu. There are 3 ways to "group" methods:

1. highlight the methods you want to group and choose the **Group** item from the right-click menu
2. highlight a group of methods and drag them into a folder
3. drag a folder down to the **Target Group** well and then make successive additions to the group by dragging methods from the list onto the **Target Group** well

2.5 Creating a Wallet

You can create any number of job wallets and the same methods can appear in more than 1 wallet. However, methods cannot be aliased within a given wallet - they must be unique. To create and populate a job wallet follow these steps:

1. flip the wallet pane open by clicking the triangle button at the top right hand corner
2. click the + button at the bottom of the wallet list (far right column) and save the new wallet name
3. drag methods and folders from the main list into the job wallet pane
4. use the job wallet contextual (right-click) menu to perform interactive wallet functions



2.6 Expanding Dependencies

Once methods have been dragged into the wallet, use **alt-click** to display a method's calls and **ctrl-click** to explore its callers.

[NOTE:] On **Windows**, *the shortcut for callers is* **alt-ctrl-click** *to avoid confilcting with the multi-row highlight action*

3 Method Browser Reference

3.1 Using the 3-View Modes

When viewing project methods, the list can be configured in one of 3 view modes. The view mode is set using the bevel controls at the bottom of the method list. Method filtering is supported by all 3 modes who's operation is described in detail below:

- **Flat** *-The methods are listed in alphabetical order without their group structures. This mode corresponds to 4D's native method explorer. You can open methods, copy methods to the wallet and generally perform most functions in this mode, however grouping is not available*
- **Grouped** *this is the default browsing mode and displays the methods in their associated folder groups. Use this mode when adding or removing folders or with the* **Group** *command from the contextual menu*
- **Recent** *in this view, the method list is displayed exactly as described in Flat but the content is sorted in reverse chronological order - use this view to bring the most recent methods you've worked on to the top of the list*

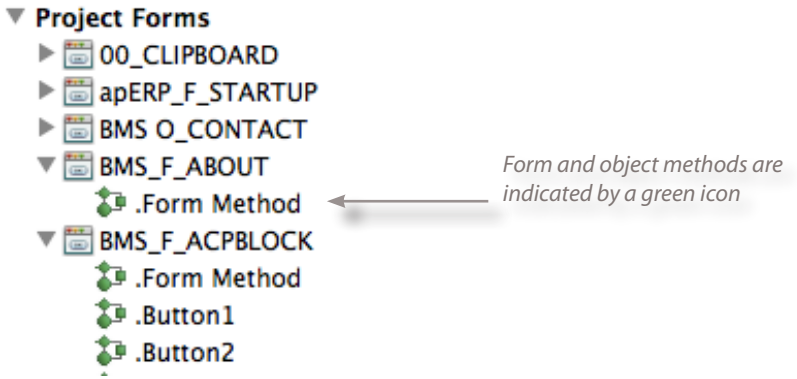
Main method list view modes

3.2 Form and Object Methods

On selecting the **Forms** option in the side toolbar, project and table forms are displayed along with thir associated object methods. Project forms and table forms are grouped separately but in each case the form method appears first followed by object methods listed by object name. Due to limitations in 4D's component command scope, the form itself cannot be opened from 4D Job Wallet - you have to use 4D's native explorer for that. However 4D Job Wallet gives you direct access to all the form's object methods without having to open the form first.

Form and object methods are indicated by a distinct green icon both in the main

method browser and in the wallet list. This helps you to distinguish them from project methods when a wallet contains a mix of both types.



[TIP: In the Forms option, the 3 view modes have no effect but you can still use the filter control]

Opening a Method for Editing

You can access methods from 4D Job Wallet in exactly the same way as in the native 4D method editor. To open a method in 4D's method editor either double click on it or choose the **Open** command from the contextual menu with the method highlighted.

3.3 Using Filters in the Method List

4D Job Wallet supports enhanced method browsing by integrating many of the features found in 4D's "Find in Design" function straight into the basic method list. The filter control located just below the main method list is activated by entering a value and hitting the Tab key.

[TIP: The filter control displays with a yellow background whenever a non-default filter mode is selected. To change the filter mode, choose from the drop down menu by clicking the small triangle inside the control]

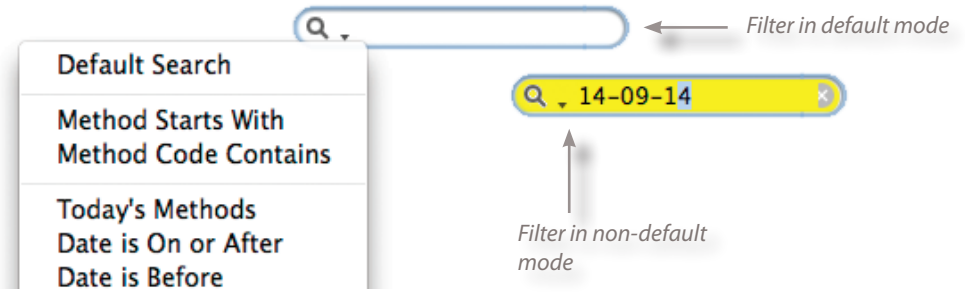
You can choose from the following filter modes:

String Filters

- **Default Filter** *does a "contains" type filter on the method name*
- **Method Starts With** *displays items starting with the specified string*
- **Method Contains** *in this mode, the method list will display actual method content which matches the filter criteria (described in more detail below)*

Date Filters

- **Today's Methods** *filters methods which have a "Last Modified" date corresponding to the today's date*
- **On or After, Before**, **Is On** *perform a filter on the last modified date relative to the specified date*



[TIP: In the 4D Job Wallet preferences, you can specify whether folders are to be included or excluded from filter results. To include folders, check the Folders are hits in method list option]

3.4 Searching on Method Content

4D Job Wallet further integrates features from "Find in Design" by optionally displaying actual method content directly in the method browser. Method content searches are supported both by project method and form/object method lists. To invoke method content searching, follow these steps:

1. choose the **Grouped** view mode (you can use **Flat View** or **Recent**, but only the method names will be displayed - content display will be disabled)
2. select the **Method Code Contains** option from the filter sub-menu
3. enter a search string in the filter control and hit Tab

4D Job Wallet will automatically configure the method browser to display method content which matches the filter string equivalent to executing "Text" "Contains" in "Find In Design".

3.3 Contextual Menu

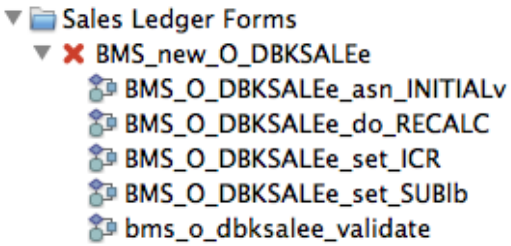
Many of the functions applicable in the method list can be accessed via the Right-Click menu in the main method list. They are enabled according to the highlighted item and described in detail later.

3.5 Detection of Renamed or Deleted Methods

Method deletion and renaming is not supported by 4D Job Wallet due to limitations in the 4D Design Access command set. You need to do this with the native 4D method browser, however upon each launch or refresh, 4D Job Wallet will detect deleted or renamed methods and reflect their state accordingly as follows:

In the Main Method List renamed or deleted methods will be removed from their original groups. Renamed methods will be treated as "new" methods as far as 4D Job Wallet is concerned and will be placed at the root level in the method browser. This means that if you rename a grouped method, you need to re-group it in 4D Job Wallet as if it were a brand new method because it will be re-tokenised with a different token.

In the Job Wallet list, renamed or deleted methods will be kept in place but represented with a special icon indicating that the method no longer exists. This acts as a placeholder to remind you that your "mind map" has changed and prompt you to reorganise the list to reflect the reasons for renaming or deletion



A deleted method as it appears in the Job Wallet method list

3.5 Structuring Your Codebase and Working with Groups

The core functionality in 4D Job Wallet that addresses complexity management in your codebase is the method grouping capability offered over 4D's native method browser. Method groups are supported both in the main method list and in the wallet interface. In 4D Job Wallet, method groups act as a productivity multiplier in 4 key ways:

- *contextulisation - by allowing you to build a static, permanent model of your core code base that serves as a reference inventory for more focused task based groups in the wallet interface*
- *portability - folders in the main method list are directly archivable as exportable libraries on disk which can be imported to another structure using 4D Job Wallet's Libraries features*
- *code templating - the main method list toolbar includes a Cloning feature which takes a method group as its input and is able to output a cloned method group with appropriately mapped method and variable names*

- *task management - main list method groups can be dragged into a task-based wallet and subsequently adapted according to the task specific requirements including renaming, re-assignment of method groups and wholesale restructuring of the original groups*

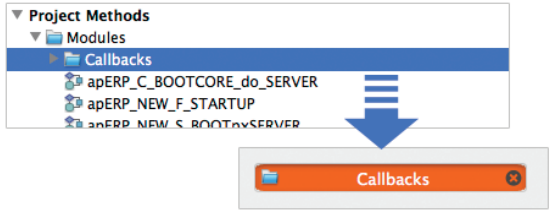
Since method groups are so important in 4D Job Wallet, it provides 3 distinct approaches to creating and maintaining them:

- *Contextual Menu - to create a method folder and populate it at the same time, highlight the methods and choose the* **Group** *command from the contextual (right-click) menu. Use this approach if the methods are close together in the list and there are only a few of them*
- *Dragging - to move methods into an existing group, simply highlight the items to be moved and drop them onto the group folder directly in the list*
- *Target Group Well - use this approach if the methods to be grouped are spread out or staggered amongst the main method list and you need to perform successive grouping actions without "loosing sight" of the target folder. This approach is described in more detail below*

Using the "Target Group" Well

This feature is the most powerful way to create complex groups because it lets you combine 4D Job Wallet's grouping function with the list filter to iteratively locate appropriate content without loosing sight of the target folder. To create and populate method groups using this approach, follow these steps:

1. either create a new group folder using the **Add Folder** tool or identify an existing group in the main method list
2. drag the folder to the **Target Group** well at the base of the method list. It will highlight in an orange colour to indicate that it's active
3. you are now free to manipulate the method list in any way needed to identify group content with which to populate the new group. For example you might want to use the list filter to locate all methods with a common prefix or which contain a common name segment
4. highlight the methods you want to group and drag them to the Target Group Well. The methods will be regrouped in the main list accordingly
5. when complete, de-activate the Target Group Well by clicking the little de-activation icon inside the well



Adding a Folder

You can add folders to the method list without regrouping methods. To add a folder follow these steps:

…at root level:

1. check that the list is not highlighted and click the **Add Folder** tool on the bottom toolbar (or right-click on the Project Methods root node and choose **Add Folder** from the contextual menu)
2. enter the folder name and save

…at a pre-determined level:

1. highlight the item in the list where you want to add the new folder
2. click the **Add Folder** item in the toolbar or right-click the list item and choose **Add Folder** from the contextual menu
3. the folder will be added below the selected item

The Add Folder tool on the main method list toolbar

Renaming a Folder

You can rename a folder simply by double-clicking or by choosing the Rename command from the contextual menu.

Deleting a Folder

To delete a folder, highlight it and perform one of the following actions:

- *choose the Delete command from the right-click contextual menu*
- *drag the folder to the wastebasket*
- *hit the Delete key on an extended keyboard*

3.6 Method Cloning

The main method browser features a tool which levers a group of existing methods as templates for the automatic creation of new ones. See the full discussion on method cloning later in the reference.

3.7 Synchronising with 4D and Keeping Tidy

4D Job Wallet maintains its own tokenised method inventory which is not sync'd in realtime with 4D. If you add or delete a method using 4D's native method explorer, 4D Job Wallet will not know about the new items until you click the **Refresh** button on the method list toolbar. To keep 4D Job Wallet's method inventory up to date, use the **Add Method** tool in 4d Job Wallet when you want to create new methods. Some other method actions however, are not supported due to limitations in 4D's design access commands. These actions have to be performed using 4D's own method explorer. The table below shows you when you need to use 4D's native method explorer and for which actions a refresh is needed to re-sync 4D Job Wallet's inventory with 4D:

Action	Where	Refresh Required
Add Method	4D Job Wallet	No
Delete Method	4D Native Explorer	Yes
Rename Method	4D Native Explorer	Yes

3.8 Deleted and Renamed Methods

When you delete or rename a method, this has to be done from 4D's native explorer. If you do this while 4d Job Wallet is open, the changes will not reflect in 4D Job Wallet's method list until either the Refresh command is invoked from the method list toolbar or 4D Job Wallet is restarted. In the case of deleted methods or renamed methods, the behaviour is as follows:

- **Deleted methods** *are removed from the main list (but are flagged as deleted in the wallet - see next section)*
- **Renamed Methods** *are treated as "new". They will disappear from their group and re-appear with the new name at the root level. You need to regroup these manually*

Purging Empty Folders

The main method list toolbar contains a function to purge the list of empty group folders. To invoke this action click the Purge command on the toolbar. The list will be scanned for any empty folders and these will be deleted.

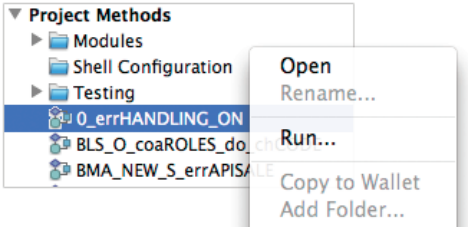
The Purge tool from the main method list toolbar

3.9 Executing a Method Directly from the Method List

You can execute a method from the contextual (right-click) menu simply by highlighting it and choosing the **Run** command. Note that it must have the "Shared by components and host" method attribute set, otherwise it is not executable by a component and the **Run** command will be disabled on the contextual menu.

Methods with Arguments (Parameters)

4D Job Wallet will attempt to detect if the method contains arguments ($1, $2 etc). If it thinks that this is the case then a confirmation message will be displayed warning that a runtime error may occur if you continue, but you will still be able to execute the method.

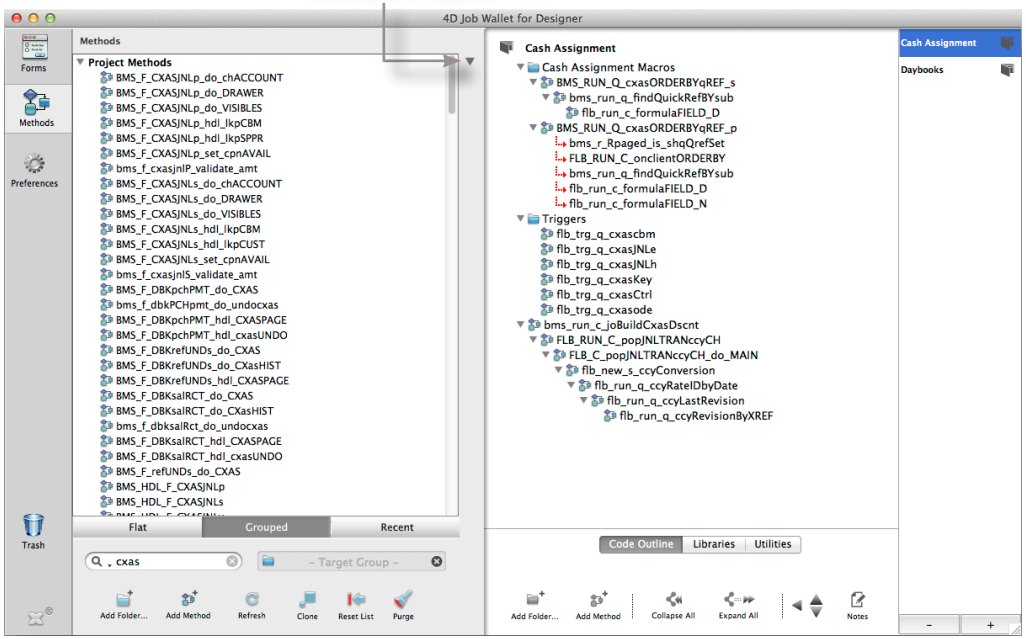


Executing a Run-compliant method from the main method list contextual menu

4 The Job Wallet

By invoking the wallet pane in the 4D Job Wallet browser interface, you can reduce your visible code base to a core subset required for focusing on a specific development task. Each "Job Wallet" is completely independent of all others and you can create a custom method list with its own task-specific group structure according to specific areas of your application.

Wallet pane "flipout"



The "Wallet" function is single biggest departure from 4D's native method explorer in that it supports multiple, highly focused, task-oriented method lists which can be structured in any way that best suit the work at hand. For example, you can nest methods within each other to represent a call chain or just keep regularly needed libraries to hand such as configuration methods and generic queries.

4.1 Creating and Populating a Job Wallet

To create a new job wallet, follow these steps:

1 flip the 'wallet' pane out from the main method list pane by clicking on

the small triangle at the top right hand corner of the 4D Job Wallet main method list

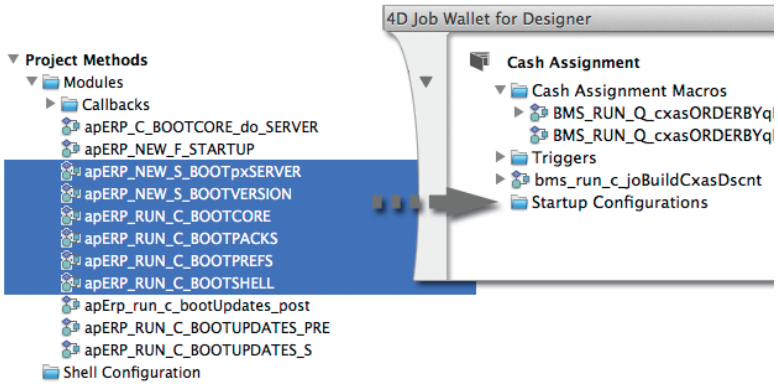
2. click the + button at the base of the wallet list (extreme right column in the wallet pane)

3. enter a unique name for the wallet and save

At this point, you have created an empty wallet. To populate the wallet follow these steps:

1. identify the items you want to include from the main list, either by inspection or by filtering

2. drag the items from the main list to the wallet area. You can drag either individual methods, staggered highlights (using command-key highlighted subsets) or entire folders



Content Restrictions

A given method can re-appear in any number of wallets but you cannot alias a method within the wallet - it has to be unique. If you attempt to populate a job wallet with a group of methods, some of which have already been added, then a message will appear indicating that only the unique methods will be copied to avoid duplicates.

4.2 Structuring a Job Wallet and Working with Groups

Managing the object content in a job wallet is similar to doing so in the main method list with some significant exceptions:

1. in the main method list, items are auto-sorted (e.g alphabetically) with no manual override. In the wallet interface, all items retain their original order until manually re-ordered using the nudge feature on the wallet toolbar

(see below)

2. in the wallet, methods can be "nested" within each other (for example, if you want to reflect a call chain as part of a "mind map"). This is not possible in the main list

3. when a method is deleted or renamed in 4D's native browser, it is not deleted from the wallet but rather indicated as "obsolete" by means of a red X symbol next to the method name. This serves as a placeholder for the legacy method which you then have to delete manually

4. the job wallet supports certain context-specific functions which the main method list does not such as discovery of called methods and calling methods

5. you can delete methods from the wallet list using the Remove function. This is not possible in the main list

[Tip: deleting a method from the wallet list does not delete it from the 4D structure nor from the 4D Job Wallet main method list]

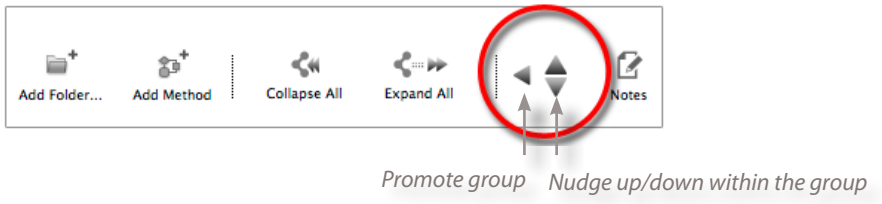
Reordering Items in the Wallet - The Nudge Tool

Items in a Job Wallet are not sorted automatically. They appear in the order you add them unless you manually re-order them. A special tool is provided for this purpose because dragging one item on top of another will otherwise cause it to be nested "inside" the dropped-on object instead of being re-ordered. To re-order the object sequence in a wallet, follow these steps:

1. highlight the item or items you want to nudge

2. to move the item up or down within its group click the Nudge Up or Nudge Down triangles on the wallet toolbar

3. to promote the item (move it into a higher level group) click the Nudge Left triangle



[TIP: To move an item to all the way to the top or bottom within its current nested group, hold the **Alt** *key down while clicking the nudge tool]*

Creating Groups in the Wallet List

Many of the contextual menu commands available in the main method list are mirrored in the wallet list. To create a new folder and populate it with content in a single action, follow these steps:

1. highlight the rows you want to include in the new group in the wallet list
2. right-click the mouse on the highlighted selection and choose the **Group...** command from the contextual menu

To create an empty folder in the wallet list hierarchy, follow these steps:

1. highlight the row *above* that which the folder is to be added
2. right-click the mouse and choose **Add Folder Below...** from the contextual menu or use the **Add Folder** command on the wallet toolbar.

To add a folder at the root level, follow these steps:

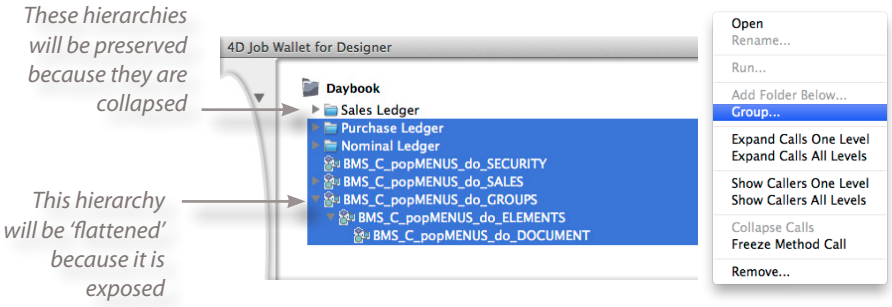
1. click the **Add Folder** command in the toolbar with nothing highlighted

Use of the "Group" Command with Hierarchies

If you highlight a range in the wallet list where the rows are not all at the same level, then the **Group** command will behave differently depending on whether items are expanded or collapsed:

- *members of expanded subgroups will be ungrouped and placed immediatelly below the new parent group being created by the* **Group** *command*
- *collapsed subgroups will have their internal hierarchy respected and only the parent item will be explicitly placed below the new group*

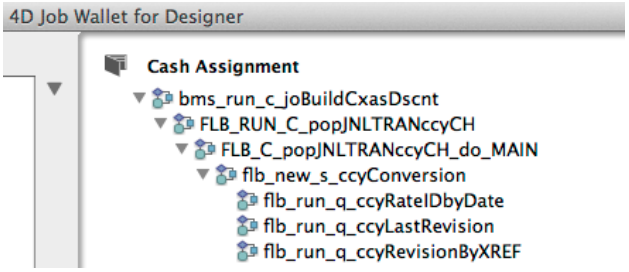
In other words, if you want to preserve the hierarchy of a subgroup which is to be included in the scope of the **Group** command, then collapse it first before grouping.



Use of the "Group" command on a hierarchical selection

Nesting Methods in the Wallet Display

Unlike the main list, the wallet list allows you to nest methods inside each other. This can be useful as an "outliner" when you want to represent call chains or complex dependency models. You can also nest entire groups inside a method. To create nested methods simply drop one method on top of another. You can drop methods in this way from both the wallet list and the main list to populate the wallet and nest it at the same time.



Nested methods in 4D Job Wallet

Un-nesting Methods

To revert a nested method back to a flat structure, follow these steps

1. highlight the nested method in the wallet list
2. click the Nudge Left triangle in the wallet toolbar to promote it by 1 level

How to promote a wallet item in the wallet hierarchy

Contextual Menu

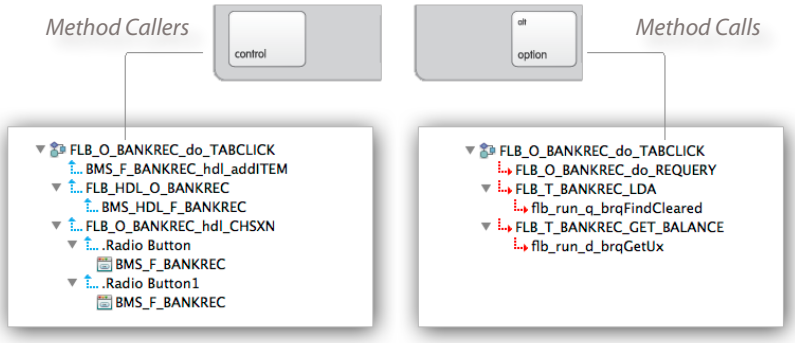
The Job Wallet list supports its own contextual menu where you access many of the toolbar functions in a context-specific way. To invoke the contextual menu, right-click on a row in the wallet list or click the gearwheel icon. The menu items are enabled according to the type of object highlighted.

4.3 Exploring Method Dependencies and Call Chains

4D Job Wallet supports dependency parsing which allows you to expand an item in the wallet list to reveal either called methods or callers. Called dependencies

are indicated by a red coloured arrow icon pointing from the caller to all its called methods. Callers are indicated by a blue icon pointing in reverse to the called method from all its callers.

[Tip: To toggle a single level of dependency at a time for calls or callers, **alt-click** *or* **ctrl-click** *(***alt-ctrl-click** *on Windows) a method in the wallet list respectively.* **ctrl-click** *ing a method which already has had its calls expanded will switch the dependencies from calls to callers]*



Shortcuts for expanding dependencies

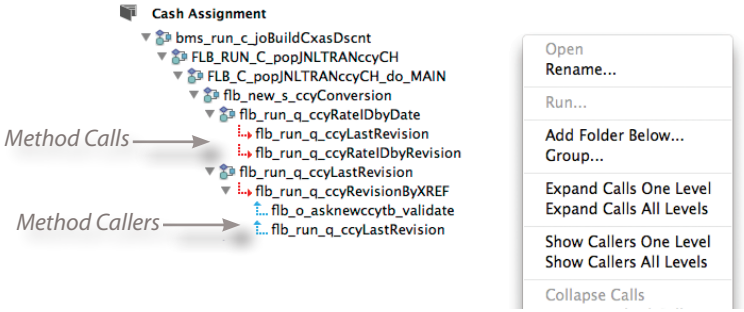
With one command you can expand the dependencies for a single item, a group of highlighted items or the entire wallet. You can ask 4D Job Wallet to reveal only the first level (primary callers and called methods) or all levels. If there are recursive calls in any call chain, the expansion for that particular chain will be halted at the first recursion.

To display callers or called methods in the wallet, follow these steps:

For Highlighted Items

1. highlight the item or items you want to expand
2. right-click the mouse on the highlighted items and choose one of the call chain expansion commands which function as follows:

- **Expand Calls One Level** *will reveal the immediate called methods in the highlighted items*
- **Expand Calls all Levels** *will make successive expansions down the call chain of called methods and their called methods until it reaches the end of the call chain*
- **Expand Callers One Level** *will reveal the immediate callers of the highlighted methods*
- **Expand Callers all Levels** *will make successive expansions up the call chain of caller methods and their caller methods until it reaches the beginning of the call chain*



For the Whole Wallet List

You can fully expand all call chains for the entire wallet list by clicking the **Expand All** command in the wallet toolbar. Successive clicks will expand one further level with each click.

[Tip: Hold the Alt key down to expand all levels in one click]

Limitations

4D Job Wallet uses a combination of raw text parsing and cross-referencing with it's own tokenised method inventory to attempt to detect method calls. It does not have access to 4D's native method tokenisation layer and is therefore potentially not 100% efficient in detecting all method calls. In particular, the following limitation applies due to optimisation techniques used:

[Warning: Method names with spaces in them are not supported by the dependency detection mechanism]

Disabled Commands and Clearing Dependency Displays

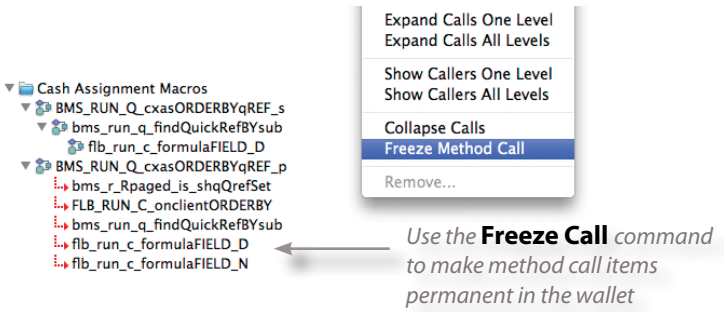
While dependencies are being displayed, certain commands are disabled to prevent loss of integrity in the wallet lists's group structure. Most of the disabled commands relate to modifying the wallet's content, dragging new content from the main list or deleting existing content. If you attempt to invoke any such actions while dependencies are displayed, a message will appear to the effect that the relevant command has been temporarily disabled. To clear dependencies, follow this step:

1 click the **Collapse All** button on the wallet toolbar or alternatively

2 right-click only the rows you want to collapse and choose Collapse Calls from the contextual menu

"Freezing" Method Dependencies

Calls and caller methods remain distinct from the normal wallet content when you invoke any of the call chain commands. The expanded methods will not become permanent members of the wallet list by default. However, sometimes you may want this to be the case and 4D Job Wallet therefore provides a command to Freeze dependency calls so that they remain in the wallet permanently. When you do this, the method will loose its dependency icon and switch to a regular method icon to indicate that this is now fully fledged wallet content. If you freeze a dependency which is not the immediate child of a permanent wallet member, its parents and ancestor items will also be frozen to retain the hierarchical integrity of the group. If you do not want these auxiliary items to be included simply highlight them and hit the Delete button on the extended keyboard.



4.4 Working with Multiple Job Wallets

You can quickly switch between wallets without loosing any of the changes you made to the current wallet. To set the current wallet, highlight a wallet name in the wallet list to the right of the method display. The wallet display will immediately update to reflect the new current wallet. There is no restriction on wallet content for a new wallet - you can have the same methods and method groups appear in multiple wallets as required, however, a method cannot be aliased within a given wallet.

4.5 Writing Coding Notes

4D Job Wallet features a notepad function which lets you create wallet specific or application general notes. The notepad will automatically reconfigure itself to display the notes corresponding to the current wallet - even in realtime - if it is open while you switch wallets. To invoke the notepad for writing, follow these steps:

1. click the **Notes** button in the wallet toolbar. The notepad will be displayed
2. switch to Write mode by clicking the pencil icon at the left end of the notepad toolbar
3. when complete, toggle Write mode back to Read by clicking the pencil icon again

[Tip: If you close the notepad without explicitly switching back to Read mode, your notes will still be saved - even if you closedown the entire 4d Job Wallet editor]

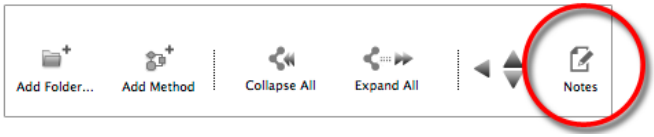
Exporting the Notepad Contents

The entire contents of the notepad for all wallets can be exported to disk by clicking the Export command on the notepad toolbar.

Other Notepad Functions

The remaining notepad toolbar functions comprise:

- **Scrollbar activation** - *click this command to toggle the scrollbar if the notes being viewed extend beyond the viewable area*
- **Opaque background** - *this function invokes an opaque background instead of transparent for easier viewing*
- **General / Wallet modes** - *when this control is in the General state, the notes displayed are not wallet-specific. Use this to create remarks which pertain to the application in general and not any one job wallet*




General Notes
General Notes
Wallet Notes
Notepad Commands
General Notes
Wallet Notes
Toggle edit
Show / hide scrollbar
Export notes
Toggle opaque background
Toggle general / wallet notes

5 Creating and Managing Portable Code Libraries

In most development environments, it's common to find an abundance of coding resources in the form of externally accessible code libraries. These usually take the form of text documents containing code which can be imported to a project and compiled in order to accelerate development and increase reliability. 4D's component model successfully fulfils this role for formal and complete functional units but proves to be too closed a mechanism for casual sharing of host code or for supporting many multi-developer scenarios.

4D Job Wallet introduces a library creation and management feature directed at addressing this shortfall, both in respect of delineating the library source code and in version controlling the associated library exports.

Features supported by the library management module include:

- *broad brush to fine grained content selection for populating exportable code libraries*
- *native diffing support on import at row and character level*
- *an inventory meta catalogue for content inspection prior to importing*
- *object level action and error logging during imports*
- *support for form and object methods as well as project methods*
- *optional automated version control via build number stamping*
- *user friendly build selection on import without recourse to the desktop*

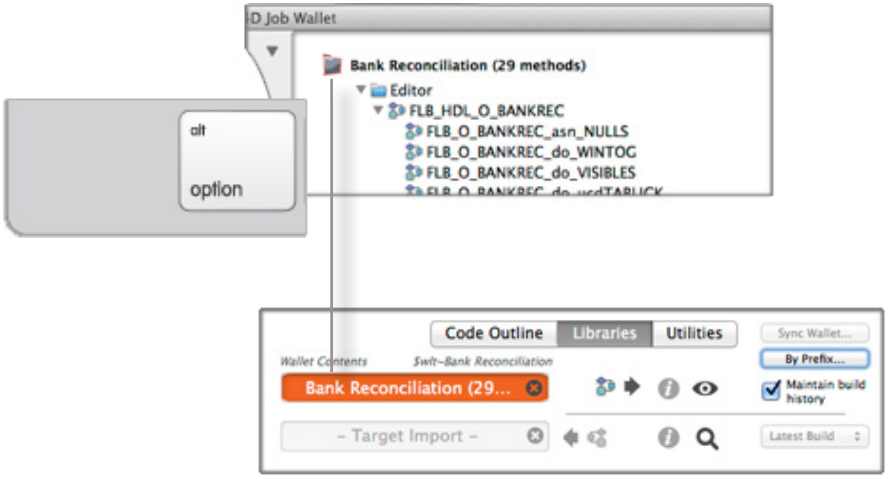
5.1 How Libraries are Created

The library management functions are located in the middle page of the wallet toolbar tab control under the **Libraries** tab. Select this tab to expose the library toolbar. To

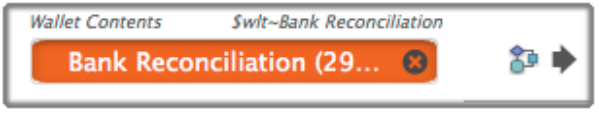
create a 4D Job Wallet code library, you need to first identify the library content in one of the method lists and then drag it to the **Target Export** well. This is known as "charging the export well" as it doesn't actually export it, only prepares the content for export.

To create a new 4D Job Wallet library, follow these steps:

1. drag a method or folder from the main method list to the Target Export Well or Option-Drag the wallet icon to export the wallet contents
2. click the Export command on the library management toolbar
3. 4D Job Wallet will write the library to an automatically named folder inside the 4D Job Wallet home folder
4. following the export you can use the locator tool **Show exported library on disk** to reveal the library folder in the OS desktop

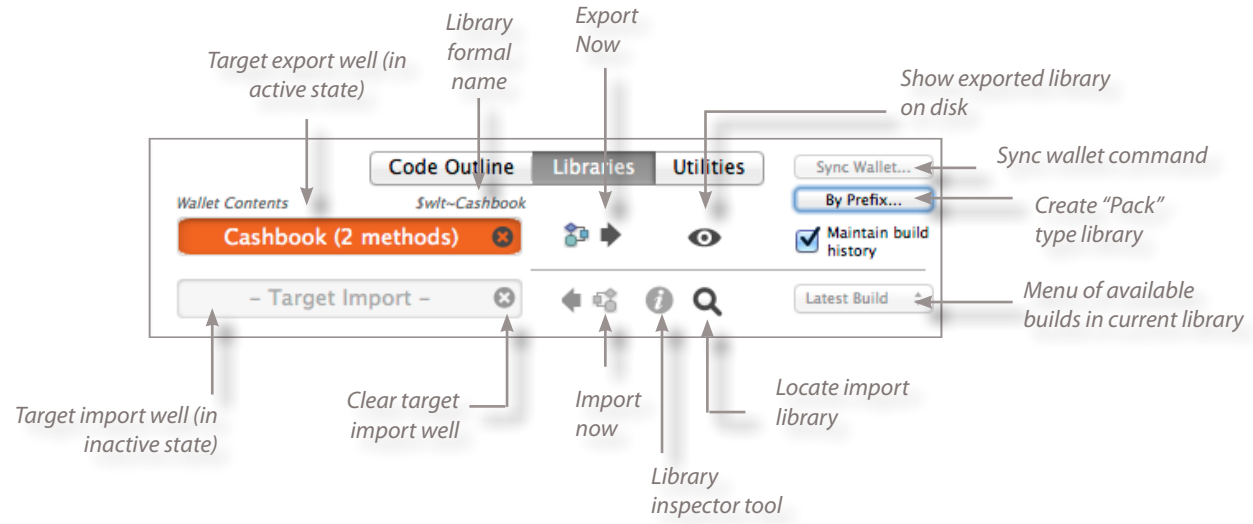


Creating a Wallet-Sourced Library Step 1 - Alt-Click the wallet icon and drag it (with alt key) to the target export well



Step 2 - Click the export command to generate the library or to append a new build to an existing library

[**Note**: Regardless of how library content is initially selected, no grouping information is exported. Only the methods are written to the library folder]



The library management toolbar

Selecting the Library Content

There are 4 options for content selection depending on how much fine grained control you need:

1. **individual methods** - to export a single method then simply drag it directly to the Target Export Well
2. **by folder** - to export the entire contents of a single folder in the main method list. In this case simply drag the folder directly to the Target Export Well
3. **by wallet** - use this option if you need ultimate control over what is being exported. Create an empty wallet and populate it with the methods (project, form and object) that you want to export. To charge the export well, **option-drag** the wallet icon to the **Target Export** Well
4. **by prefix** - there is a special option to export everything in the entire structure having a common prefix. In 4D Job Wallet, such libraries are referred to as Packs.

Identifying the Library on Disk

The normal location for 4D Job Wallet code libraries is:

/4D Job Wallet/Libraries

Note that in each case above, the library is given a provisional name and prefix according to its original content source. The type of library is displayed at the left end of the target export well and the proposed library name is displayed at the right end.

Library prefixes are generated according to type as follows:

Library Type	Prefix
Individual Methods	$mth-
Folders	$usr-
Wallets	$wlt-
Packs	$pak-

To complete the library name, the source object name is appended to the library type prefix. This allows you to identify the original type and source of the library content even after it's written to disk.

A folder will be created in 4D Job Wallet's desktop directory (located next to your structure folder) for the library ,where the exported methods will be saved. You do not need to locate the export destination - it is automatically determined based on the library name, however you can immediately locate the archive using the **Show exported library on disk** function (the 'eye' icon in the library management toolbar).

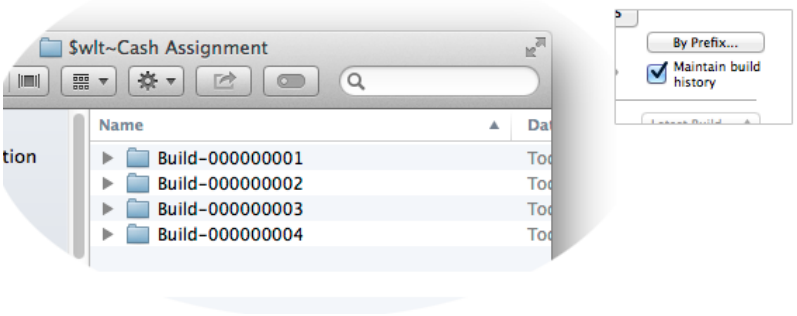
Inspecting an Exported Library's Contents

You can inspect the meta information associated with a recently exported library by dragging it out of the 4D Job Wallet home folder and loading it into the import well in order to use the **Library inspector tool**. This will display the library properties including date of export, type, UUID and some information about the source structure. In a separate tab, you can view the inventory list which documents the library's contents at object level.

5.2 Version Controlling a Code Library (Build Tracking)

The 4D Job Wallet library manager includes a version control feature which lets you maintain a history of successive library builds. With Build Tracking active, the library manager will not overwrite successive exports but write each output to a new "Build" folder inside the main library folder. At each export, the library build number is incremented and appended to the associated folder.

If Build Tracking is de-activated then each time you export a library with a given name, it will overwrite the last one exported (as "Build 0"). A warning message is displayed to this effect prior to exporting.



[Tip: You can change the default setting for version control in the 4D Job Wallet preferences]

Importing a Version Controlled LIbrary

When a version controlled library is loaded into the import target well, the "Builds" menu on the import row of the library management toolbar becomes active. By default it will set itself to the Latest Build option. To import a library build that is not the latest one, follow these steps:

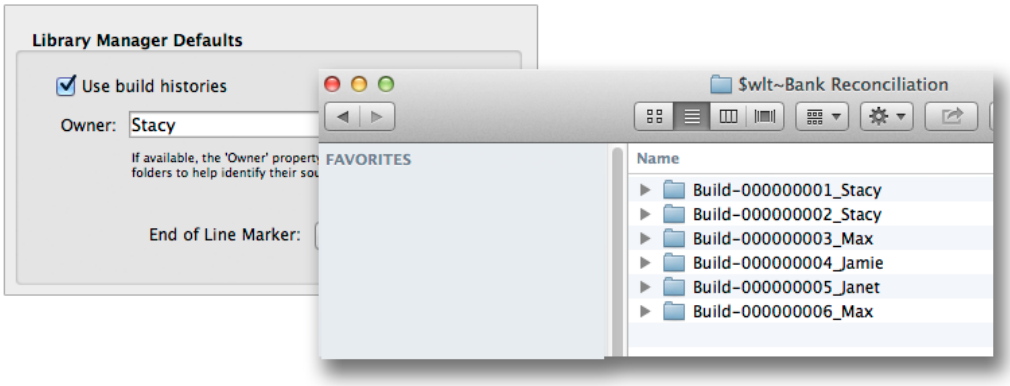
1. locate the library using the library locator tool in the library management toolbar
2. when the "Builds" menu activates, select the build number you wish to import

Associating Owners with Builds

If more than one developer is associated with the build history of a given code library, it can be useful to tag a caption onto the library builds to indicate their respective 'owners'. To have 4D Job Wallet do this automatically, you can configure a default owner for each 4D Job Wallet home folder as follows:

1. open the **Preferences** window from the side toolbar in the main method editor
2. enter the default owner under **Library manager defaults**

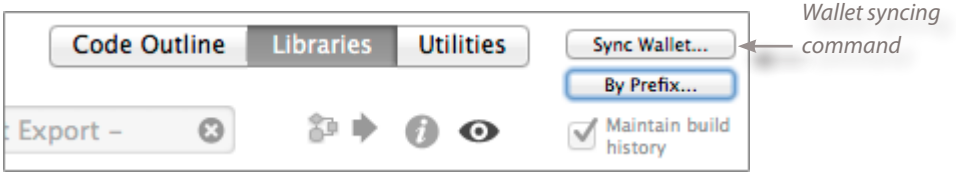
4D Job Wallet will append the owner property to each build that is exported from the structure.



Setting the Owner property in preferences means it gets appended to builds.

5.3 Working with Sync'd Wallets

This feature lets you use an external library to de-lineate a codebase subset within the current structure. This can be useful in multi-developer environments where there is more than one contributor and saves creating a new job wallet by inspection that matches the library contents.



[Note: the wallet syncing feature is used in library exports for the purpose of appending a new build to an existing library. For that reason it imposes certain constraints for consistency with the behaviour of regular library exports:

1. the wallet name must correspond to the library name minus the type prefix (For example, if the wallet name is "stock-queries" then a library should exist called "$wlt~stock-queries")
2. the syncing library must be located in the 4D Job Wallet home folder target output directory (/4D Job Wallet/libraries)

if no syncing reference library is present in the library output folder, 4D Job Wallet will display a message to that effect]

To create a wallet who's contents are sync'd to the library contents, follow these steps:

1. create a new empty wallet whose name corresponds to the reference

library name with which the wallet contents are to be syncd

2. move the reference library into the library output folder within the 4d Job Wallet home folder ("libraries")

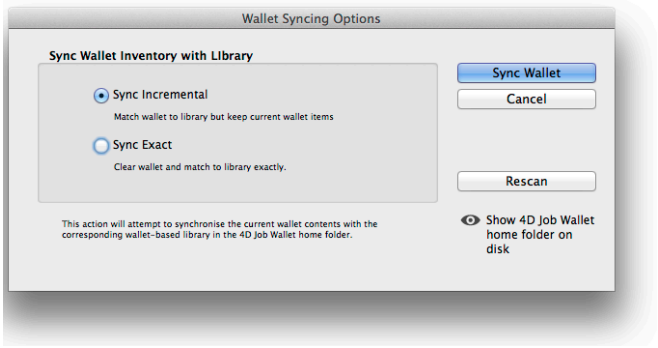
3. click the Sync Wallet command in the library management toolbar

If the library methods exist in the current structure, the wallet will be populated with those methods. If there are methods in the library which do not exist in the current structure then these will be ignored and the wallet contents will exhibit a shortfall with reference to the library contents.

[NOTE: The wallet syncing feature is not an import function. It simply uses the library inventory as an automated content selector for moving methods into the wallet. If a shortfall exists whereby some library methods are missing from the current structure, these should be imported first using the regular library import procedure, otherwise you may choose to simply ignore the shortfall and populate the wallet with structure methods which correspond to a subset of the library]

Incremental vs Exact Wallet Syncing

There are two syncing modes available which respectively determine whether to preserve or replace the existing wallet contents:



If **Sync Incremental** is selected, the new methods will be appended to the wallet. This mode is usefull if a developer wants contribute updates to an existing library but also add new methods. In this case, drag new methods into an empty library as they are being created and then do a final sync-incremental with the reference library to export a new build which is a superset of the last one.

If **Sync Exact** is selected, 4D Job Wallet will empty the wallet (methods are not deleted from the structure - just moved out of the wallet) prior to syncing so that an exact correspondence with the library contents is acheived.

5.4 How Libraries are Used - Import Steps

4D Job Wallet libraries are intended to function as coherent portable code units. Meta data is included with the library which contains information about the originating structure, user and desktop environment as well as an inventory of methods. This meta data approach ensures that the actual library contents are validated against documented inventory and that an audit trail is supported for the import of each object. If an error occurs during import of any single method or the method is skipped by the developer, this is logged and reported at the end of the operation.

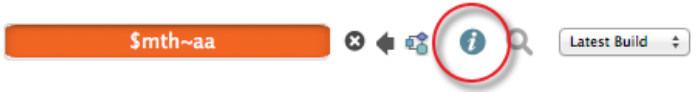
How to Import a Code Library

Most of the export commands in the library management toolbar are mirrored on the import toolbar row immediately below. The import tools are not enabled until the **Target Import** well has been "charged". To locate and import a code library, follow these steps:

1. Use the locator tool to locate the target library on disk (double-click the library folder starting with the $ sign and click Open)

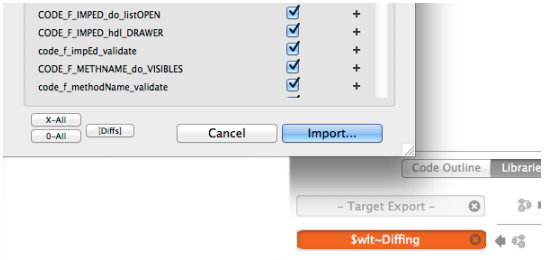
2. If you have located a valid library folder, the import toolbar functions will be enabled

3. Use the Inspector tool to check the library properties and inventory (optional)



4. If the inventory is satisfactory, click the Import tool to activate the **Import Control Form**

6. Click the **Import** button to start reading the library

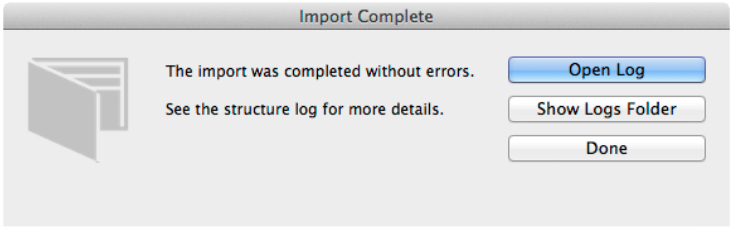


7 When import is complete, 4D Job Wallet will present a message allowing you to inspect the import log. If any errors occurred during import, these will appear in the log

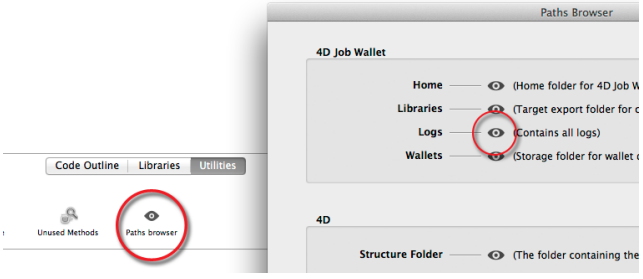
[Tip: To make a recently exported library more accessible for import, use the quick locator function to reveal the 4D Job Wallet home folder and drag a library folder out to the desktop]

The Structure Logfile

All changes that 4D Job Wallet makes to the structure are logged in a document called structure.log. This is located in the 4D Job Wallet home folder in /logs.



You can quickly access the log folder either by opening it from the "Import Complete" message presented immediately after an import or by looking it up from the **Paths Browser** tool in the **Utilities** tab.



De-Charging the Target Export Well

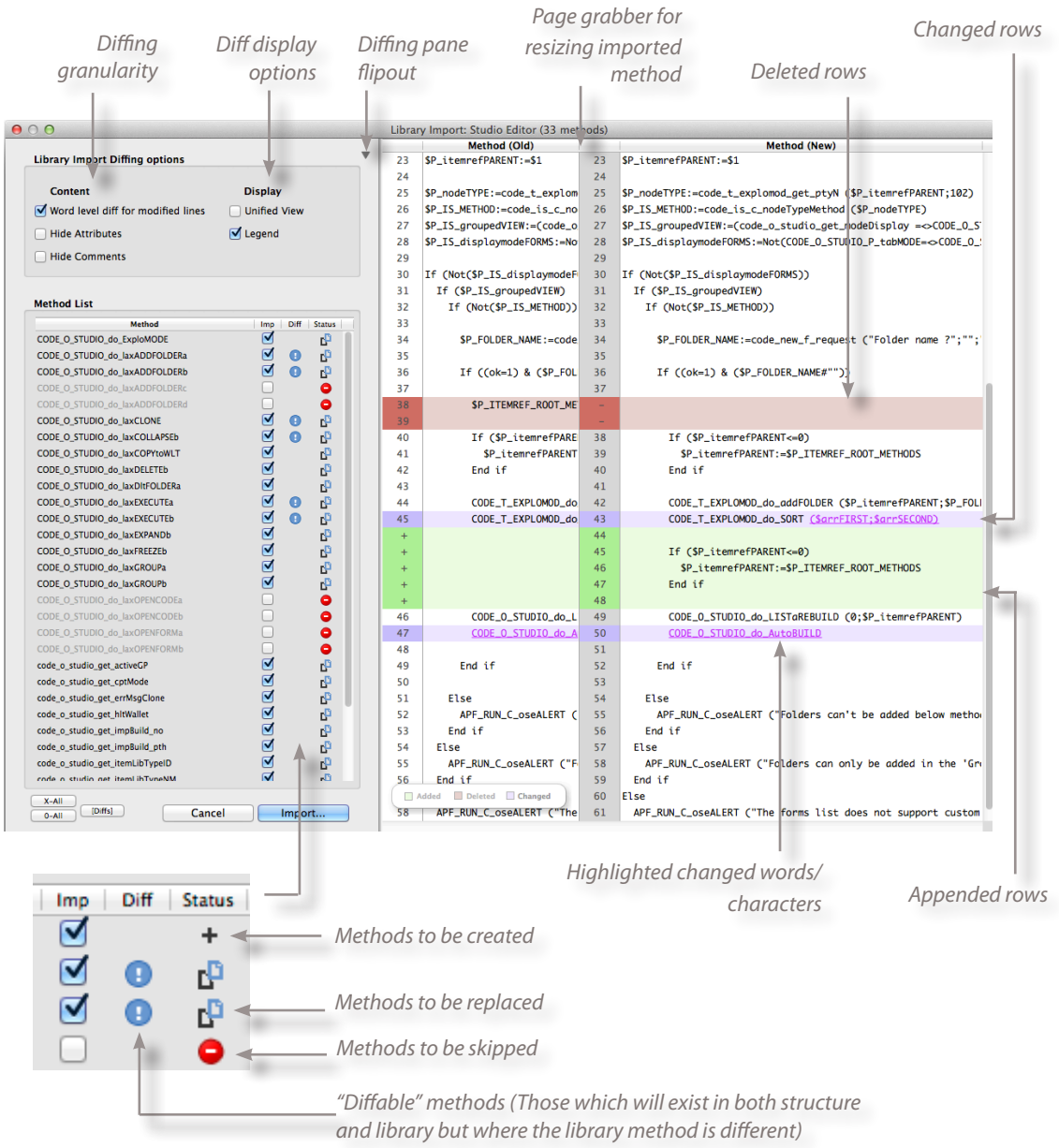
To de-activate the target export well, click the small cancel button inside the well. This will un-highlight the active library and disable the export functions.

5.5 Doing Selective Imports and Method "Diffing"

4D Job Wallet provides granular control over the import of library contents by way of the **Import Control Form** which lets you perform a number of relevant actions prior to the final import. In particular, this feature lets you:

- *detect which methods have changed in the library relative to those in the current structure*
- *skip selected methods and exclude them from the import*
- *perform automated code comparison ("diffing") actions on all library contents and have 4D Job Wallet highlight code differences for you at row and character level*

The **Import Control Form** is invoked unconditionally for every library import. If you just want to import the library in full you don't have to do anything else except click the **Import** button.



Selecting a Method Subset for Import

To include or exclude individual methods form the import, click the checkbox in the **Imp** column. When a method is excluded from the import, a red icon will be displayed in the status column and the method's name greyed out. The fact that the method was excluded from the library import will be registered in **structure.log** at the end of theimport.

Invoking the Diffing Pane

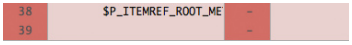
Click the small triangle at the top right hand corner of the import control form to open the diffing pane.

[NOTE: In the discussion that follows, by convention the **library** *method is considered the* **new** *method and the one in the current* **structure** *is refered to the* **old***]*

Interpreting Code Comparisons

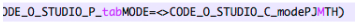
4D Job Wallet implements Myer's classic diffing algorithm at row and character levels to display 3 types of revision to the original method in the current structure:

1. **Deleted rows** These are rows of code which exist in the old method but which are no longer detected in the new method being imported. They are represented by a red highlight. In the side-by-side view deleted rows are always blank in the library (imported) method and filled in the structure method



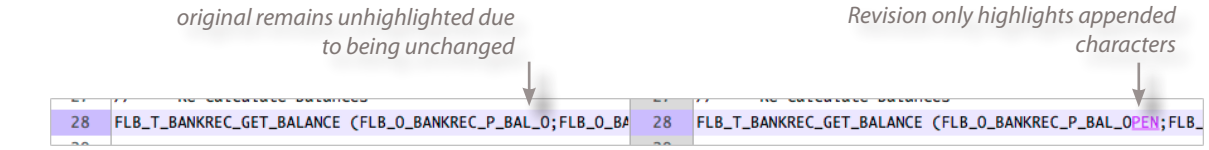
2. **Appended or Inserted Rows** Code which has been added since the old version is indicated by a green highlight. Here the same pattern applies in reverse - the old method rows are blank and the new method rows are populated in the side-by-side view

3. **Changed Rows** If the diffing algorithm identifies a method row as being modified between the old and new method, it will highlight this in purple and embolden the parts of the row which have changed with respect to its counterpart version

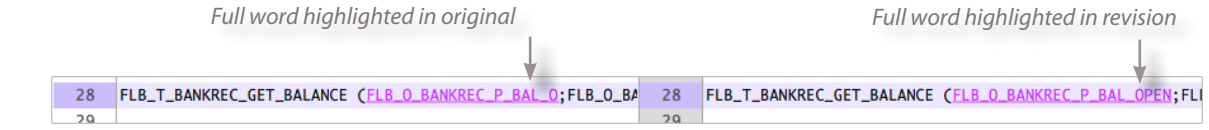


Diffing Granularity - Word vs Character Level Diffing

For modified rows, you can choose to highlight differences at word level rather than character level for easier reading. If the **Word level diff for modified lines** option is checked, 4D Job Wallet will extend any highlights out to the limit of the word to which it belongs. This can be useful for highlighting changes to variable names or structure objects which appear in the code.



Modified row display with word level diffing off



Modified row display with word level diffing on

[NOTE: The Myer's algorithm often has multiple solutions to a given diffing problem. In this case, 4D Job Wallet's implementation arbitrates in favour of deletions from the original. This may not always result in the most intuitively satisfactory result. Toggling the diffing granularity can help to mitigate this problem and identify differences more quickly in complex lines of code]

Unified View

The diffing pane supports an alternative unified view in which the steps to reach the revised version from the original are indicates purely as row deletions and row additions. To invoke the unified view click the **Unified View** checkbox in the display options of the library import control form.

6 Method Cloning

The method cloning tool is useful if you need to create a small group of methods which are "templated" from an already existing set in your codebase. The cloning tool performs the following actions for each method in the cloned set:

1. creates a new method who's name is "mapped" from the template method name using a replace rule
2. copies the template method code to the new method
3. performs a succession of search and replace actions on the method content according to the mapping rules in the cloning tool

This can be useful when your code contains regular logic patterns that are not consistent enough to refactor as generic methods, for example cross table query logic, iterative object handling on forms or batch processing on tables.

[Note: The cloning tool outputs to a single target folder. Before using the cloning tool, you need to create an empty folder and drag it to the Target Group well]

6.1 Creating the Methods

To create a cloned method set, follow these steps:

1. create an empty folder to serve as the output folder for the new cloned methods (you can use an existing folder
2. highlight the method or methods you want to clone (you can also highlight a method folder)
3. open the cloning tool by clicking Clone in the main method list toolbar (Note: if the output target folder is not empty, a warning to that effect will be displayed pointing out that the cloned methods will be mixed in with

that folder's existing content)

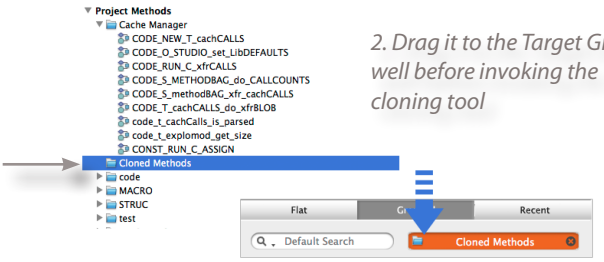
4 check that the list of template methods is correct

5 enter a mapping for method names. Unless you use manual naming, this part is mandatory because otherwise 4D Job Wallet would create duplicate methods

6 enter one or more mappings for the method content. These will invoke a "search and replace" type operation on the method content and is intended for supporting mappings for variable names and method names

7 click the **Clone** command

1. Create an empty folder to hold cloned methods

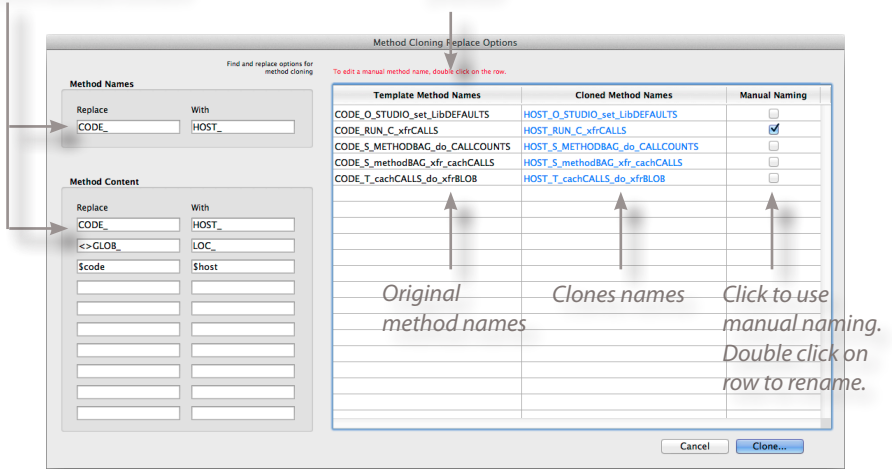


2. Drag it to the Target Group well before invoking the cloning tool

The cloned methods will be delivered to the folder you originally dragged to the **Target Group** well prior to invoking the cloning tool. From there they can be edited and fine-tuned as needed.

Mapping rules for method names and method content

Cloned method names preview



Original method names

Clones names

Click to use manual naming. Double click on row to rename.

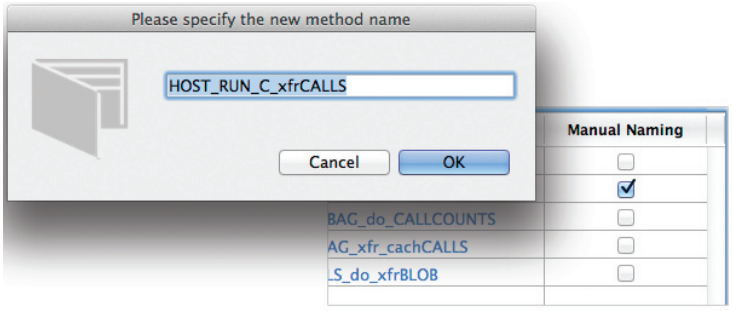
The cloning preview tool

[Tip: Any automated changes to the structure are logged in structure.log. See the discussion on logging in the section on importing code libraries. You can locate the log file from the paths browser in the Utilities tab]

6.2 Manual Method Naming

If the template method's naming convention is not systematic enough to be mapped automatically, you can override this feature by specifying manual names directly in the cloning tool's method list. To use manual naming, follow these steps:

1. activate the checkbox in the Manual Naming column of the template method list
2. double-click on the method name(s) you want to manually name
3. enter the new method name and click OK



7 Utilities

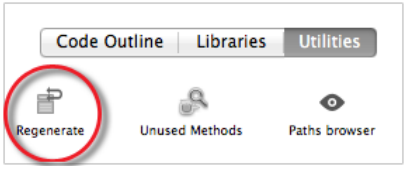
To compliment the core feature themes in 4D Job Wallet, a collection of utilities is provided which together represent a comprehensive toolkit for levering your development effort in the host layer.

7.1 Regenerate Tables

This tool is a "wrapper" interface for 4D's **REGENERATE MISSING TABLE** command. It supports a user interface for this action which allows the datafile - as opposed to the structure file - to be used as a vehicle for porting tables from one structure to another. The regenerate tables approach is useful in managing forks. See the appendix discussion on managing forks for more information.

To recover tables from the data file and represent them in the current structure, follow these steps:

1. open a datafile containing tables belonging to a code fork not supported by the current host structure (referred to in 4D documentation as "missing tables")
2. open the **Regenerate** tool from the **Utilities** toolbar in the wallet pane



3. activate the checkboxes for each missing table to be recovered
4. click the Recover command
5. if successful, the tables will appear in the **Recovered** list on the right hand side of the editor

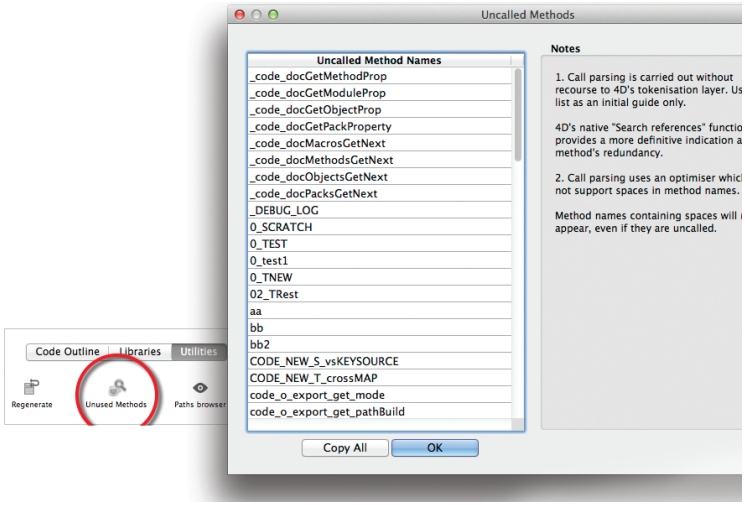
To verify that the recovery is complete, visually inspect the structure editor to locate the recovered tables.

[Note: Changes made to the structure by this tool are recorded in structure.log along with any errors encountered during the recovery. See the discussion on structure update logging in the section on code library imports for more information]

7.2 Detecting Uncalled Methods

4D Job Wallet levers its dependency parsing logic to report on uncalled methods which can be useful in identifying redundant code in your application. To detect unused methods, follow these steps:

1. open the **Unused Methods** tool from the **Utilities** toolbar in the wallet pane
2. a reparse may be required if you added any methods since the last 'dependencies' parse (which is distinct from a regular parse)
3. a list of methods which 4D Job Wallet thinks may be redundant is displayed
4. the list can be copied to a text editor by clicking the **Copy All** command



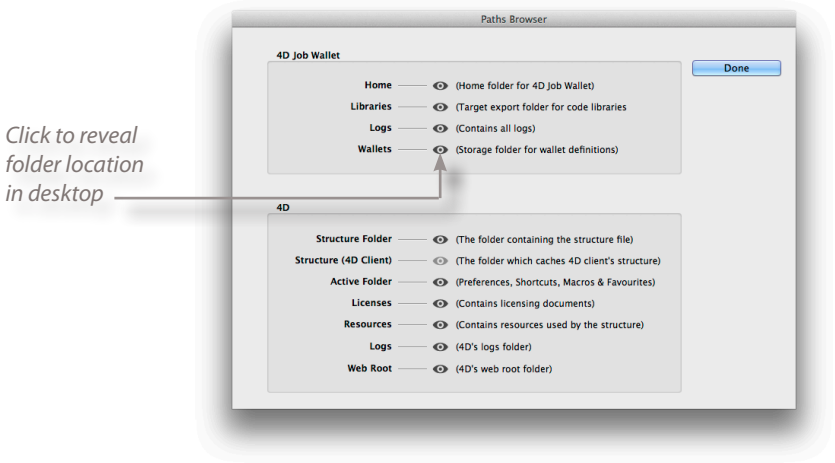
[Warning: 4D Job Wallet's dependency parsing may not be 100% efficient at detecting all dependencies due to the fact that it does not have access to 4D's native tokenisation layer. Always verify with 4D's Find in Design tool if you are unsure that a method is in use prior to deleting it]

7.3 Paths Browser

The home folder that 4D Job Wallet uses for its various resources and outputs is located in the 4D structure folder for each structure that uses the component. It adds several new paths to the already burgeoning list that 4D has, so this tool provides an overview and quick access to all the key desktop locations used by both environments.

To use the paths browser, follow these steps:

1. open the **Paths Browser** tool from the **Utilities** toolbar in the wallet pane
2. to reveal a location on the desktop, click the locator icon next to each folder identifier



8 Comparing 2 Structure Snapshots

Increasingly, 4D business application developers are finding themselves having to manage more than 1 version of a host application. This can occur for many reasons including situations where:

- *there is more than one developer working on coding tasks and the work has to be merged from 1 or more "forks"*
- *an application vendor wants to create a customised version of a generic code base and therefore has to manage a separate fork while periodically synchronising common code*

You can perform structure comparisons on part or all of a 4D application using the **Structure Snpashot** tool in 4D Job Wallet. This feature performs a raw text export of meta information such as table, form and object properties plus all method code in a hierarchical format that can be processed with a 3rd party text comparison (or "diffing") tool such as Kaleidoscope (http://www.kaleidoscopeapp.com).

8.1 Installing the Structure Snapshot Callback Method

Due to limitations in the scope of some commands in the design access theme, the Structure Snpashot tool requires a single callback method to be installed in the host code for comprehensive support of all elements of the structure.

The method code is included with the 4D Job Wallet installation package and will be installed automatically in the host with v14 and greater versions of 4D when you invoke a **Structure Snapshot** for the first time.

Note: Do not modify the code of the callback. The 4D Job Wallet component maintains its own copy of the callback method which will be used to validate the integrity of host callback. If they do not match, the associated functions in the structure snapshot will be disabled.

8.2 Creating a Structure 'Snapshot'

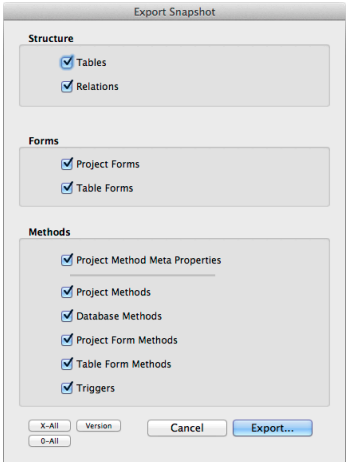
4D Job Wallet will create a comprehensive property export for most detectable properties in the database. The export generates a hierarchical folder structure on

disk containing the property data so that the diffing tool can report the changes at successive levels of granularity using a drill-down technique. For example, at the top level, you can immediately detect if changes have occurred in forms, tables, method code or relations. The next drill down one level in code-content, for example, resolves to database methods, project forms, project methods, table forms and triggers and so on all the way to individual lines in method code.

To create a structure snapshot, follow these steps:

1. open the **Structure Snapshot** tool from the **Utilities** toolbar in the wallet pane
2. select the property categories you want to analyse by activating the associated checkboxes
3. click the Export button

4D Job Wallet will create a timestamped folder which is sortable alphabetically in the desktop environment.



8.3 Using a Diffing Tool with the Snapshot Data

There are a wide range of text comparison and analysis tools available on the market. In this example we'll use Kaleidoscope who's feature set supports hierarchical parse reporting which represents a perfect compliment to 4D Job Wallet's snapshot facility. You can download a time limited demo of Kaleidoscope from their website http://www.kaleidoscopeapp.com.

To open a 4D Job Wallet snapshot in Kaleidoscope, follow these steps:

1. drag the first timestamped snapshot folder onto Kaleidoscope's data drop

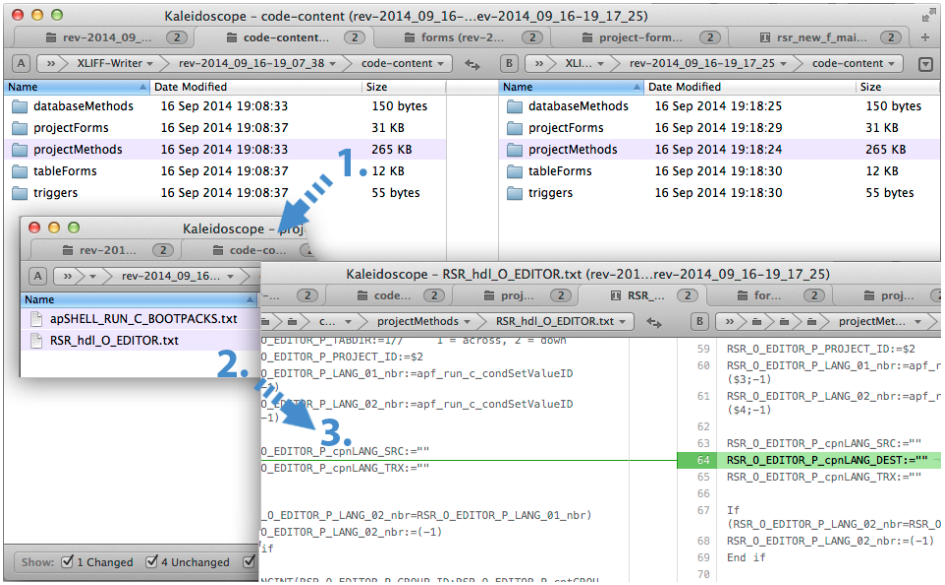
area. The main toolbar will appear

2. drag the comparison timestamped snapshot folder up to the top toolbar and drop into well labelled B "At Least two Files Needed"

3. Kaleidoscope will now parse the two structures and display the top level of the snapshot hierarchy with any differences highlighted

4. to simplify the display and focus only on changes from one structure to the other, de-activate the **Unchanged** checkbox in Kaleidoscope's bottom toolbar

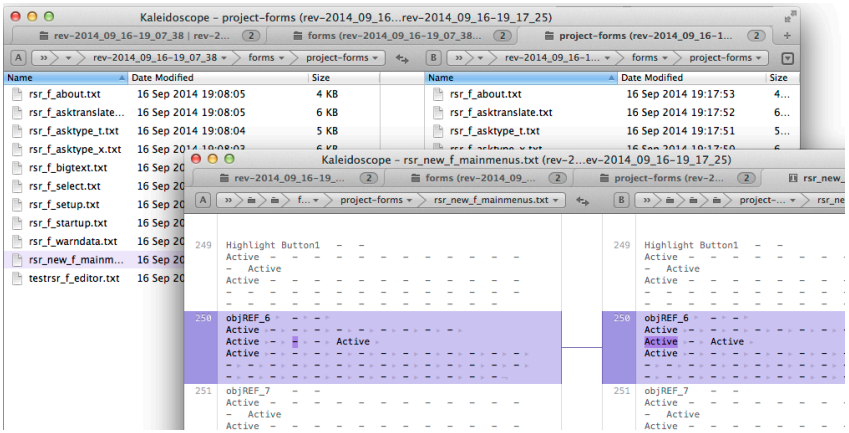
5. you can now drill into any of the property categories to explore the detail of the high level changes reported



Auditing The Snapshot by Inspection

While the automated snapshot parsing with "diff-ing" tools represents a rapid way of identifying differences, it can sometimes obscure which objects are involved. To address this, the diffing approach can be complimented with a spreadsheet view where certain database properties are output to a grid.

The snapshot is written in tab-delimited format which allows for easy import into most spreadsheets. In all cases, object properties are listed either 1-dimensional attribute-value pairs or 2-dimensional grids.



A diff'd snapshot of a project form where the "On Clicked" event has been activated

	A	B	C	D	E	F
178	object-name	on-load	on-validate	on-clicked	on-header	on-printing-break
179	form-level-events	Active	Active	Active	Active	Active
180	objGROUPLIST	-	-	Active	-	-
181	objREF_6	-	-	Active	-	-
182	Button5	-	-	Active	-	-
183	Button1	-	-	Active	-	-
184	3D Button	-	-	Active	-	-
185	objREF_52	No object method				

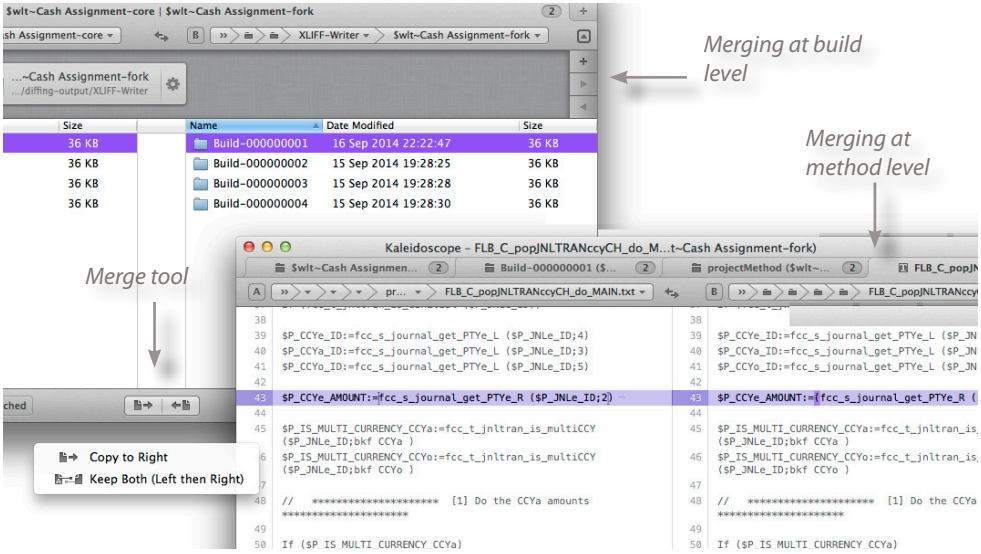
The same data viewed in a spreadsheet

8.4 Performing Code Merging between Forks

Continuing with the Kaleidoscope example (most other diffing' tools support this feature), when drilling down to method code, you can merge one codebase into the other using the merge tool in the bottom toolbar.

You can also take advantage of the snapshot's hierarchical structure and Kaleidoscope's hierarchical support by merging at a category level as opposed to method level.

[Note: If you intend to work across structures with any regularity and perform merge operations on them with a 3rd party diffing tool (as opposed to 4D Job Wallet's native one), it is recommended to work with code libraries instead of structure snapshots. These can be automatically re-imported after the merge is complete whereas method code from structure snapshots has to be manually imported].



Merging a 4D Job Wallet code library at build level and at method line level

Appendix A: Maintaining a Forked Codebase

One of the most challenging tasks that often face 4D development projects is maintaining customised versions of an otherwise common (host level) codebase. For example, you have an ERP solution which is targeted at the general market. You want to create a module which serves a specific vertical market but is integrated with the generic product.

The difficulty here is in keeping the forked version up to date with changes in the generic codebase. Simply migrating the changes into the custom fork is barely viable because:

1. it forms the majority of the codebase and therefore potentially requires a substantial effort to migrate
2. updates to the structure have to migrated by inspection as they are incremental
3. it creates redundancy if there is shrinkage in the generic codebase

On the other hand, migrating the smaller custom codebase from the forked structure into the generic trunk presents its own problems, not least datafile incompatibilities stemming from revised table level UUID identifiers after they are migrated.

This section looks at deploying several complimentary features from 4D Job Wallet to solve the forking problem and presents a technique to optimally manage such a scenario.

A.1 Creating the Fork

It's useful, if not essential, for the every object in the customised structure to be clearly distinguished from the original codebase. Normally a prefix is used to denote

a modular de-lineation in this regard. The prefix is ideally extended to all relevant structure objects including project methods, forms and tables. The forked structure may contain more than one customised module as long as it is easily identifiable and distinct from the common code.

A.2 Subscribing to Updates from the Common Codebase

Lets assume that the common codebase has undergone revisions and we want now to update the custom version with these revisions to keep it in "sync". We ideally want to merge the smaller codebase (custom code) into the larger one (common code) while at the same time retaining datafile compatibility with the resulting merged structure.

To solve this, we use a "bi-channel" approach to migrating the custom code objects where the method and form objects are copied directly from custom codebase to the common codebase while the datafile itself is deployed as a vehicle for migrating the custom tables. This maintains the consistency of the table level UUIDs and therefore datafile compatibility while at the same time supporting the "small into large" merge requirements.

In the following steps, the respective structures as referenced as follows:

- *Common code base (trunk application): Structure-A*
- *Forked codebase (custom application): Structure-B*

To carry out the merge with 4D Job Wallet, follow these steps:

Step 1: Migrate Custom Tables into Generic Codebase Trunk

1. backup all essential resources including structure and datafiles
2. open the Structure-B's datafile with Structure-A (generic structure). If Structure-B is a genuine fork of Structure-A then this will be possible because its UUID will still correspond to the forked datafile
3. launch 4D Job Wallet while having the custom (Structure-B) datafile open in Structure-A
4. open the **Regenerate** tool from the **Utilities** tab of the wallet toolbar
5. the custom tables from the datafile will appear if detected, even though they still don't exist in the current (generic) structure
6. regenerate the tables (see the section on using the Regenerate utility earlier)
7. manually re-assign any relations

Step 2: Copy the Forms

1. open the two structures simultaneously in respective copies of 4D
2. use 4D's native transport mechanism to migrate all of Structure B's custom forms to Structure A

Step 3: Create the Custom Code Libraries *

1. in Structure B, open the wallet pane in 4D Job Wallet and navigate to the Libraries tab
2. create a **Pack** type library in the **Libraries** tab for each custom code module (use the "By Prefix" command to specify the module prefix). By choosing a Pack module you will ensure that all the custom code elements are scavenged, including project methods, form and object methods
3. export the Pack library(/ies, if the fork contains more than 1 distinct prefix)
4. close Structure B and open Structure A
5. import the Pack modules from Step 3

** you could use 4D's native migration mechanism for this step, however there are reported issues with this which may produce unpredictable results, in particular where triggers are concerned or methods containing multiple table references*

At this point the fork integration is complete. You should have a structure which exhibits all of the following states:

- *contains the updated version of the common codebase*
- *contains the latest custom code fork*
- *is backwardly compatible with the custom structure's datafile*