



---

## **Static Instances Initialization in UML Foundation for C**

Version 2.6

September 26, 2003

---

### *PathMATE Technical Notes*

Pathfinder Solutions LLC  
33 Commercial Drive, Suite 2  
Foxboro, MA 02035 USA  
[www.PathfinderMDA.com](http://www.PathfinderMDA.com)  
888-662-7284

# Table Of Contents

- 1. Introduction.....1**
- 2. Analysis Conventions.....1**
  - Class.....1
  - Association.....2
- 3. Instance Data Specification.....2**
  - Spreadsheet-Based Data.....2
- 4. Template Changes and Extensions.....6**
  - Data Schema Generation.....6
  - External Tracking Structure for "Many" Side of Association.....6
  - Static Instance Module.....7
- 5. Additional Example.....9**

## 1.Introduction

This Technical Note describes the capabilities to be deployed in support of static instance initialization in the UML Foundation for C implementation architecture. It describes the model conventions required, coloring properties used, how spreadsheet-based instance data is structured, and requirements for the generated static initializers.

The benefits of static initialization of instances include:

- Instance setup happens with program loading, and it likely to be faster than the execution of setup code.
- The instance information is stored in the program image, eliminating the need for a separate persistence or message mechanism

Based on Pathfinder's extensive field experience, specific initialization needs will vary substantially from one customer project to the next. Therefore it is expected that the base solution described herein will form a base for those project-specific customizations.

## 2.Analysis Conventions

### Class

The new coloring property "StaticPopulation" will be defined for classes with a default value of "FALSE". A class with a "StaticPopulation" value of "TRUE" will be considered *static*, and cause code to be generated to support static initialization of the class instance population. If "StaticPopulation" == "TRUE" the class instance data for that class will be held in an array. If the "MaxIndex" property is specified it will be used to determine of entries in this array, otherwise the array will be automatically sized to fit the number of instances specified in the static instance data.

If "StaticPopulation" == "const", then the static instance data will be specified with a const qualification. This data is considered read-only, and any PAL operations writing to these instances are prohibited. Spotlight cannot run with const classes (agent instrumentation must connect at runtime) so if "SpotlightEnabled" == "T" for the domain containing "const" classes, the "StaticPopulation" property for these classes is forced to "TRUE".

### Class Modeling Conventions/Rules

- If a value for the "MaxIndex" property is specified, it is up to the analyst/designer to ensure that this value is at least as large as the number of instances specified in the static instance data.
- Active classes will not specify their initial state explicitly in the instance data – they are required to specify their initial state using the initial state pseudostate on their state diagram.
- The SortKey property and the SORT action language directive are not supported for static classes.

- If any class in a subtype/supertype hierarchy is static, then all classes in the hierarchy must be static. (Supertypes with static subtypes automatically have their "StaticPopulation" values set to "TRUE".)
- CREATE and DELETE operations, lifecycles (state models), and attribute write accesses are not supported on "const" classes.

## Association

Like the class, the association will enjoy a new coloring property "StaticPopulation" with a default value of FALSE. An association with a "StaticPopulation" value of "TRUE" will be considered *static*, and cause code to be generated to support static initialization of the association population.

If an association has a "many" participant that is a statically initialized class, then that association either must have a full population in the static instance data, or specify a MaxIndex large enough for all participating instances (dynamic or static). In other words, an association requires a MaxIndex value if:

- One or both participants are "many" and are statically initialized
- A link for the association is not statically specified for each instance of the "many"/static participant.

### Association Modeling Conventions/Rules

- Every *static* association must have at least one role phrase.
- The SortKey property and the SORT action language directive are not supported for static associations.
- LINK and UNLINK operations are not supported on associations with a participant that is a "const" class.

## 3.Instance Data Specification

### Spreadsheet-Based Data

The instance data is assumed to be in a spreadsheet per domain saved in CSV format, named with <system\_name>\_<domain\_name>.csv, and following these layout conventions:

<system name>.<domain  
name>

CLASS INFORMATION:

<class name>	<attribute name>	<attribute name>	<attribute name>
1	<attribute value>	<attribute value>	<attribute value>
2	<attribute value>	<attribute value>	<attribute value>
...			
N	<attribute value>	<attribute value>	<attribute value>

<other class info>

ASSOCIATION  
INFORMATION:

<association name>	<participant 1 class name> (role phrase, if any)	<participant 2 class name> (role phrase, if any)	<association class name, if any>
	<participant index>	<participant index>	<assoc class index>
	<participant index>	<participant index>	<assoc class index>
...			
	<participant index>	<participant index>	<assoc class index>

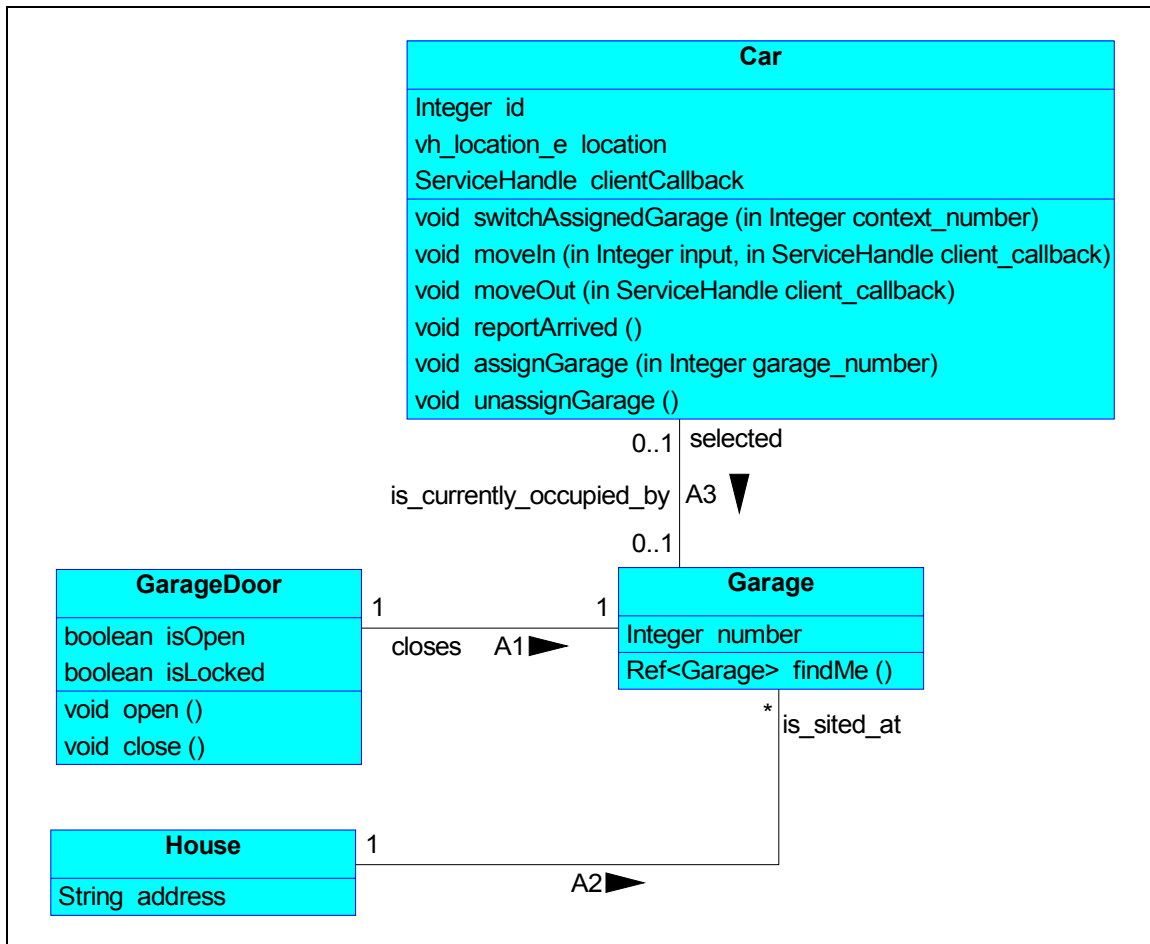
<other association info>
--------------------------

**Table 1: Instance Data Spreadsheet Template**

The following conventions apply when building an instance data spreadsheet:

- The first column of class information contains a 1-based array index, used primarily to identify the class instance in subsequent association link rows. The user must ensure that these index values start at 1, are unique for each instance, and that no holes in the array are created by skipping values. The instances can appear in any order in the class' table.
- Comments describing a class instance row can be specified as a string starting with "#" in the same row as the instance data in the column immediately after the last attribute value (please see example). Similar commenting is available for 1:m or m:m associations (comments in 1:1 associations will not appear in the generated code).
- A blank or comment line (comment in column 1) marks the end of a class' instances.
- All tables of class instance information must appear before any association link information
- If an association has only one role phrase, those instance indices must appear in the second column (as opposed to the third)
- In an association link table column where the participant is a supertype class, the specific subtype instance is specified by <subtype class prefix> <instance index>.
- A blank row or comment (in column 1) must appear immediately after the last instance of a class, or the last link of an association.
- If a literal string value is specified (via '"') then no spaces may appear within the cell outside of the quotes (the spaces could only be inserted via a text editor outside of Excel).

The example below specifies the instance population for an updated version of the CarShuffle example system VehicleHousing domain:



**Figure 1: CarShuffle VehicleHousing Classes**

CarShuffle.Vehicle Housing

CLASS INFORMATION:

<i>Car</i>	<i>clientCallback</i>	<i>id</i>	<i>location</i>
1	EMPTY_SERVICE_HANDLE	1	VH_LOCATION_GARAGE
2	EMPTY_SERVICE_HANDLE	2	VH_LOCATION_GARAGE
3	EMPTY_SERVICE_HANDLE	3	VH_LOCATION_GARAGE
4	EMPTY_SERVICE_HANDLE	4	VH_LOCATION_GARAGE
5	EMPTY_SERVICE_HANDLE	5	VH_LOCATION_GARAGE

# This is a comment describing the GarageDoors

<i>GarageDoor</i>	<i>isLocked</i>	<i>isOpen</i>
-------------------	-----------------	---------------

1	FALSE	FALSE	# Comment for first door
2	FALSE	FALSE	
3	FALSE	FALSE	
4	FALSE	FALSE	
5	FALSE	FALSE	

<i>Garage</i>	<i>number</i>
1	1
2	2
3	3
4	4
5	5

<i>House</i>	<i>address</i>
1	"12 Broad Street, Nashua, NH 22303"
2	"1 Main Street, Wrentham, MA 02093"

# This is a general comment

#### ASSOCIATION INFORMATION:

<i>A1</i>	<i>GarageDoor (closes)</i>	<i>Garage</i>
	1	1
	2	2
	3	3
	4	4
	5	5

<i>A3</i>	<i>Car (selected)</i>	<i>Garage (is_currently_occupied_by)</i>
	1	1
	2	2
	3	3
	4	4
	5	5

<i>A2</i>	<i>Garage (is_sited_at)</i>	<i>House</i>	
	1	1	# At NH location

2	1	# At NH location
3	2	# At MA location
4	2	# At MA location
5	2	# At MA location

**Table 2: CarShuffle\_VehicleHousing.csv**

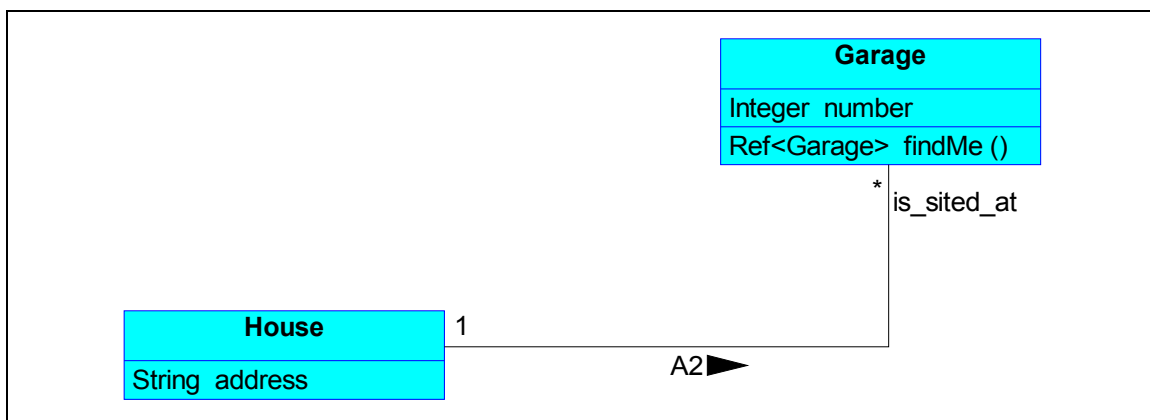
## 4. Template Changes and Extensions

### Data Schema Generation

If the spreadsheet data alternative is chosen, a Springboard template will be provided to generate a template CVS file from for the *static* classes and association model information. A .bat file will be provided to generate these domain data templates when the class modeling is complete for a given build. This template will be generated to <system\_name>\_<domain\_name>\_template.csv so any existing domain data will not be overwritten. When the template is filled in with actual data it should be renamed to <system\_name>\_<domain\_name>.csv.

### External Tracking Structure for "Many" Side of Association

This feature requires a change to structures generated for UML classes that are associated to "many" instances (either the "1" side of a "1:many", or both sides of a "many:many") so the association instance tracking structure is not contained within the class structure, but instead the class structure only carries a pointer to the association instance tracking structure.

**Figure 2: House->A2->Garage, 1:"many"**

In the model fragment above the House has many Garages across A2. In the current version of UML Foundation for C, the generated code below the implementation of the A2 instance tracking structure is within the House structure:

```

struct VH_House
{

```



```

. . .
    struct SW_InstanceTable acrossA2_to_is_sited_at;
. . .
};

```

**Figure 3: House – Old Generated Structure**

This tracking structure will be moved out of the OI\_Context structure, and a pointer will remain in OI\_Context:

```

struct VH_House
{
. . .
    struct VH_Garage** acrossA2_to_is_sited_at;
. . .
};

```

**Figure 4: House – New Generated Structure**

## Static Instance Module

If the spreadsheet data alternative is chosen, new templates are required to import and format the instance data from .csv files into C-language initializer statements. If the XML instance data alternative is chosen, XSL scripts will be developed to generate these statements.

A new module will be generated for each domain:

```
<domain prefix>_static_instances.c
```

to contain these initializers. Each *static* class will have an array

```
<domain_prefix>_<class name>_static_instances [<instance count>]
```

declared and initialized with that class's instance information from the spreadsheet.

Each *static* association will have an array

```
<domain_prefix>_across_A<association number>_<participant name>_<role
phrase>_<participant instance index> [<instance count>]
```

declared and initialized with the that association's link information FOR A SINGLE PARTICIPANT INSTANCE from the spreadsheet. In the example below note that the House is the only class that participates across from the "many" side of an association, so each instance of the House has it's own A2 link array.

### CarShuffle VH Instances

#### Class Structures

The following code snippets show the VH class structure declarations:

```

struct VH_Car
{
. . .
    short int rttiNumber ; /* Defined in <VH> types.h: VH OBJNUM Car */
. . .
};

```

```
short int state_;
SW_Incident_handle_t clientCallback;
int id;
vh_location_e location;
VH_Garage_handle_t acrossA3_to_is_currently_occupied_by;
};
```

**Figure 5: Class Structure**

```
struct VH_Garage
{
    short int rttiNumber_; /* Defined in <VH>_types.h: VH_OBJNUM_Garage */
    int number;
    VH_Car_handle_t acrossA3_to_selected;
    VH_House_handle_t acrossA2;
    VH_GarageDoor_handle_t acrossA1_to_closes;
};
```

**Figure 6: Garage Structure**

```
struct VH_GarageDoor
{
    short int rttiNumber_; /* Defined in <VH>_types.h: VH_OBJNUM_GarageDoor */
    short int state_;
    bool_t isLocked;
    bool_t isOpen;
    VH_Garage_handle_t acrossA1;
};
```

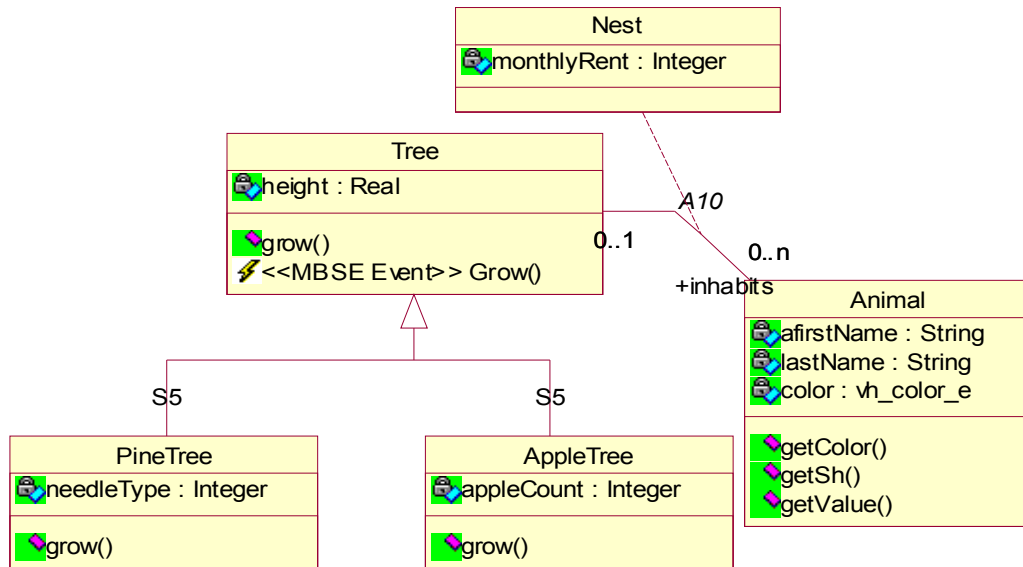
**Figure 7: GarageDoor Structure**

```
struct VH_House
{
    short int rttiNumber_; /* Defined in <VH>_types.h: VH_OBJNUM_House */
    struct SW_String address;
    struct VH_Garage **acrossA2_to_is_sited_at;
};
```

**Figure 8: House Structure**

## 5. Additional Example

The following example is provided to illuminate a more complicated example, including associate classes and supertype participants. It is set in the context of the CarShuffle\_test.VehicleHousing (VH) domain.



**Figure 9: Carshuffle\_test.VehicleHousing Class Model (partial)**

```

Object,CarShuffle_test.VH.Animal,StaticPopulation,TRUE
Object,CarShuffle_test.VH.Nest,StaticPopulation,TRUE
Object,CarShuffle_test.VH.AppleTree,StaticPopulation,TRUE
Object,CarShuffle_test.VH.PineTree,StaticPopulation,TRUE
BinaryRel,CarShuffle_test.VH.A10,StaticPopulation,TRUE

```

**Table 3: CarShuffle\_test properties.txt**

CarShuffle_test.VehicleHousing			
# between comment			
CLASS INFORMATION:			
AppleTree	appleCount	height	
1	30	20.5	
2	33	16	
3	55	71	

PineTree	height	needleType		
1	62	8		
2	15	2		
3	2	2		
# Varying color Animals				
Animal	color			
1	VH_COLOR_DARK_BROWN			
2	VH_COLOR_DARK_BROWN			
3	VH_COLOR_BROWN	# Wow - something different!		
4	VH_COLOR_DARK_BROWN			
5	VH_COLOR_BROWN			
Nest	monthlyRent			
1	100			
2	100			
3	125			
4	95			
5	101			
# between comment				
ASSOCIATION INFORMATION:				
# Not all Trees have inhabitants, and not all Animals have a tree				
A10	inhabits (VH_Animal)	(VH_Tree)	assoc (VH_Nest)	
	1	PineTree 1	1	# rel comment
	2	PineTree 2	2	
	3	AppleTree 1	3	
	4	PineTree 1	4	
	5	PineTree 1	5	

**Table 4: CarShuffle\_test\_VehicleHousing.csv**

**Static Instance Module**

The instance data for the VH domain will result in the following `VH_static_instances.c`:

```
#include "sw_conf.h" /* used for configuration information on some platforms */
#include "sys_incl.h" /* project-specific includes */

/*===== */
/* Code translated on : Tue Apr 01 10:27:21 2003 */

/* Static instance initialization for VehicleHousing domain
===== */
/* INCLUDES: */
#include "sw_base.h"
#include "sw_string.h"
#include "sw_instance_table.h"
#include "sw_incident.h"

#include "VH.h"
#include "VH_objs.h"

/*=====*/
/* DEFINITIONS AND FORWARD DECLARATIONS: */

/* Class instance counts */
#define VH_Animal_COUNT 5
#define VH_AppleTree_COUNT 3
#define VH_Nest_COUNT 5
```

```
#define VH_PineTree_COUNT 3

/* Static class instance storage */
extern struct VH_Animal VH_Animal_instanceData[];
extern struct VH_AppleTree VH_AppleTree_instanceData[];
extern struct VH_Nest VH_Nest_instanceData[];
extern struct VH_PineTree VH_PineTree_instanceData[];

/* Static association arrays (for "many" side only) */
extern struct SW_InstancePointerTable VH_acrossA10_to_inhabits_table_AppleTree_1;
extern struct SW_InstancePointerTable VH_acrossA10_to_inhabits_table_AppleTree_2;
extern struct SW_InstancePointerTable VH_acrossA10_to_inhabits_table_AppleTree_3;
extern struct SW_InstancePointerTable VH_acrossA10_to_inhabits_table_PineTree_1;
extern struct SW_InstancePointerTable VH_acrossA10_to_inhabits_table_PineTree_2;
extern struct SW_InstancePointerTable VH_acrossA10_to_inhabits_table_PineTree_3;
extern struct SW_InstancePointerTable VH_associated_acrossA10_to_inhabits_table_AppleTree_1;
extern struct SW_InstancePointerTable VH_associated_acrossA10_to_inhabits_table_AppleTree_2;
extern struct SW_InstancePointerTable VH_associated_acrossA10_to_inhabits_table_AppleTree_3;
extern struct SW_InstancePointerTable VH_associated_acrossA10_to_inhabits_table_PineTree_1;
extern struct SW_InstancePointerTable VH_associated_acrossA10_to_inhabits_table_PineTree_2;
extern struct SW_InstancePointerTable VH_associated_acrossA10_to_inhabits_table_PineTree_3;
/*===== */
/* CLASS INSTANCE DATA */

/*===== */
struct VH_Animal VH_Animal_instanceData[VH_Animal_COUNT] =
/* rttiNumber_, color, A10 -no role- (VH_Tree), associated class VH_Nest */
{
```

```

    VH_OBJNUM_Animal, VH_COLOR_DARK_BROWN, (VH_Tree_handle_t)&VH_PineTree_instanceData[1-1], &VH_Nest_instanceData[1-1],
    VH_OBJNUM_Animal, VH_COLOR_DARK_BROWN, (VH_Tree_handle_t)&VH_PineTree_instanceData[2-1], &VH_Nest_instanceData[2-1],
    VH_OBJNUM_Animal, VH_COLOR_BROWN, (VH_Tree_handle_t)&VH_AppleTree_instanceData[1-1], &VH_Nest_instanceData[3-1],
    VH_OBJNUM_Animal, VH_COLOR_DARK_BROWN, (VH_Tree_handle_t)&VH_PineTree_instanceData[1-1], &VH_Nest_instanceData[4-1],
    VH_OBJNUM_Animal, VH_COLOR_BROWN, (VH_Tree_handle_t)&VH_PineTree_instanceData[1-1], &VH_Nest_instanceData[5-1],
};

/* Support structures for manipulating the above array */
struct SW_InstanceTable VH_Animal_instanceList_storage_ =
{VH_Animal_COUNT, sizeof(struct VH_Animal), &VH_Animal_instanceData, -1, FALSE, 0};
SW_InstanceTable_handle_t VH_Animal_instanceList = &VH_Animal_instanceList_storage_;

/*===== */
struct VH_AppleTree VH_AppleTree_instanceData[VH_AppleTree_COUNT] =
/* rttiNumber_, height, A10 -no role- (array table), associated class VH_Nest, appleCount */
{
    VH_OBJNUM_AppleTree, 30, &VH_acrossA10_to_inhabits_table_AppleTree_1, &VH_associated_acrossA10_to_inhabits_table_AppleTree_1, 30,
    VH_OBJNUM_AppleTree, 33, &VH_acrossA10_to_inhabits_table_AppleTree_2, &VH_associated_acrossA10_to_inhabits_table_AppleTree_2, 33,
    VH_OBJNUM_AppleTree, 55, &VH_acrossA10_to_inhabits_table_AppleTree_3, &VH_associated_acrossA10_to_inhabits_table_AppleTree_3, 55,
};

/* Support structures for manipulating the above array */
struct SW_InstanceTable VH_AppleTree_instanceList_storage_ =
{VH_AppleTree_COUNT, sizeof(struct VH_AppleTree), &VH_AppleTree_instanceData, -1, FALSE, 0};
SW_InstanceTable_handle_t VH_AppleTree_instanceList = &VH_AppleTree_instanceList_storage_;

/*===== */
struct VH_Nest VH_Nest_instanceData[VH_Nest_COUNT] =
/* rttiNumber_, monthlyRent, A10 part1, A10 part2 */
{

```

```

    VH_OBJNUM_Nest, 100, (VH_Tree_handle_t)&VH_PineTree_instanceData[1-1], &VH_Animal_instanceData[1-1],
    VH_OBJNUM_Nest, 100, (VH_Tree_handle_t)&VH_PineTree_instanceData[2-1], &VH_Animal_instanceData[2-1],
    VH_OBJNUM_Nest, 125, (VH_Tree_handle_t)&VH_AppleTree_instanceData[1-1], &VH_Animal_instanceData[3-1],
    VH_OBJNUM_Nest, 95, (VH_Tree_handle_t)&VH_PineTree_instanceData[1-1], &VH_Animal_instanceData[4-1],
    VH_OBJNUM_Nest, 101, (VH_Tree_handle_t)&VH_PineTree_instanceData[1-1], &VH_Animal_instanceData[5-1],
};

/* Support structures for manipulating the above array */
struct SW_InstanceTable VH_Nest_instanceList_storage_ =
{VH_Nest_COUNT, sizeof(struct VH_Nest), &VH_Nest_instanceData, -1, FALSE, 0};
SW_InstanceTable_handle_t VH_Nest_instanceList = &VH_Nest_instanceList_storage_;

/*===== */
struct VH_PineTree VH_PineTree_instanceData[VH_PineTree_COUNT] =
/* rttiNumber_, height, A10 -no role- (array table), associated class VH_Nest, needleType */
{
    VH_OBJNUM_PineTree, 62, &VH_acrossA10_to_inhabits_table_PineTree_1, &VH_associated_acrossA10_to_inhabits_table_PineTree_1, 8,
    VH_OBJNUM_PineTree, 15, &VH_acrossA10_to_inhabits_table_PineTree_2, &VH_associated_acrossA10_to_inhabits_table_PineTree_2, 2,
    VH_OBJNUM_PineTree, 2, &VH_acrossA10_to_inhabits_table_PineTree_3, &VH_associated_acrossA10_to_inhabits_table_PineTree_3, 2,
};

/* Support structures for manipulating the above array */
struct SW_InstanceTable VH_PineTree_instanceList_storage_ =
{VH_PineTree_COUNT, sizeof(struct VH_PineTree), &VH_PineTree_instanceData, -1, FALSE, 0};
SW_InstanceTable_handle_t VH_PineTree_instanceList = &VH_PineTree_instanceList_storage_;

/*===== */
/* SUPERTYPE INSTANCE POINTER TABLES */
/* Pointers to all static subtype instances */

```



```

struct VH_Tree *VH_Tree_instancePointers[6]=
{
    (VH_Tree_handle_t)&VH_AppleTree_instanceData[0],
    (VH_Tree_handle_t)&VH_AppleTree_instanceData[1],
    (VH_Tree_handle_t)&VH_AppleTree_instanceData[2],
    (VH_Tree_handle_t)&VH_PineTree_instanceData[0],
    (VH_Tree_handle_t)&VH_PineTree_instanceData[1],
    (VH_Tree_handle_t)&VH_PineTree_instanceData[2],
};

struct SW_InstancePointerTable VH_Tree_instanceList_storage_ =
{6, (void*)&VH_Tree_instancePointers, -1, FALSE, 0};
SW_InstancePointerTable_handle_t VH_Tree_instanceList = &VH_Tree_instanceList_storage_;

/*===== */
/* ASSOCIATION LINK DATA ("many" side only) */

/* Not all Trees have inhabitants, and not all Animals have a tree */

/* A10 (Animal side) for AppleTree instance 1 */
struct VH_Animal *VH_acrossA10_to_inhabits_AppleTree_1[1] =
{
    &VH_Animal_instanceData[3-1]
};

struct SW_InstancePointerTable VH_acrossA10_to_inhabits_table_AppleTree_1 =
{1, (void*)&VH_acrossA10_to_inhabits_AppleTree_1, -1, FALSE, 0};

/* A10 (Animal side) for AppleTree instance 2 */
struct VH_Animal *VH_acrossA10_to_inhabits_AppleTree_2[1];

```

```
struct SW_InstancePointerTable VH_acrossA10_to_inhabits_table_AppleTree_2 =
{0, (void**)(&VH_acrossA10_to_inhabits_AppleTree_2), -1, FALSE, 0};

/* A10 (Animal side) for AppleTree instance 3 */
struct VH_Animal *VH_acrossA10_to_inhabits_AppleTree_3[1];
struct SW_InstancePointerTable VH_acrossA10_to_inhabits_table_AppleTree_3 =
{0, (void**)(&VH_acrossA10_to_inhabits_AppleTree_3), -1, FALSE, 0};

/* A10 (Animal side) for PineTree instance 1 */
struct VH_Animal *VH_acrossA10_to_inhabits_PineTree_1[3] =
{
    &VH_Animal_instanceData[1-1] /* rel comment */,
    &VH_Animal_instanceData[4-1],
    &VH_Animal_instanceData[5-1]
};
struct SW_InstancePointerTable VH_acrossA10_to_inhabits_table_PineTree_1 =
{3, (void**)(&VH_acrossA10_to_inhabits_PineTree_1), -1, FALSE, 0};

/* A10 (Animal side) for PineTree instance 2 */
struct VH_Animal *VH_acrossA10_to_inhabits_PineTree_2[1] =
{
    &VH_Animal_instanceData[2-1]
};
struct SW_InstancePointerTable VH_acrossA10_to_inhabits_table_PineTree_2 =
{1, (void**)(&VH_acrossA10_to_inhabits_PineTree_2), -1, FALSE, 0};

/* A10 (Animal side) for PineTree instance 3 */
struct VH_Animal *VH_acrossA10_to_inhabits_PineTree_3[1];
```

```
struct SW_InstancePointerTable VH_acrossA10_to_inhabits_table_PineTree_3 =
{0, (void**)(&VH_acrossA10_to_inhabits_PineTree_3), -1, FALSE, 0};

/* A10 associated class (Animal side) for AppleTree instance 1 */
struct VH_Nest *VH_associated_acrossA10_to_inhabits_AppleTree_1[1] =
{
    &VH_Nest_instanceData[3-1]
};

struct SW_InstancePointerTable VH_associated_acrossA10_to_inhabits_table_AppleTree_1 =
{1, (void**)(&VH_associated_acrossA10_to_inhabits_AppleTree_1), -1, FALSE, 0};

/* A10 associated class (Animal side) for AppleTree instance 2 */
struct VH_Nest *VH_associated_acrossA10_to_inhabits_AppleTree_2[1];
struct SW_InstancePointerTable VH_associated_acrossA10_to_inhabits_table_AppleTree_2 =
{0, (void**)(&VH_associated_acrossA10_to_inhabits_AppleTree_2), -1, FALSE, 0};

/* A10 associated class (Animal side) for AppleTree instance 3 */
struct VH_Nest *VH_associated_acrossA10_to_inhabits_AppleTree_3[1];
struct SW_InstancePointerTable VH_associated_acrossA10_to_inhabits_table_AppleTree_3 =
{0, (void**)(&VH_associated_acrossA10_to_inhabits_AppleTree_3), -1, FALSE, 0};

/* A10 associated class (Animal side) for PineTree instance 1 */
struct VH_Nest *VH_associated_acrossA10_to_inhabits_PineTree_1[3] =
{
    &VH_Nest_instanceData[1-1]      /* rel comment */,
    &VH_Nest_instanceData[4-1],
    &VH_Nest_instanceData[5-1]
};
```

```
struct SW_InstancePointerTable VH_associated_acrossA10_to_inhabits_table_PineTree_1 =
{3, (void**)(&VH_associated_acrossA10_to_inhabits_PineTree_1), -1, FALSE, 0};

/* A10 associated class (Animal side) for PineTree instance 2 */
struct VH_Nest *VH_associated_acrossA10_to_inhabits_PineTree_2[1] =
{
    &VH_Nest_instanceData[2-1]
};

struct SW_InstancePointerTable VH_associated_acrossA10_to_inhabits_table_PineTree_2 =
{1, (void**)(&VH_associated_acrossA10_to_inhabits_PineTree_2), -1, FALSE, 0};

/* A10 associated class (Animal side) for PineTree instance 3 */
struct VH_Nest *VH_associated_acrossA10_to_inhabits_PineTree_3[1];
struct SW_InstancePointerTable VH_associated_acrossA10_to_inhabits_table_PineTree_3 =
{0, (void**)(&VH_associated_acrossA10_to_inhabits_PineTree_3), -1, FALSE, 0};
```