Application Management 1.1

Tutorial

June 1994

Georg Heeg
Objektorientierte Systeme
Baroper Str. 337
44227 Dortmund

Tel: (x 49-2 31) 9 75 99-0
Fax: (x 49-2 31) 9 75 99-20
email: info@heeg.de

This document was written, formatted and illustrated using the Analyst Document System.

Contents

This document introduces into the work with Application Management. Following the examples and excercises step by step will give you some good practice on creating, managing and maintaining your Smalltalk applications independently. Moreover, you will find hints and exercises on advanced techniques and functions of Application Management. The reader should be familiar with basic techniques in the ObjectWorks Smalltalk development environment, suchas opening browsers or inspectors. You should know browsers, inspectors, and text menus, and how to execute code in a workspace.

For demonstration purposes the Tutorial comprises the files

tutorial/gsdummy/v1/s.st

tutorial/gsdummy/v1/applic.bos

tutorial/gsdummy/v1/versions

tutorial/ihsupprt.st1

tutorial/ihsupprt.st2

These files include source code for an in-house support application of a software company.

1 Creating Applications

1.1 The Basic Method

This method applies both for beginners in Objectworks Smalltalk, and when you start a new application (or project) with no or only few classes and methods implemented:

- Select the menu item ’new application’ in the Application Management Browser.
- A dialog prompts for the name of the new application. Input is valid if the name is not already used for some other application.
- Make sure that the ’modified’ filter of the Application Management Browser is disabled, otherwise you won’t see the new application in the list of the Application Management Browser.

Selecting the menu item ’open browser’ in the Application Management Browser opens an Application Browser, where you can construct the new application step by step. These steps will be discussed later, so we directly move over to the "fast" methods of creating applications.

1.2 Loading Applications

No doubt, this is the fastest method to install an application in your image. It requires that you already created applications with Application Management and saved them in the file system.

Demanding to load an application starts a retrieval which looks up certain directories (called the "search path" of Application Management) for application directories. The following image shows the dialog inquiring the search path.

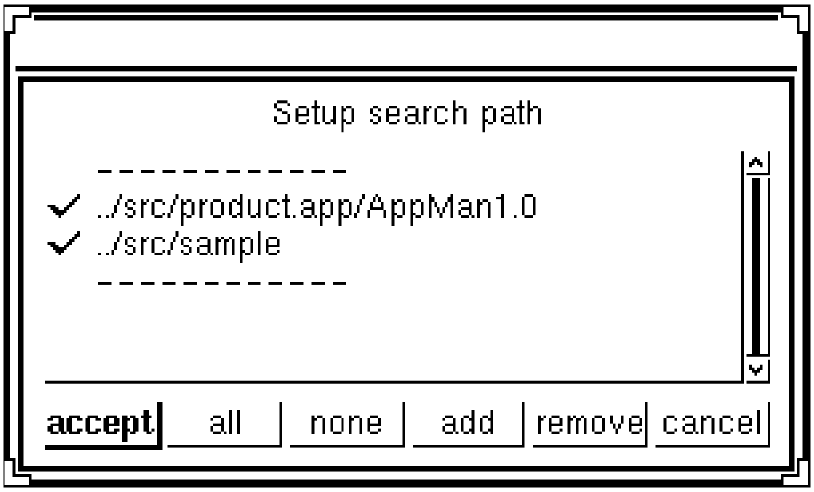

Figure 1: Dialog for setting up the search path

Execute the exercise following the next steps.

- Select the menu item ’file in/out> load...’ in the Application Management Browser.
- Make sure that the search path of Application Management includes the path to the tutorial applications.
  - Press the button ’add’.
  - A dialog asks for the name of an existing directory. Type in the string ’tutorial’ probably preceded by the name of the directory where all Application Management applications reside.
  - Deselect all paths other than the tutorial path; deselected items are skipped during the next search, thus minimizing retrieval time. To discard a path from all future search use the button ’remove’.
  - Press the button ’accept’.

Application Management now searches the given paths for applications and shows the result in a menu.

- Select ’tutorial gsdummy v1’.

- A dialog will ask whether you want to load in a binary mode or not. This happens if the chosen application has a binary file. Answer ’yes’.

  In general you should answer ’yes’, because this may shorten loading time remarkably. Answering ’no’ invokes a normal file-in of the source code file of the application (without appending the sources to the image changes file).

  Note: The current version of Application Management sometimes has problems with binary import of classes which define the method #binaryRepresentationVersion; select the non-binary mode in those cases.

Now you have rebuilt all the class and method definitions of application ’gsdummy’ in your image. You can now open an Application Browser and see what happened. The current changes of ’gsdummy’ might be interesting: any existing class or method definition which overwrites source residing in a file other than the changes stores a ’backup’ of the previous version. This allows restoring the prior state or comparing with the new one. Have a look at section ’definition versions’ for details.

If the name of the application to be loaded identifies an already registered application Application Management asks if the registered version should be replaced. Answering ’yes’ proceeds loading application structure and source. After completion Application Management checks for possible ’garbage’: Each class or selector registered in the previous version of the application but not registered in the new one is a candidate for a removed item. If there are such candidates a Change Browser opens with each list element representing a candidate for removal. If you are sure that you don’t need the classes and/or selectors shown by the list just select the item ’replay all’ from the Change Browser menu, thus actually removing the given classes and selectors. But be careful: An element which does not occur in the new application might also have been moved to another application. In this case you have to restrict the change list to elements verified for removal and apply replay functions.

1.3 Generating Applications

This section shows how to generate applications from existing sources. The described method applies well to those cases where you have quite a good amount of definitions residing in one or more source files (especially file-in packages of software vendors).

- Open a FileBrowser (item ’file list’ in the ’utilities’ menu of the Launcher)

- Select file ’ihsupprt.st1’ (probably preceded by the name of the directory where the tutorial sources reside).

- Select the menu item ’file in’

  This is just the normal loading of Smalltalk source code from a text file. The only difference to the standard procedure is that Application Management shows a menu to select an application which should register the definitions and log the changes resulting from the file-in. In principle you may select any application as target of definitions and changes.

- Since we are now interested in creating an application, select the first element of the menu shown: ’NEW APPLICATION’.

- Application Management prompts for the name of the new application. Enter ’InHouseSupport’.

- Next you will be asked: ’Do you want the sources to be kept external?’. Answer ’yes’.

  This question arises with the special source code management of Application Management. If you answer ’no’, all the source code in the file to be read will be appended to the ’changes file’ as usual. If you answer ’yes’ this will not happen; each access on the source code will read this file. The only restriction for you is that you must not remove or edit that file until a new ’major version’ of the chosen target application has been released. Advantage: The changes file remains nicely short.

After this question loading sources will proceed as usual, registering any additions or modifications of class and method definitions in the new application. Now open an Application Browser for application ’InHouseSupport’ and let’s see what happened.

2 Major Versions

This section shows how to create your first 'major version' of an application. The operation can only be started from an Application Browser, so open one on 'InHouseSupport'. All further actions are executed in the context of this browser.

The main steps for creating a 'major version' are rather simple (see 2.5). We begin with the description of some preparing steps which often have to be executed first.

2.1 Maintaining Changes

The changes list (or simply: changes) of an application includes Change objects representing arbitrary modifications of class or method definitions or organisational steps (renaming method protocols, moving classes/methods between applications, etc.). A special case of Change objects in changes are 'backups'. Keep 'backups' referring to the global sources file or to applications other than the owner of a changes list. They are important when deinstalling applications.

When creating a 'major version' the elements of the changes are not written to the sources file, but they are remembered in the 'history'. Of course not all the elements must be remembered. It is really annoying if too much unimportant elements cover the few interesting ones. We want to encourage you to inspect the changes (frequently) and remove unnecessary elements. Especially when creating the very first 'major version' of an application, all change objects but 'backups' should be obsolete. The following steps treat this case.

- Select the menu item 'spawn> changes' on the category list of the browser to open an editor on the changes.

- Select in that editor menu item 'remove...> remove all'. All elements appear stroked through.

- Scroll through the changes editor and find an element of type 'backup' (beginning with '(*)'). Select it, activate the filter 'type', select menu item 'restore all' and deactivate the filter 'type'. Now anything but 'backup' elemente are stroked through.

- Select further elements of the list, if you want them to be remembered and undo the striking by selecting menu item 'restore it'.

- Select the menu item 'accept'. All elements which are not stroked through, will now be assigned as new value of the application's changes. Elements which might be invisible due to an active filter will also be transferred (unless they are marked for removal).

2.2 Defining Pre- and Post-Installation do its

'Pre-' und 'post-installation do its' are expressions which are executed before and after loading an application in order to integrate it in some environment and initialize it properly. We show the definition of 'post-installations' ('pre-installations' are made up the same way).

- Select the menu item 'spawn> post-installation' on the category list of the browser to open an editor on the 'post-installation do its'.

- Type in a valid Smalltalk expression in the text view of the editor (e.g. " Transcript show: 'installation completed' ") and save it using menu item 'accept' in the text view. The syntax of the expression will be checked; if there are any errors they will be treated as known from method browsers.

- A Dialog asks. whether the expression shall be inserted as 'do it'. Confirm with 'yes'.

- Select the menu item 'accept' in the menu of the list view of the editor and assign the contents of the editor to the 'post-installation' list of the application.

A well known problem is the definition of the order of class initializations. Likeas 'post-installation do its' (but before them) they are executed after installation of all classes and methods. ObjectWorks Smalltalk determines the order of class initializations automatically, and Application Management does not change this. But if you need some special order, you can achieve this by entering initialization statements in the 'post-installation do its'. For instance, let's say that the class 'Address' from application 'InHouseSupport' has to be initialized at last.

- Execute the four steps just explained, entering the expression "Address initialize" (without quotes) in the second step.

All expressions occurring in the 'post-installation do its' and matching the pattern <aClass initialize> effect, that class <aClass> will be initialized at the time defined by order of all the 'post-installation do its', rather than by the automatically generated order. Application Management uses a Smalltalk Scanner to recognize occurrences of the given pattern. So, the formulation is very free. But note, the following expressions do not achieve the goal!

- #(Address) do: [:aClass | aClass initialize]
  -> The scanner returns the tokens (... #aClass #initialize ...), but #aClass is not a class name.

- 'Address initialize'
  -> The scanner returns the single token #'Address initialize' and not the pair (#Address #initialize).

- "Address initialize"
  -> The scanner ignores comment and returns nothing.

If you want to prohibit the initialization, just use the simple trick and insert the expression

[Address initialize]
-> A valid statement, but without effect.

The whole procedure looks a bit peculiar on the first glimpse. But it is general, easy to understand, and avoids complex definitions of execution orderings. Moreover you need to specify only few special cases and let the system initialize the rest of classes by default.

2.3 Defining a MS-DOS Compatible Name

Normally you will identify applications with a self-explaining name. Because Application Management uses names of applications as basis of file names, there are problems on platform MS-DOS. To solve these problems, every application holds a dictionary for mapping arbitrary application names to platform compatible file names. Example:

- Select the menu item 'file in/out> set platform names' in the category list of an Application Browser on 'InHouseSupport'.
- A Dialog appears; enter 'ihsupprt' in the text field with the label 'dos'; press the button 'accept'.

  The dialog proposes compatible names for all supported platforms. If an entry is not defined the system provides one, trailed by a quotation mark to mark automatically gained entries. If you accept the values the quotation marks are removed automatically. Empty entries are left undefined. You may enter "what you like". The dialog will reject any input which does not specify a valid compatible file name. In practice, you might adapt the strategy of unix und Macintosh file systems when exporting files to MS-DOS: simply use the first eight letters ...

2.4 Defining Required Applications

This section shows how to define for an application A, that it requires installation of an application B to run properly.

- Select an arbitrary category in the category list of an Application Browser.
- Select the menu item 'include> required application'. A menu will offer the names of all registered applications.
- Select 'gsdummy'.

If no category is selected in the category list, the browser text will display that 'gsdummy' is now required for 'InHouseSupport'. Retracting such an entry is easy:

- Select the menu item 'spawn> required'. This opens an Inspector on all required applications.
- Select the field of the Inspector which has the value 'gsdummy'.

- Select ’remove’ from the Inspector menu and close the Inspector.

Application Management succeeds well in finding required applications automatically.

- Select an arbitrary category in the category list of an Application Browser.
- Select the menu item ’include> all required applications’. The operation takes some time, wait until the busy cursor disappears.

Deselect the category list selection and you will see again, that ’gsdummy’ is required for ’InHouseSupport’. The operation just described always adds, never removes required applications. Please refer to the manual, to get an overview how Application Management finds required applications. The result is surprisingly good, but not 100 %. You should know the kind of relationships which are not analysed and where you have to make manual corrections.

2.5 Creating a new ’major version’

Now that all the preparations have been completed, we finally create the new ’major version’.

- Select the menu item ’file in/out> new major version’.
- Confirm the question ’Do you really want to create a major version of application ’InHouseSupport’’ with ’yes’.
- Next you have to define the path, where the directory for application ’InHouseSupport’ will be created. Enter ’.’ in this case (or the name of an arbitrary existing directory). If necessary, create a new directory using a file manager or the shell.
- A dialog requests the name of the application. Enter ’InHouseSupport’. Normally you enter the name of the application as is. At this point, however, you have the (one and only!) possibility for renaming the application.
- Enter the new ’major version’ number. Normally you will accept the value proposed (= 1).
- Confirm the question whether to forget the ’history’ with ’yes’. Changing to a new ’major version’ the ’history’ is often not important.

The code and definitions of the application is now accessible for all other users. During major or minor version release Application Management associates new source code pointers to Compiled Methods registered in the application. Therefore you should save your image in order to save the new source code pointers. Leaving the image with 'Quit' has no impact on the application itself. It has been saved in its code and data files. But restarting the image there would be inconsistency with respect to the saved state of the application: The application as saved in the file system would be newer than in the image.

3 Minor Versions

’Minor versions’ define the incremental changes from a ’major’ or ’minor version’ to the next ’minor version’. We will now modify the demo applications to learn how Application Management reacts on adding and changing code. To get easily some effects without manual action, we prepared a file ’ihsupprt.st2’, which includes modified versions of methods registered in application ’InHouseSupport’.

- Open a FileBrowser (’file list’ from ’utilities’ menu of the Launcher)
- Select file ’ihsupprt.st2’ (probably preceded by the name of the directory where the tutorial sources reside)
- Select the menu item ’file in’
- Select ’InHouseSupport’ from the offered menu as target application.

Now loading of Smalltalk sources proceeds as usual, while logging all changes of class and method definitions to application ’InHouseSupport’. Open an Application Browser on application ’InHouseSupport’. First we want some further change on the application.

- Select category ’InHouse-Support’, class ’Licence’, protocol ’accessing’, and selector ’dateOfExpirationString’.
- Modify the code shown in the text view; for instance append a space and select ’accept’ from the text menu.
- Select the menu item ’spawn> changes’ from the category list menu and view the changes of the application (there should be four elements referring to class ’Licence’).

- Select the menu item ’remove> purge methods’ from the list menu of the Change Browser in order to cleanup from redundant elements (two items should be discarded).

- Select the menu item ’accept’ to reassign the non-discarded elements as current change list of the application (discarded elements disappear).

- Close the Change Browser.

Now that there are changes on the application we will finally create a new minor version.

- Select the menu item ’file in/out> new minor version’ from the category list menu.

The new state of the application is now accessible for all other users. Again you should save your image (see also remarks in section 2.5 ’Creating a new major version’).

4 Teamwork with Application Management

All users of an application share the same source code, reading the one and only source file. The right to write a 'minor' or 'major' version can be equivalenced to the write access on this source file. Only one of the users has write access. On unix platforms the file system guarantees the file protection via access right system. On other platforms this protection must be organized and watched by the user group itself. Regardless access rights Application Management rejects any attempt to write a new 'minor version', if the 'minor version' in the image is not consistent (less or equal) to the file saved state.

The manager of an application must be very careful with the source code file! Unless there are backup files (create them!) or individually copied files, there is no place keeping some replicate of the sources. You can't rely on changes files, cause loading sources with Application Management will not copy them and blow up changes files. The file must not be removed unless it is 100% sure that none of the users shares it. Editing the file is strictly forbidden, cause moving source code relative to the origin of the file will impose faulty source text access on any user. However, you are allowed to move the file, with the restriction that you move the entire directory tree of a major version. Example:

- Use a file manager or a shell and move the directory 'gsdummy' to '..'.
- Open an Application Browser on 'InHouseSupport' and select an arbitrary method of the application.
- The source code access fails and Application Management requests the new path. Enter '..'.

Of course you could have entered the new path selecting menu item 'file in/out> set path' of the Application Browser. The advantage of the first procedure is that Application Management detected that the original path is not valid any more. After entering the new path it notifies all applications to change their path also.

Every user may define local changes to an application. These changes are kept in the changes list of the application with the source code residing in the image changes file. They can easily passed over to the user which is responsible for the application. Example:

- Open an Application Browser on 'InHouseSupport'.
- Select the menu item 'file in/out> file out changes...'.
- Enter 'ihsupprt.cs' as file name for the source code export.
- Send the file 'ihsupprt.cs' to the manager of application 'InHouseSupport'.

  The receiver of the file will thank you, if the file does not contain "garbage" (i.e. redundant changes). Do him a favor, and clean the changes before filing them out.

The manager of the application will review the changes and integrate them appropriately. As soon as he writes out a new minor version, the new source code is public, i.e. accessible for every user in the net.

At any time you can run efficient tests to find out if some application installed in the image got new source since loading or last upgrade.

- Select 'InHouseSupport' in the Application Management Browser or open an Application Browser on 'InHouseSupport'.
- Select the menu item 'file in/out> upgrade...'.
- If the application has a minor version number less than the saved file state, a dialog will ask, up to which minor version number you want to upgrade.

  Usually you will accept the default value (latest minor version), but you may enter smaller version numbers. Example: Version 1.2 is installed in the image, version 1.5 is public, but you need only the state of 1.4. Enter '4' and anything defined after 1.2 and before 1.5 will be loaded.