# A. ACTIVE VALUES IN BUTTRESS LOOPS

In the Buttress version of LOOPS, the concept of active values was implemented differently. The current **ExplicitFnActiveValue** acts very much like the old active values, and is used to provide compatibility with existing Xerox LOOPS code. Most of the functions described in this chapter are found only in the LOOPSBACKWARDS user package, work as they did in the Buttress version of LOOPS, and should be used only to bring existing code into the current system.

Descriptions of the functionality in this appendix are written in terms of the new **ActiveValues** wherever possible.

The active value/annotated value system discussed in Chapter 8, Active Values, is a new implementation. Programs developed using the Buttress activeValue system automatically convert into the new system when loaded, using the **ExplicitFnActiveValue** capability described in Chapter 8, Active Values, and in this appendix.

Note: The following functions and records are maintained for compatibility purposes only; they are not fully supported and may not exist in future Xerox LOOPS releases. Programs that use these records and functions should be changed. The LOOPSBACKWARDS user package must be loaded for these functions to work.

## A.1 Buttress System of ActiveValues

The Buttress LOOPS implementation combined the notions of annotated value and active value. To annotate a variable, the value was replaced with an instance of an Interlisp-D data type called activeValue, but there were no LOOPS classes with similar names and functions as there are now.

**activeValue** [Record]

Purpose: Buttress implementation of the active values concept. Specifically, the Lisp data type equivalent to the present annotatedValue.

Behavior: An activeValue placed as the value of a variable invoked evaluation of code on access attempts rather than just returning a stored value.

Field Names:

**localState** A place for data storage.

**getFn** The name of a function applied when the program retrieves the value of a variable that contained an activeValue.

**putFn** The name of a function that was applied when the program replaces the value of a variable that contained an activeValue.

If either the **getFn** or **putFn** fields is NIL, default actions returned or replaced the **localState**, respectively. Nesting was accomplished by the **localState** of an activeValue being itself an activeValue.

**ExplicitFnActiveValue** [Class]

Purpose: Mimics the behavior of the Buttress-style active values and allows simple changes to the user code triggered by the **ActiveValue** mechanism.

Behavior: **Get** accesses to the wrapped variable cause the **getFn** to be called, and **Put** accesses cause **putFn** to be called. Enables the old style activeValues to look like the new style without changing any functionality.

Instance Variables:

**localState** A place for data storage.

**getFn** The name of a function applied when the active variable is read.

**putFn** The name of a function applied when the active variable is changed.

(**MakeActiveValue** *self varOrSelector newGetFn newPutFn newLocalState propName type*) [Function]

Purpose: Makes the value of some variable an active value.

Behavior: Creates a new activeValue record and installs it according to the arguments.

Arguments:

*self* Object whose variable is changed to an active value.

*varOrSelector*
Variable name or method selector where the data type activeValue is placed.

*newGetFn* and *newPutFn*
If NIL, the old values of **getFn** and **putFn** are not overwritten. If T, the values of **getFn** and **putFn** are changed to NIL. Any other values are placed in the **getFn** and **putFn** fields of the activeValue.

*newLocalState*
The value of this argument is ignored. A new **ActiveValue** instance is always created. The contents of **localState** is changed to the previous value of the variable or property being made active.

*propName* Name of the property, if the active value is to be placed on a property list. This is NIL if the active value is associated with a variable or method.

*type* Indicates the type of the variable *varNameOrSelector*. Must be one of IV (or NIL) for instance variable, CV for class variable, CLASS for a class property, or METHOD for a method property.

(**DefAVP** *fnName putFlg*) [Function]

Purpose: Creates a template for defining an active value function.

Behavior: Creates a template and leaves you in the Interlisp-D function editor.

Arguments:

*fnName* Name of the function.

*putFlg* T indicates function is a **putFn**; NIL indicates a **getFn**.

Returns: The function name on exit from the editor.

(**GetLocalState** *activeValue self varName propName type*) [Function]

Purpose: Retrieves data from **localState**.

Behavior: Retrieves the value stored in the **localState** of *activeValue*. Nested active values will be triggered.

Arguments:
- *activeValue* An **ActiveValue**.
- *self* The object containing the **ActiveValue**.
- *varName* The name of the variable were the **ActiveValue** is stored.
- *propName* The name of an instance or class variable property. This is NIL if the **ActiveValue** is associated with the value of the variable itself.
- *type* Specifies where the **ActiveValue** was stored. NIL means an instance variable, CV means class variable, CLASS means a class property, METHOD means a method property.

Returns: Contents of the **localState** field of *activeValue*.

(**PutLocalState** *activeValue newValue self varName propName type*) [Function]

Purpose: Data replacement.

Behavior: Stores *newValue* in the **localState** field of *activeValue*. Nested active values will be triggered.

Arguments:
- *activeValue* An **ActiveValue**.
- *newValue* A new value to be stored in **localState**.
- *self* The object containing the **ActiveValue**.
- *varName* The name of the variable were the **ActiveValue** is stored.
- *propName* The name of an instance or class variable property. This is NIL if the **ActiveValue** is associated with the value of the variable itself.
- *type* Specifies where the **ActiveValue** was stored. NIL means an instance variable, CV means class variable, CLASS means a class property, METHOD means a method property.

Returns: The value of *newValue*.

(**GetLocalStateOnly** *activeValue*) [Function]

Purpose: Gets a value from **localState** without triggering any nested **ActiveValue**.

Behavior: Retrieves the value stored in the **localState** field of the **ActiveValue** without triggering any nested **ActiveValue**.

Arguments:
- *activeValue* The **ActiveValue** in which the **getFn** and **putFn** is found.

Returns: The contents of **localState.**

(**PutLocalStateOnly** *activeValue newValue*) [Function]

Purpose: Puts a value into a **localState** without triggering any nested **ActiveValues**.

Behavior: Replaces the value stored in the **localState** of *activeValue* without triggering any nested **ActiveValue**.

Arguments: *activeValue* An **ActiveValue**.

*newValue* Value used for the replacement.

Returns: The value of *newValue.*

(**ReplaceActiveValue** *activeValue newVal self varName propName type*) [Function]

Purpose: In an object's variable which has an **ActiveValue** installed, overwrites *activeVal* with *newVal*, providing a way of removing an **ActiveValue**.

Behavior: Searches arbitrarily deep nesting to replace the occurrence of *activeVal* with *newVal*. If no match is found in the list that is the value of the variable described by the arguments, an error is invoked.

Arguments: *activeValue* The **ActiveValue** to be replaced.

*newVal* A new value to be stored in the object's variable.

*self* The object containing the **ActiveValue**.

*varName* The name of the variable were the **ActiveValue** is stored.

*propName* The name of an instance or class variable property. This is NIL if the **ActiveValue** is associated with the value of the variable itself.

*type* Specifies where the **ActiveValue** was stored. NIL means an instance variable, CV means class variable, CLASS means a class property, METHOD means a method property.

*newValue* Value used for the replacement.

Returns: Value of *newVal.*



A.2 GETFNS AND PUTFNS

## A.2 getFns and putFns

In the Buttress version of LOOPS, where the only kind of active value was equivalent to **ExplicitFnActiveValue**, specialization of active values was done not the way it is in Xerox LOOPS, but by the equivalent of putting special purpose functions into the **getFn** and **putFn** instance variables. The following functions emulate the behaviors they had in the Buttress version, using the current **ActiveValue** mechanisms.

In all cases, the functions are installed in the **getFn** or **putFn** instance variable of an **ActiveValue**, and are called when an attempt is made to get or put the variable where the **ActiveValue** is stored. The arguments and values returned are irrelevant to the use of these functions.

(**NoUpdatePermitted** *self varName oldOrNewValue propName activeValue type*) [Function]

Purpose: **putFn** for preventing the updating of a variable.

Behavior: LOOPS-defined **putFn** that causes a break if an attempt is made to replace the value of the variable containing the **ActiveValue**.

(**FirstFetch** *self varName oldOrNewValue PropName activeValue type*) [Function]

Purpose: **getFn** for dynamic variable initialization.

Behavior: LOOPS-defined **getFn** that expects the **localState** of *activeValue* to be an Interlisp-D expression to be evaluated. On the first fetch, the expression is evaluated and the variable or property is set to the value of the expression.

(**GetIndirect** *self varName oldOrNewValue PropName activeValue type*) [Function]

Purpose: LOOPS-defined **getFn** that functions as a pointer to another variable.

Behavior: **GetIndirect** and **PutIndirect** together set up an **ActiveValue** whose **localState** contains a pointer to where the actual value is stored. This is used when the value of a variable should always be the same as another.

(**PutIndirect** *self varName oldOrNewValue PropName activeValue type*) [Function]

Purpose: LOOPS-defined **putFn** that functions as a pointer to another variable.

Behavior: See **GetIndirect**.

(**ReplaceMe** *self varName oldOrNewValue PropName activeValue type*) [Function]

Purpose: LOOPS-defined **putFn** which removes both itself and any **getFn**.

Behavior: In some cases, you may want to compute a default value if given, but replace the active value by the value given if you set the value of a variable. For this, you can employ **ReplaceMe**. Any replacement attempt at the variable containing an **ActiveValue** with this as its **putFn** results in the value of the variable being replaced, and the **ActiveValue** disappearing.

(**AtCreation** *self varName oldOrNewValue PropName activeValue type*) [Function]

Purpose: LOOPS-defined **getFn** used to replace the active value with a dynamically computed value at instance creation.

Behavior: This function no longer works.

To achieve the closest functionality, use the **FirstFetchAV** specialization of the class **ActiveValue** (see Chapter 8, Active Values) or the **FirstFetch** function described above.

[This page intentionally left blank]