14. FILE MANAGER

LOOPS data structures are fully integrated into Medley. This includes the definition of new File Manager commands so that any LOOPS object or method can be saved on files and loaded into the environment in exactly the same way that normal Medley data types are saved and loaded.

In addition, the LOOPS file browser provides a menu-driven interface to the File Manager. When using a LOOPS file browser, newly created objects are associated with files automatically. If you are not familiar with LOOPS file browsers see Chapter 10, Browsers.

This chapter describes the functions, methods, and variables used to load and store files containing LOOPS objects. It describes the File Manager commands related to LOOPS objects. It also describes how to add objects to files, delete them from files, and move them from file to file. These are primarily of interest when customizing either the File Manager or LOOPS file browser.

14.1 Manipulating Files

LOOPS takes advantage of the ability to create user-defined File Manager commands to fully integrate LOOPS into the Medley environment. As a result, the same steps used to manipulate files containing Medley data structures are used to manipulate files containing LOOPS data structures. Furthermore, both LOOPS and Medley data structures can be saved together in the same file. This section contains a brief review of the three basic functions used to manipulate files. For a more detailed description which includes additional functions, see the *Lisp Release Notes* and the *Interlisp-D Reference Manual*.

In addition, there is a LOOPS file browser which provides a convenient way of loading files and guaranteeing that newly created classes and methods are associated with files during the development of LOOPS programs. The LOOPS file browser is different from the Lisp Library Module **FILEBROWSER**. Files can be loaded and put into new or existing file browsers by a series of menu selections.

You can manipulate files with these basic steps:

- Assign data structures to a specific file using **FILES?**.
- Write data structures to a file using **MAKEFILE**.
- Enter data structures stored in a file into the environment using **LOAD**.

The following example shows these steps.

```
30←(FILES?)
PIPEANDTANK,LOOPSPRINT,LOOPSUTILITY...to be dumped.
   plus the instances: FFAV1,Datum1,TestW
   plus the class definitions: Datum
   want to say where the above go ? Yes
(instances)
FFAV1  File name: LOOPSFILE
create new file LOOPSFILE ? Yes
Datum1   File name: LOOPSFILE
TestW   File name: LOOPSFILE
(class definitions)
Datum   File name: LOOPSFILE
NIL

31←(MAKEFILE 'LOOPSFILE)
Copyright owner for file LOOPSFILE: XEROX
{DSK}<LISPFILES>LOOPSFILE.;1

32←(LOAD 'LOOPSFILE)
{DSK}<LISPFILES>LOOPSFILE.;1
FILE CREATED  7-Jan-87 16:25:24
LOOPSFILECOMS
{DSK}<LISPFILES>LOOPSFILE.;1
```

See the *Lisp Release Notes* and the *Interlisp-D Reference Manual* for more information on **FILES?** and **MAKEFILE**. See the following section for details on **LOAD**.

14.2 Loading Files

The following table shows the functions and commands described in this section.

Name	Type	Description
LOAD	Function	Loads Medley symbolic files which includes LOOPS objects and methods.
LOADFNS	Function	Allows selective loading from Medley symbolic files.
UNDO	Prog. Asst.	Undoes previous entries into the Medley Executive which are stored on a history list, including calls to **LOAD**.

(**LOAD** *FILE LDFLG*) [Function]

Purpose/Behavior: Loads Medley symbolic files which includes all LOOPS objects and methods; see the *Lisp Release Notes* and the *Interlisp-D Reference Manual*.

Arguments:

FILE File to be loaded.

LDFLG Alters the effect of loading a file.

- If it is set to **PROP**, the definitions of functions, including **METHOD** functions, are stored on the property **EXPR** of the function name. Thus, any existing definitions are not overwritten.

- If it is set to **ALLPROP**, the values of variables are also saved on property lists.

Returns: Full file name.

(**LOADFNS** *FNS FILE*) [Function]

Purpose/Behavior: Allows selective loading from Medley symbolic files including LOOPS files . The most likely use for this facility is to load the source code for method functions when the compiled versions are already loaded. The methods must be specified by their explicit function names in the form **ClassName.Selector**, for example,

```
(LOADFNS '(SomeClass.AMethod OtherClass.AMethod) '{DSK}<LISPFILES>SOMEFILE 'PROP)
```

It is not recommended that LOOPS objects be selectively loaded by using **VARS** (see the *Lisp Release Notes* and the *Interlisp-D Reference Manual*), because it is not possible to guarantee that all necessary related objects, such as superclasses or methods of a class, are also loaded.

Arguments: *FNS* Selected functions to be loaded.

FILE File from which functions specified in **FNS** are to be loaded.

Returns: List of functions that have been loaded

UNDO [Program Assistant Command]

Purpose/Behavior: LOOPS saves enough information about objects that are created as a result of loading a file to allow the call to **LOAD** to be undone. The objects are destroyed and any preexisting objects that were deleted by the load are restored. See the *Lisp Release Notes* and the *Interlisp-D Reference Manual*.

14.3 LOOPS FILE PACKAGE COMMANDS

14.3 LOOPS File Manager Commands

Four File Manager types are defined to allow LOOPS objects to be stored in Medley files:

- CLASSES
- METHODS
- INSTANCES
- THESE-INSTANCES

These types and the functions and methods used by LOOPS to process these types are described in this section.

Note: The order of items in the filecoms is important. In particular, class definitions must appear in the file before any methods on that class or any instances of that class. Similarly, methods on a class must appear before any instances of that class.

Name	Type	Description
CLASSES	File Mgr Command	Writes the appropriate **DEFCLASSES** and **DEFCLASS** expressions for the named classes.
DEFCLASSES	NLambda NoSpread	Creates a series of empty classes in preparation for reading their definitions via **DEFCLASS**.

DEFCLASS	NLambda NoSpread	Takes a source specification of a class from a file and causes the appropriate internal representation to be constructed.
METHODS	File Mgr Command	Writes the appropriate **METH** and **DEFINEQ** expressions for each method object and its associated function.
METH	NLambda NoSpread	Creates a method object and attaches it to the appropriate class.
INSTANCES	File Mgr Command	Writes the appropriate **DEFINST** expressions for each instance in the list.
THESE-INSTANCES	File Mgr Command	Appears as a sublist in a filecoms.
DEFINSTANCES	NLambda NoSpread	Creates empty structures for each instance name in a list.
DEFINST	NLambda NoSpread	Creates internal representations for source specifications of an instance.
FileIn	Method	Creates internal representations for source specifications of an instance.

(**CLASSES** *ClassName1...ClassNameN*) [File Manager Command]

Purpose/Behavior: Appears as a sublist in a filecoms. The keyword **CLASSES** tells the File Manager to use the appropriate **DEFCLASSES** and **DEFCLASS** expressions for the named classes when writing to a file.

Arguments: *ClassName* Accepts any symbol, but only gives meaningful result when you use **DEFCLASS** to actually create the class.

Example: `(CLASSES Myclass)`

(**DEFCLASSES** *CLASSES*) [NLambda NoSpread Function]

Purpose/Behavior: Used in a file to create a series of empty classes in preparation for reading in their definitions via **DEFCLASS**. This allows the classes to be read in any order. Otherwise, superclasses would have to be read in before their subclasses.

Arguments: *CLASSES* Accepts any symbol, but only gives meaningful result when you use **DEFCLASS** to actually create the class.

Returns: NIL

Example: The command

```
(DEFCLASSES MyClass)
```

returns `NIL`.

(**DEFCLASS** *FORM*) [NLambda NoSpread Function]

Purpose/Behavior: Takes a source specification of a class, such as produced by the method **MakeFileSource**, from a file and causes the appropriate internal representation to be constructed.

Arguments: *FORM* The source specification of a class.

Returns: NIL

Example:
```
(DEFCLASS MyClass
(MetaClass Class doc (* Something for my project)
          Edited:  (* nbm "18-Oct-87 13:20"))
(Supers Object)
(InstanceVariables ( Iv1 (22) doc
          (* Initial value for my instances)]
```

(**METHODS** *ClassName.Message1...ClassName.MessageN*) [File Manager Command]

Purpose/Behavior: Appears as a sublist in a filecoms. The keyword **METHODS** tells the File Manager to use the appropriate **METH** and **DEFINEQ** expressions for each method object and its associated function.

Arguments: *ClassName.Message*
The source specification of a class.

Example:
```
(METHODS  MyClass.Method1)
```

(**METH** *methDescr*) [NLambda NoSpread Function]

Purpose/Behavior: Creates a method object and attaches it to the appropriate class.

Arguments: *methDescr* Method object to create.

Returns: NIL

Example:
```
(METH MyClass  MyClass.Method1 NIL
(category (Datum)))
```

(**INSTANCES** *InstName1...InstNameN*) [File Manager Command]

Purpose/Behavior: Appears as a sublist in a filecoms. The keyword **INSTANCES** tells the File Manager to use the appropriate **DEFINST** expressions for each instance in the list and also for any other instances that are referenced inside any instances in the list. This assures that there are no references to nonexistent instances when read back in. The method **SaveInstance?** can be specialized to prevent instances from being saved in more than one file when they are referred to by instances in different files.

Example:
```
(INSTANCES  TestW)
```

(**THESE-INSTANCES** *InstName1...InstNameN*) [File Manager Command]

Purpose/Behavior: Appears as a sublist in a filecoms. The keyword **THESE-INSTANCES** tells the File Manager to use the appropriate **DEFINST** expressions for each instance in the list. Unlike the **INSTANCES** File Manager command, **THESE-INSTANCES** does not recursively dump instances that are pointed by *InstName1...InstNameN*.

(**DEFINSTANCES** *Instances*) [NLambda NoSpread Function]

Purpose/Behavior: Takes a list of instance names and creates empty structures for them in preparation for reading in their structures from a file.

Arguments: *Instances* Accepts any symbol but result is useless unless you use **DEFINST** to actually create the *Instance*.

Returns: NIL

Example:
```
(DEFINSTANCES TestW)
```

(**DEFINST** *DEFINST% FORM*) [NLambda NoSpread Function]

Purpose/Behavior: Takes a source specification of an instance and causes the appropriate internal representation to be created. It does this by sending the message **FileIn** to the instance's class. It creates the class if it does not exist.

Arguments: *DEFINST% FORM*
The source specification of an instance.

Returns: NIL

Example:
```
[DEFINST Window
  (TestW (JEW0.0X:.H<4.NZ9 . 532))
  (left 179)
  (bottom 446)
  (width 12)
  (height 12)]
```

(← *self* **FileIn** *fileSource*) [Method of Class]

Purpose/Behavior: Takes a source specification for an instance as it appears in a file and causes the appropriate internal representation to be constructed.

Arguments: *self* Class of the instance to be created.

fileSource Loadable form of an instance as stored in a file.

Returns: *self*

Categories: Class

14.4 SAVING LOOPS OBJECTS ON FILES

14.4 SAVING LOOPS OBJECTS ON FILES

14.4 Saving LOOPS Objects on Files

Adding LOOPS classes, methods and instances to files can be done in the same way that functions and variables are saved in Medley. In addition, the LOOPS browser allows newly created objects to be automatically associated with files. LOOPS also provides the means for moving objects from file to file.

Whenever a class, method, or named instance is created or edited, it is marked as changed. This allows the File Manager to prompt for a file in which to store new objects and see to it that changed objects are written out when **MAKEFILE** is called.

The following table shows the items in this section.

Name	Type	Description
FILES?	Function	LOOPS adds a prompt for classes, methods and instances along with the normal Medley types.
ObjectModified	Method	Notifies the File Manager that an object has been changed or created.
OnFile	Method	Determines if a class is in **FILELST**.
SaveInstance	Method	Causes newly created instances to be noticed by the File Manager.
SaveInstance?	Method	Determines if an instance needs to be added to the list of instances to be saved.
DelFromFile	Method	Deletes an object from any file in **FILELST** in which it appears.
MoveToFile	Method	**Class**.**MoveToFile** moves a class and its methods from one file to another. **Object**.**MoveToFile** moves an instance from one file to another.
MoveToFile!	Method	Moves a class, all of its methods, and all of its subclasses and their methods from one file to another.
DontSave	IVProperty	Controls what parts of an instance are saved in a file.
OldInstance	Method	Sends a message to an object after it is loaded from a file.

(**FILES?**) [Function]

Purpose/Behavior: The File Manager types have been extended so that, when a call is made to **FILES?**, you are prompted to add classes, methods and instances to files along with the normal Medley. For an example of **FILES?**, see Section 14.1, "Manipulating Files."

After a class is associated with a file, any methods that are added to it are automatically added to that file as well. Thus, it makes sense to put classes in files as soon as possible. This could be done by repeated calls to **FILES?**, but the LOOPS file browser allows classes to be automatically added to files as they are created. Any class that is created by adding a root to a file browser or by specializing a class in a file browser is added to that brower's file. If more than one file is associated with the browser, a menu appears to prompt you to specify a file for the new class. The LOOPS browser also can be used to create a new file and associate it with a file browser. Thus, there is never any need to wait until the end of a session to put classes and methods in files.

You can also save instances on files. Of course, only those instances which should be present after a file is first loaded should be saved. Instances which are constructed "on the fly" as a consequence of running a LOOPS program should not be saved. Only named instances are marked as changed so many such temporary instances may never be noticed. However, if named instances which should not be saved are created, then you are prompted to put them into files after a call to **FILES?** and must respond by typing a right square bracket (]) to each one. Alternatively, it is possible to specialize the method **ObjectModified** so that it does not call **MARKASCHANGED**. Then any instances of classes which have or inherit the specialized method are not noticed by the File Manager regardless of whether or not they are named.

(← *self* **ObjectModified** *name*) [Method of Object]

Purpose: Notifies the File Manager that an object has been changed or newly created.

Behavior: Uses the File Manager command **MARKASCHANGED**. It does nothing if *name* is not given, thus unnamed objects are never marked.

Arguments:
- *self* — A LOOPS object.
- *name* — Name of object specified in *self*.
- *reason* — Reason is **MARKEDASCHANGED** (see the *Interlisp Reference Manual* for information on **MARKEDASCHANGED**).

Returns: *self*

Categories: Object

Specializations: Method

(← *self* **OnFile** *file*) [Method of Class]

Purpose: Determines if an object is in a file in **FILELST**.

Behavior: Calls **WHEREIS** (see the *Lisp Release Notes* and the *Interlisp-D Reference Manual*).

- If *file* is not given, it returns the name of the file in **FILELST** that the object is contained in or NIL if *self* is not in a file.
- If *file* is given, it must still be a member of **FILELST**, and T or NIL is returned.

Arguments:
- *self* — A LOOPS object.
- *file* — The file to be searched.

Returns: Value depends on the arguments; see **Behavior**.

Categories: Class

(← *self* **SaveInstance** *name reason*) [Method of Object]

Purpose: Causes newly created instances to be noticed by the File Manager.

Behavior: Sends *self* the message **ObjectModified**.

Arguments:
- *self* — A LOOPS object.
- *name* — Name of object specified in *self*.
- *reason* — Reason is **MARKEDASCHANGED** (see the *Interlisp Reference Manual* for information on **MARKEDASCHANGED**).

Returns: *self*

Categories: Object

(← *self* **SaveInstance?** *file outInstances*) [Method of Object]

Purpose: Determines whether an instance needs to be added to the list of instances to be saved in *file*.

Behavior: Checks to see if the current instance is a member of *outInstances*. It is used by the LOOPS File Manager command **INSTANCES** to guarantee that the same instance does not appear more than once in a given file.

This method must be specialized to be used; it cannot be used directly by the user.

Arguments: *self* A LOOPS object.
file The file to be searched.
outInstances A list of LOOPS names. See Behavior.

Returns: T if the instance should be saved on the file; NIL if it should not be saved.

Categories: Object

(← *self* **DelFromFile**) [Method of Object]

Purpose: Deletes an object from any file in **FILELST** in which it appears.

Behavior: Searches through the filecoms of all files in **FILELST** and deletes the object everywhere it appears.

Arguments: *self* A LOOPS object.

Returns: Used for side effect only.

Categories: Object

Specializations: Class, Method

(← *self* **MoveToFile** *file*) [Method of Class]

Purpose: Moves an object from one file to another. If an object is a class, it, and all its methods, move.

Behavior: Adds the object to the filecoms of *file* so that the object will be saved on that file. If *file* is NIL, it prompts for a file form **FILELST** via a menu.

Arguments: *self* A class or method.
file File to which object is moving.

Returns: NIL

Categories: Object

Specializes: Object

(← *self* **MoveToFile!** *file fromFiles*) [Method of Class]

Purpose: Moves a class, all of its methods, and all of its subclasses and their methods from one file to another.

Behavior: Similar to **MoveToFile**.

Arguments: *self* A LOOPS class.
file File to which object is moving.
fromFiles A list of files from which classes may be moved.

Returns: NIL

Categories: Class

DontSave [IV Property Name]

Purpose/Behavior: Controls what parts of an instance are saved in a file. Its value is a list of property names of the instance variable which should not be written out when

the instance is dumped. If **Value** is in the list, the instance variable's value is not saved. If the property is **Any**, nothing is saved except the instance variable name. (Must be added by the user.)

(←*self* **OldInstance** *name arg1 arg2 arg3 arg4 arg5*) [Method of Object]

Purpose: Sends a message to an object after it is loaded from a file. This method can be specialized by applications that need to perform some operation on every object when it is created.

Behavior: If *name* is non-NIL, the message **SetName** is sent to *self*.

Instance variables with an **:initForm** property are filled. See the discussion of **:initForm** in Chapter 2, Instances.

Sends the message **SaveInstance** to *self* with the arguments *name*, *arg1*, and *arg2*.

Arguments:

self	Evaluates to a class.
name	LOOPS name of the class or instance.
arg1...arg5	Optional arguments referenced by user-written specialization code.

Categories: Object

Specializations: IndexedObject

14.5 STORING FILES

14.5 Storing Files

This section describes the functions and methods used by LOOPS and Medley to store files.

Name	Type	Description
MAKEFILE	Function	Writes files that contain Medley data types which include LOOPS objects and methods.
PrettyPrintClass	Function	Prints classes in a file in a form that can be read back in.
PrettyPrintInstance	Function	Prints instances in a file in a form that can be read back in.
MakeFileSource	Method	Constructs the representation of an object that is appropriate for printing in a file.
FileOut	Method	Controls the printing of a LOOPS object in a file.

(**MAKEFILE** *FILE*) [Function]

Purpose/Behavior: When all LOOPS objects are associated with their files, the files are written by a call to **MAKEFILE** or **MAKEFILES**. This is identical to the standard use of **MAKEFILE** in Medley. See the *Lisp Release Notes* and the *Interlisp-D Reference Manual*.

Arguments:

FILE	Name of file to be written out.

Returns: Full file name

(**PrettyPrintClass** *className file*) [Function]

Purpose/Behavior: Used by the File Manager command **CLASSES** to print out classes in a file in a form that can be read back in. It checks to make sure the class exists and then sends it the message **FileOut**. It is also used by the method **PP** to print classes to a display stream.

Arguments: *className* The name of the class to be printed on the file *file*.

file The file on which the class *className* is to be printed.

Returns: Pointer to class in the form #,($ className)

(**PrettyPrintInstance** *instanceName file*) [Function]

Purpose: Used by the File Manager command **INSTANCES** to print instances in a file in a form which can be read back in. Sends the message **FileOut** to instance.

Arguments: *instanceName*
Name of a LOOPS instance.

file The file on which the instance i*nstancename* is to be printed.

Returns: NIL

(← *self* **MakeFileSource** *file*) [Method of Object]

Purpose: Constructs the representation of an object that is appropriate for printing in a file.

Behavior: Uses the relevant access functions to obtain the parts of the object and then stores them into a list structure.

Arguments: *self* A LOOPS object.

file The file on which *self* is to be printed.

Returns: Loadable form of a LOOPS object.

Categories: Object

Specializations: Class, Method

Example:
```
63←(← ($ TestW) MakeFileSource)
        (DEFINST Window
        (TestW (NEW0.1Y%:.;h.eN6 . 501)))
```

(← *self* **FileOut** *file*) [Method of Object]

Purpose: Controls the printing of a LOOPS object in a file.

Behavior: Gets the appropriate source representation by sending the object the message **MakeFileSource** and prettyprints the result.

Arguments: *self* A LOOPS object.

file The file on which self is to be printed on if T prints to the Lisp Executive window.

Returns: *self*

Categories: Object

Specializations: Class, Method

Example:

```
62_(_ ($ TestW) FileOut T)
(DEFINST Window (TestW (NEW0.1Y%:.;h.eN6 . 501)) )
#,($& TestW (NEW0.1Y%:.;h.eN6 . 501))
```


14.6 COMPILING FILES

14.6 Compiling Files

LOOPS uses the new XAIE compiler and its macrolet facilities. When doing **CLEANUP** on LOOPS files your ***DEFAULT-CLEANUP-COMPILER*** should be set to 'CL:COMPILE-FILE. More information on this cleanup flag and the new compiler are available in the *Lisp Release Notes*.

[This page intentionally left blank]