



---

## **PathMATE Dynamic Tasks**

Version 1.0  
December 13, 2007

---

## *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

|  |          |
|--|----------|
| <b>1. Introduction.....</b>                      | <b>1</b> |
| <b>2. Dynamic Tasks – Overview .....</b>         | <b>1</b> |
| Task Lifecycles.....                             | 1        |
| <i>Service-Based Dynamic Task .....</i>          | <i>2</i> |
| <i>Instance-Based Dynamic Task .....</i>         | <i>2</i> |
| Target Task Resolution with Dynamic Tasking..... | 2        |
| <i>Service-Based Resolution .....</i>            | <i>2</i> |
| <i>Instance-Based Resolution .....</i>           | <i>2</i> |
| Dynamic Tasks and Bundles .....                  | 3        |
| <i>Marking Bundles .....</i>                     | <i>3</i> |
| Changing Tasks.....                              | 4        |
| <b>3. Marking Summary.....</b>                   | <b>5</b> |
| <b>4. Dynamic Task Management .....</b>          | <b>6</b> |
| Dynamic Task Mechanisms .....                    | 6        |
| Memory Pools .....                               | 7        |

# 1. Introduction

This Technical Note describes Dynamic extensions to the PathMATE distributed deployment and mutex capabilities in support of *dynamic tasks*. Static distributed deployment capabilities support the allocation of domains to a fixed set of tasks at translation time. For computationally intensive actions, it may be necessary to dynamically allocate the processing of these actions to a separate task to make the most effective use of multiprocessor platforms.

Dynamic Tasking provides capabilities for the flexible addition and deletion of tasks or allocation of processing to different tasks at runtime, including:

- Start processing within a newly activated task within an existing process
- Allocate bundles of class instances, as a unit identified through markings, to tasks.
- Route incidents to dynamic class instances.
- Manage pools of dynamic tasks via a specific realized-code-level interface to software dynamic task mechanisms.

The capabilities identified in this technical note are intended to support the execution of class instance behavior (class operations and state actions) to multiple tasks for a single domain. This may involve the use of mutexes and therefore all the cautions required in their application apply.

## 2. Dynamic Tasks – Overview

### Task Lifecycles

A static task is started upon system initialization, and is shut down with system shutdown. Its priority and task local memory pool can be specified uniquely. The initial processing allocated to this task is allocated at transformation time. Alternatively, dynamic tasks start when needed, and stop when the allocated processing is "done". (Processing may be idle, but the task is still active and allocated.) Dynamic task processing is allocated in two distinct contexts: non-instance service based and class instance-based. The designer will specify a maximum number of dynamic tasks that will be created, using the MaxDynamicTasks system marking. All of these tasks will run at the same priority and use separately allocated local memory pools which will all have the same configuration. When using dynamic markings, static tasks may actually have processing dynamically allocated to them, but the task never becomes exclusively available for allocation as a DYNAMIC task. In other words, a static task can take on additional processing for a class or operation, but it is not dedicated to that additional processing alone as a DYNAMIC task would be.

*Task Pools Note:* Although the terms "start" and "stop" are used throughout this section, they may resolve to the allocation of an existing, idle task from a pool and returning that task to that pool as opposed to asking an RTOS service to actually create or destroy a task.

### ***Service-Based Dynamic Task***

When a domain service or non-instance based ("class-based") class service is marked with a TaskID of "DYNAMIC", a new task is started upon invocation of the service. A service handle is passed to the new task. The service runs to completion within the new task, and the task ends when the service returns. It is the designer's responsibility to ensure that any class instances created within the processing of this service are allocated to another task.

### ***Instance-Based Dynamic Task***

When a class is marked with a TaskID of "DYNAMIC", this indicates a new task is started upon creation of an instance, and this instance is allocated to this new task. All instance-based operations and event processing is done in this task. The task completes when the instance is deleted or when the instance is allocated to another task.

## **Target Task Resolution with Dynamic Tasking**

For dynamic tasking, non-local dispatches of domain or class service invocations and event generations to dynamic class instances use a run-time resolution of the target task. A dynamic task is one that does not have processing allocated to it at start up time. A dynamic task may be initialized at startup, but left idle until a dynamic service or class is allocated to it. A dynamic class is a class that has a dynamic marking on it or is part of a bundle which has a dynamic marking on it. In other words, a dynamic class is one which may change the task to which it is allocated at least once during its lifetime. A dynamic class does not need to be allocated to a dynamic task in order to be considered dynamic. A dynamic class can be created in either a static or dynamic task and it can transition to another task either static or dynamic zero or more times in its lifetime.

### ***Service-Based Resolution***

Run-time resolution for "Dynamic" non-instance-based services starts a new task, and dispatches to this new task. Only services which do not return a value may be marked as dynamic.

### ***Instance-Based Resolution***

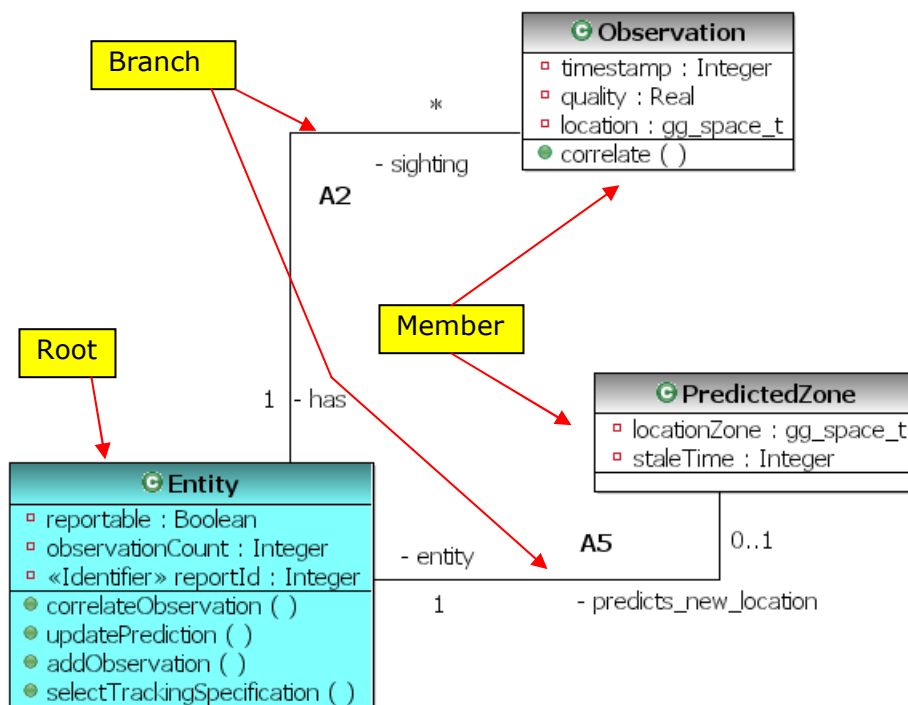
An instance of a dynamic class carries its current task id in a data member called myTask of type PfdTask. Run-time resolution of event generations and instance-based operation invocations to dynamic class instances invokes a service PfdTask getTask() which refers to this data member. If this instance is the root of a Bundle (see below) then all member instances are also allocated to this task. The service getTask() for members of the bundle traverses the BundleBranch to call the participant's getTask() service iteratively until it reaches the root class, where getTask() returns the root class instance's value of myTask. Only instance based operations which do not return a value should be invoked on a class instance which will be in different task.

## Dynamic Tasks and Bundles

A set of instances working together in a task can be marked with the Bundle pattern. The core design assumption is that all class instances within a single dynamic bundle "instance" have nested lifecycles with related processing. When the root instance in a dynamic bundle is deleted, all member class instances of the dynamic bundle are also deleted. When the root instance in a dynamic bundle changes its task, all the member class instances of the dynamic bundle also change the task in which they run.

### Marking Bundles

A set of classes can be allocated to a single task (static or dynamic) as a unit when the set conforms to the Bundle pattern. This pattern identifies a tree of information starting from a root class instance, and branching across associations to member class instances. The BundleMember marking propagates down a class hierarchy. A class which is part of a class hierarchy may only be marked as a BundleMember and not a BundleRoot.



**Figure 1: Example Bundle**

The bundle pattern requires that:

- A single root class be identified for a bundle.
- All member instances are created and deleted (directly or indirectly) by the root instance. Member instances must have lifetimes bounded by the lifetime of the root object.

- All branch associations be identified, and have a multiplicity of 1 on the side toward the root. (However, having a multiplicity of 1 does not require that it be part of the bundle.)
- A single branch association link exists between any 2 different member instances. Any other associations between member classes cannot be a branch. Bundle patterns are tree subgraphs of the class diagram.
- A class can only participate in one dynamic bundle, but may participate in multiple bundles that are not dynamic. A dynamic bundle is one in which the BundleRoot class has dynamic markings.

Bundles have names, used in PIM marking which connect the pieces of the bundle together. The new markings for the bundle pattern are:

| <b>element</b> | <b>name</b>  | <b>default value</b> | <b>notes</b>  |
|----------------|--------------|----------------------|---|
| Class          | BundleRoot   | <blank>              | Non-blank indicates this is the Root class for the named bundle. When an instance of this class is created, an instance of the bundle is also considered to exist. <i>Multiple values may be specified using a semicolon as a separator.</i>    |
| Class          | BundleMember | <blank>              | Non-blank indicates this is a member class for the named bundle. It is presumed that the lifecycle of this class nests within the root class lifecycle. <i>Multiple values may be specified using a semicolon as a separator.</i>               |
| Participant    | BundleBranch | <blank>              | Non-blank indicates this is a branch of the named bundle. The marking is applied to the root side of the association. The multiplicity of this participant must be 1. <i>Multiple values may be specified using a semicolon as a separator.</i> |

**Table 1: Bundle Markings**

In the example in Figure 1: Example Bundle, assuming the name of the Bundle is EntityBundle, the markings are:

```
Object,Sentry.EntityTracking.Entity,BundleRoot,EntityBundle
Object,Sentry.EntityTracking.Observation,BundleMember,EntityBundle
Object,Sentry.EntityTracking.PredictedZone,BundleMember,EntityBundle
Participant,Sentry.EntityTracking.A2.Entity.has,BundleBranch,EntityBundle
Participant,Sentry.EntityTracking.A5.Entity.entity,BundleBranch,EntityBundle
```

**Table 2: Example Bundle Markings**

## Changing Tasks

A dynamic class, or dynamic bundle, may be switched from a static task to a dynamic task or back. To change the task assignment of a dynamic class instance and any associated member instances (if the class is the root of a bundle) before running a class operation, mark the class operation with ChangeTaskID. To change the task assignment at the completion of an entry state action, mark the state action with ChangeTaskIDWhenDone. If a class has any operations or state actions with a non-blank ChangeTaskID or

ChangeTaskIdWhenDone, the class is considered Dynamic. If an instance is changing *from* a Dynamic task, that Dynamic task is stopped. If the class participates in a bundle it may only use the dynamic task markings if it is the root class.

### 3. Marking Summary

| <b>element</b>   | <b>name</b>          | <b>default value</b>                                  | <b>notes</b>  |
|--|----------------------|---|---|
| Class  | BundleRoot           | <blank>   | A non-blank value indicates this is the root class for a bundle of the specified name. Multiple values may be specified using a semicolon as a separator.   |
| Class  | BundleMember         | <blank>   | A non-blank value indicates this is a member class for a bundle of the specified name. It is presumed that the lifecycle of this class nests within the root class lifecycle. This class' task assignment is inherited from the bundle's root. Multiple values may be specified using a semicolon as a separator. |
| Class  | TaskID               | <if Member, Root's TaskID, otherwise domain's TaskID> | Identifies the task to which this class is allocated.. SYS_TASK_ANY will allow this class to run locally in any calling task. DYNAMIC indicates each new instance is allocated to a new task.   |
| Class Operation (instance-based only)(currently unsupported) | ChangeTaskID         | <blank>   | A non-blank value indicates this instance is to be changed from its current task to a new task. This marking is not valid for classes with BundleMember marking..   |
| Domain   | TaskID               | SYS_TASK_ID_MAIN                                      | Identifies the task to which this domain is allocated. SYS_TASK_ANY will allow this domain to run locally in any calling task   |
| Domain Service   | TaskID               | <domain's TaskID>                                     | Identifies the task to which this domain service is allocated. SYS_TASK_ANY will allow this domain to run locally in any calling task. DYNAMIC indicates this is allocated to a new task.   |
| Parameter  | Stereotype           | <blank>   | A value of "Routing" indicates this is a Routing parameter, used in explicit run-time resolution.   |
| Participant  | BundleBranch         | <blank>   | A non-blank value indicates this is a branch of a bundle of the specified name. The marking is applied to the root side of the association. The multiplicity of this participant must be 1. Multiple values may be specified using a semicolon as a separator.  |
| State  | ChangeTaskIDWhenDone | <blank>   | A non-blank value indicates this instance is to be changed from its current task to a new task upon completion of the state. This marking is not valid for classes with   |

|        |                                |           |  |
|--------|--------------------------------|-----------|--|
|        |                                |           | <i>BundleMember marking.</i>   |
| System | MaxDynamicTasks                | 0         | <i>Maximum number of dynamic tasks in a process.</i>   |
| System | ProcessAddress/<process label> | 127.0.0.1 | <i>Identifies the IP address of the processor on which the process (indicated by the process label in the property name) runs.</i> |

Table 3: Marking Summary

## 4. Dynamic Task Management

By default, dynamic tasks are created via an RTOS as their processing is started, and when their processing is completed they are destroyed. Their priority is set by the dynamic element's task priority. (clarify this? Does this mean that a dynamic tasks priority will change depending on the element that is allocated to it?)

Alternatively a fixed size pool of tasks can be created at system startup by a call to SW:InitializeDynamicTasks(count, default\_priority). (Is this needed since we have MaxDynamicTasks) A set of mechanisms are provided to facilitate the management of this pool. In this case, dynamic tasks are allocated from available (idle) tasks in the pool, and returned to the pool when their processing is complete. The tasks all run the same event loop, PfdTask::processOOA().

In either case, RTOS or pools of tasks, the maximum number of dynamic tasks will be limited by the system property MaxDynamicTasks. The default configuration will be to have a pool of dynamic tasks. In order to use tasks which are transient set the compile switch PATH\_USE\_TRANSIENT\_DYNAMIC\_TASKS.

### Dynamic Task Mechanisms

A DynamicTaskPool design mechanism is provided to manage all dynamic task requests. A new system property MaxDynamicTasks limits (at transformation time) the number of both RTOS and pooled tasks which can be created at run time in a process.

The getNewDynamicTask/releaseDynamicTask methods provide a single interface for both RTOS tasks and the pooled tasks. If the task cannot be created it will be treated similar to an out of memory exception, in that an exception will be raised. However, If a new dynamic task cannot be allocated the currently executing task will be used, rather than setting the task context to NULL which would cause a NULL dereference.

PfdTask \***getNewDynamicTask**(pfdos\_priority\_e target\_priority)

No pool: start an RTOS task of the specified priority

Pool: select an idle task from the pool with the same or a lower priority as the target\_priority and mark it as allocated

**releaseDynamicTask**(pfdos\_task\_id\_t task)

No pool: deallocate the RTOS task

Pool: mark the task as not allocated.



**Additional methods support pool-specific operations: (currently unsupported)**

**initializeDynamicTaskPool**(int task\_count, pfdos\_priority\_e default\_priority) - Start task\_count PfdTasks at default\_priority. A priority of SYS\_TASK\_PRIORITY\_LOWEST indicates all tasks can be used for any priority. (Called from SW:InitializeDynamicTasks)

**setTaskPriority**(pfdos\_task\_id\_t task, pfdos\_priority\_e priority) - Set the priority of the specified task, both within the DynamicTaskPool internal table, and in the RTOS

## Memory Pools

Dynamic tasks will work with task local pools, even though classes may be created in one task and deleted in another. See the Memory Pools technote for more information on how task local memory pools are supported.