



Using the Basic Type Fine-Grained Time

Version 1.1
September 16, 2007

PathMATE Technical Notes

Pathfinder Solutions LLC
33 Commercial Drive, Suite 2
Foxboro, MA 02035 USA
www.PathfinderMDA.com
888-662-7284

Table Of Contents

1. Introduction..... 1

1. Introduction

This Tech Note describes the basic type called `FineGrainedTime`. This type stores time with nanosecond granularity. It is currently available for the C++ maps. It can be used as an attribute value type, as a service parameter type, or as a local variable.

2. Modeling

The basic type `FineGrainedTime` can be used as an attribute on a class, as a parameter to a service, or as a local variable in action language. If uninitialized, its initial value will be zero nanoseconds. In action language you would declare a variable "duration" with type `FineGrainedTime` in the following way:

```
FineGrainedTime duration; //initial value is 0
```

Assignment

Assignment between Integers, Reals, and `FineGrainedTimes`, and `GenericValues` which hold an Integer, Real and `FineGrainedTime` is freely permitted. Assigning to an `FineGrainedTime` type assumes the values are in *milliseconds*.

```
FineGrainedTime fgt = 120.45; // 120 ms, 450000 ns
FineGrainedTime fgt2 = -1500; // -1.5 seconds
FineGrainedTime fgt3 = 3000000000; // 3,000,000 seconds:
// if this results in a compile error (because the
// constant is a long), use a double constant as follows:
FineGrainedTime fgt4 = 3000000000.0; // 3,000,000 seconds
```

`FineGrainedTime` may also be converted to and from a string. The value is represented in milliseconds with 6 decimal places representing nanoseconds.

```
String timeAsFgt = fgt; // timeAsFgt="0.000120" 120 ns
FineGrainedTime fgt2 = "5.000120" // 5 ms, 120 ns
```

Use the `SoftwareMechanisms` service `SW:GetFineGrainedTime` to get the current time in a `FineGrainedTime` variable.

```
FineGrainedTime now = SW:GetFineGrainedTime();
```

Arithmetic

Expressions with arithmetic operators `+`, `-`, `*`, and `/` are supported with the `FineGrainedTime` data type. The modulo operator, `%`, is also supported for `FineGrainedTime` data types. The resulting expression type is `FineGrainedTime`.

```
FineGrainedTime fgt = 120.45; // 120 ms, 450000 ns
```

```
FineGrainedTime fgt2 = fgt / 2; // fgt2 = 60 ms, 225000 ns
FineGrainedTime fgt3 = fgt * 2; // fgt3 = 240 ms, 900000 ns
```

Unary Operator	Type	Result	Description
+	FGT	FGT	Returns the input value
-	FGT	FGT	Returns the negative of the input value

Left-Hand Type	Binary Operator	Right-Hand Type	Result	Description
FGT	+	FGT	FGT	Sum of the two times.
FGT	+	Integer	FGT	Sum of left-hand side and the right-hand side converted to Fine-Grained Time.
Integer	+	FGT	FGT	Sum of left-hand side converted to Fine-Grained Time and the right-hand side.
FGT	+	Real	FGT	Sum of left-hand side and the right-hand side converted to Fine-Grained Time.
Real	+	FGT	FGT	Sum of left-hand side converted to Fine-Grained Time and the right-hand side.
FGT	+	String	String	Converts the left-hand side to a String in the form "m.n" and concatenates to the right-hand string.
String	+	FGT	String	Converts the right-hand side to a String in the form "m.n" and concatenates to the left-hand string.
FGT	-	FGT	FGT	Difference of the two times.
FGT	-	Integer	FGT	Difference of left-hand side and the right-hand side converted to Fine-Grained Time.
Integer	-	FGT	FGT	Difference of left-hand side converted to Fine-Grained Time and the right-hand side.
FGT	-	Real	FGT	Difference of left-hand side and the right-hand side converted to Fine-Grained Time.
Real	-	FGT	FGT	Difference of left-hand side converted to Fine-Grained Time and the right-hand side.

Left-Hand Type	Binary Operator	Right-Hand Type	Result	Description
FGT	*	FGT	Real	Product of two Fine-Grained Time values, interpreted as Reals in milliseconds. CRRS uses time-squared for acceleration calculations, so we need this operator. The result can be a Real, however. It does not make sense as a Fine-Grained Time because the units are ms ² .
FGT	*	Integer	FGT	Multiplies the seconds, milliseconds, and nanoseconds of the left-hand side by the right-hand side.
Integer	*	FGT	FGT	Multiplies the seconds, milliseconds, and nanoseconds of the right-hand side by the left-hand side.
FGT	*	Real	FGT	Multiplies the seconds, milliseconds, and nanoseconds of the left-hand side by the right-hand side.
Real	*	FGT	FGT	Multiplies the seconds, milliseconds, and nanoseconds of the right-hand side by the left-hand side.
FGT	/	FGT	Real	Ratio of two Fine-Grained Time values, interpreted as Reals in milliseconds.
FGT	/	Integer	FGT	Divides the seconds, milliseconds, and nanoseconds of the left-hand side by the right-hand side.
FGT	/	Real	FGT	Divides the seconds, milliseconds, and nanoseconds of the left-hand side by the right-hand side.
Integer	/	FGT	Real	Divides the left-hand integer by the right-hand side, treating it as a Real value in milliseconds. It does not make sense as a Fine-Grained Time because the units are 1/ms.

Left-Hand Type	Binary Operator	Right-Hand Type	Result	Description
Real	/	FGT	Real	Divides the left-hand integer by the right-hand side, treating it as a Real value in milliseconds. It does not make sense as a Fine-Grained Time because the units are 1/ms.
FGT	%	FGT	FGT	The remainder from dividing the left-hand side by the right-hand side.
FGT	%	Integer	FGT	The remainder from dividing the left-hand side by the right-hand side.

Comparison Operators

Comparisons operators, `>`, `>=`, `<`, `<=`, `==`, and `!=` between Integers, Reals, GenericValues, and FineGrainedTimes are performed in units of milliseconds on a FineGrainedTime type. An Integer or a Real is converted to a FineGrainedTime before the logical operator is performed.

```
FineGrainedTime fgt = 120.45; // 120 ms, 450000 ns
(fgt == 120.45) // yields TRUE
```

Left-Hand Type	Binary Operator	Right-Hand Type	Result	Description
FGT	==	FGT	Boolean	TRUE if the values are the same
FGT	==	Integer	Boolean	TRUE if the right-hand side converted to Fine-Grained Time equals the left-hand side.
Integer	==	FGT	Boolean	TRUE if the left-hand side converted to Fine-Grained Time equals the right-hand side.
FGT	==	Real	Boolean	TRUE if the right-hand side converted to Fine-Grained Time equals the left-hand side.
Real	==	FGT	Boolean	TRUE if the left-hand side converted to Fine-Grained Time equals the right-hand side.
FGT	!=	FGT	Boolean	FALSE if the values are the same
FGT	!=	Integer	Boolean	FALSE if the right-hand side converted to Fine-Grained Time equals the left-hand side.

Left-Hand Type	Binary Operator	Right-Hand Type	Result	Description
Integer	!=	FGT	Boolean	FALSE if the left-hand side converted to Fine-Grained Time equals the right-hand side.
FGT	!=	Real	Boolean	FALSE if the right-hand side converted to Fine-Grained Time equals the left-hand side.
Real	!=	FGT	Boolean	FALSE if the left-hand side converted to Fine-Grained Time equals the right-hand side.
FGT	>	FGT	Boolean	TRUE if the left-hand side is greater than the the right-hand side.
FGT	>	Integer	Boolean	TRUE if the left-hand side is greater than the the right-hand side converted to Fine-Grained Time.
Integer	>	FGT	Boolean	TRUE if the left-hand side converted to Fine-Grained Time is greater than the the right-hand side.
FGT	>	Real	Boolean	TRUE if the left-hand side is greater than the the right-hand side converted to Fine-Grained Time.
Real	>	FGT	Boolean	TRUE if the left-hand side converted to Fine-Grained Time is greater than the the right-hand side.
FGT	>=	FGT	Boolean	TRUE if the left-hand side is greater than or equal to the the right-hand side.
FGT	>=	Integer	Boolean	TRUE if the left-hand side is greater than or equal to the the right-hand side converted to Fine-Grained Time.
Integer	>=	FGT	Boolean	TRUE if the left-hand side converted to Fine-Grained Time is greater than or equal to the the right-hand side.
FGT	>=	Real	Boolean	TRUE if the left-hand side is greater than or equal to the the right-hand side converted to Fine-Grained Time.
Real	>=	FGT	Boolean	TRUE if the left-hand side converted to Fine-Grained Time is greater than or equal to the the right-hand side.

Left-Hand Type	Binary Operator	Right-Hand Type	Result	Description
FGT	<	FGT	Boolean	TRUE if the left-hand side is less than the the right-hand side.
FGT	<	Integer	Boolean	TRUE if the left-hand side is less than the the right-hand side converted to Fine-Grained Time.
Integer	<	FGT	Boolean	TRUE if the left-hand side converted to Fine-Grained Time is less than the the right-hand side.
FGT	<	Real	Boolean	TRUE if the left-hand side is less than the the right-hand side converted to Fine-Grained Time.
Real	<	FGT	Boolean	TRUE if the left-hand side converted to Fine-Grained Time is less than the the right-hand side.
FGT	<=	FGT	Boolean	TRUE if the left-hand side is less than or equal to the the right-hand side.
FGT	<=	Integer	Boolean	TRUE if the left-hand side is less than or equal to the the right-hand side converted to Fine-Grained Time.
Integer	<=	FGT	Boolean	TRUE if the left-hand side converted to Fine-Grained Time is less than or equal to the the right-hand side.
FGT	<=	Real	Boolean	TRUE if the left-hand side is less than or equal to the the right-hand side converted to Fine-Grained Time.
Real	<=	FGT	Boolean	TRUE if the left-hand side converted to Fine-Grained Time is less than or equal to the the right-hand side.

Mathematical Expressions

The following mathematical expressions will be added to the Math domain:

Name	Parameters	Result	Description
Math:tmin	FGT, FGT	FGT	The minimum of the two input values.

Name	Parameters	Result	Description
Math:tmin	Integer, FGT	FGT	The minimum of the first parameter converted to Fine-Grained Time and the second parameter.
Math:tmin	FGT, Integer	FGT	The minimum of the first parameter and the second parameter converted to Fine-Grained Time.
Math:tmin	Real, FGT	FGT	The minimum of the first parameter converted to Fine-Grained Time and the second parameter.
Math:tmin	FGT, Real	FGT	The minimum of the first parameter and the second parameter converted to Fine-Grained Time.
Math:tmax	FGT, FGT	FGT	The maximum of the two input values.
Math:tmax	Integer, FGT	FGT	The maximum of the first parameter converted to Fine-Grained Time and the second parameter.
Math:tmax	FGT, Integer	FGT	The maximum of the first parameter and the second parameter converted to Fine-Grained Time.
Math:tmax	Real, FGT	FGT	The maximum of the first parameter converted to Fine-Grained Time and the second parameter.
Math:tmax	FGT, Real	FGT	The maximum of the first parameter and the second parameter converted to Fine-Grained Time.
Math:tabs	FGT	FGT	The absolute value of the input value.

Delayed Events

FineGrainedTime and expressions returning fine grained time may be used in a GENERATE AFTER statement.

```
FineGrainedTime fgt = 120; // 120 ms
GENERATE <event> AFTER (fgt * 2) TO (instance); // after 240ms
```

Nanosecond timers are not supported on all platforms. When a timeout contains an interval less than 1 millisecond, the delayed event will be delivered, but on platforms with millisecond timer resolution, the event will arrive late.

Exception Handling

Exceptions for overflow, underflow and divide by zero are generated and can be caught and handled using SW:RegisterForErrorGroup() with the group SW_ERROR_GROUP_MECHANISMS. See error_codes.hpp in the cpp/mechanisms directory for the error codes which can be generated for this group of errors.

3. Marking

There are no special markings required to use the FineGrainedTime basic type.

4. Mechanisms/Realized Code

The basic type FineGrainedTime is implemented in the PathMATE mechanisms by the class PfdFineGrainedTime which is defined in pfd_timer.hpp. This class previously stored time in units of milliseconds, but has been modified to hold units of nanoseconds.

The PfdDataContainer class, which is defined in datacont.hpp, has been modified to be able to hold a FineGrainedTime value. This support was implemented using a struct with fields for seconds and nanoseconds, rather than holding the PfdFineGrainedTime class directly in order to avoid increasing the memory consumed by the PfdDataContainer class.

5. Building/Deployment

The FineGrainedTime basic type does not behave differently in different deployments. The class does have different implementations for integer multiplication and division when the compile switch PAT