2042 TOD Remediation Strategies

Application Development Subcommittee

Tim Backus

2024 TPF Users Group Conference
May 5-8, 2024, New Orleans, Louisiana

IBM Z

IBM

Problem Statement

The 8-byte time-of-day (TOD) clock will overflow on September 17, 2042. This will cause STCK and STCKF instructions to result in values that are not correct.

The 16-byte extended TOD clock provides a much larger range of timestamps and will not overflow for thousands of years.

Migration to a 16-byte clock value could require extensive changes to your traditional z/TPF database.

Users

Anna
application architect

As part of a department-wide effort, Anna is tasked with investigating the changes that should be made to prepare for the TOD clock overflow that will occur in 2042.

The applications that she is responsible for are mission critical and use several common data structures that are used extensively throughout her company’s database.

As-is Scenario

Anna
application architect

Anna's company has two choices for every TOD clock timestamp in their database:

1. Leave the field as-is.
2. Update the field to use the 16-byte extended TOD clock format.

As-is Scenario

Anna
application architect

1. **Leave the field as-is.**
   - Any application that uses the field must function properly after the overflow occurs.
     - *Example: Clock values that are used only as unique, temporary IDs.*
2. Update the field to use the 16-byte extended TOD clock format.

As-is Scenario

Anna
application architect

1. Leave the field as-is.
2. **Update the field to use the 16-byte extended TOD clock format.**
   - For clock values that represent a specific point in time.
     - *Examples: Flight time history, dates of birth.*
   - At least 8 reserve bytes are required for an easy migration.

Pain Points

If 8-byte TOD clock values exist between other values, migration to the 16-byte extended TOD clock format could require an extensive amount of effort reorganizing the database.

Example:

```
...
BookHistoryHdr hdr;
unsigned long uid;
tpf_TOD_type bookTime;
unsigned short lastFourSSN;
char dobMonth;
...
```

Pain Points

Anna
application architect

She looks through the relevant data structures and notices that several of the 8-byte TOD clock values are immediately surrounded by other values with no padding bytes to repurpose.

```
...
BookHistoryHdr hdr;
unsigned long uid;
tpf_TOD_type bookTime;
unsigned short lastFourSSN;
char dobMonth;
...
```

To-be Scenario

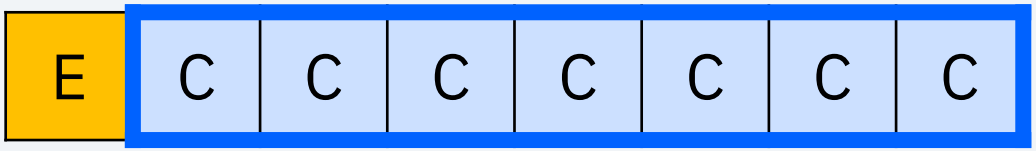
Anna
application architect

Anna realizes that there are two additional 8-byte TOD clock formats that she can use for each timestamp she is responsible for.

Her company has four strategies per timestamp:

1. Leave the field as-is.
2. Update the data structure to use the 16-byte extended TOD clock format.
3. **Use the 8-byte epoch TOD clock format.**
4. **Use the 8-byte bracketed TOD clock format.**

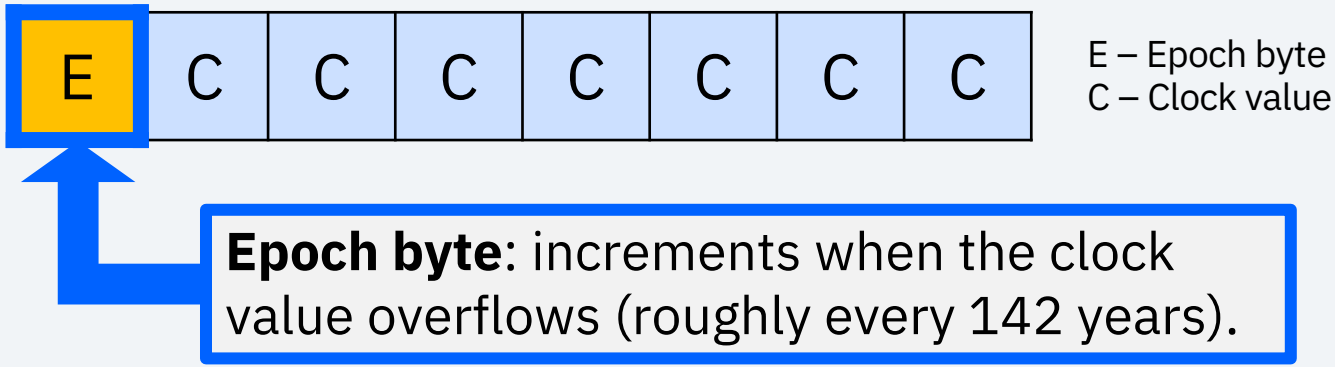
Epoch TOD Clock – Format



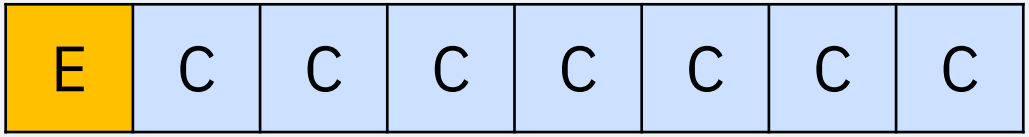
E – Epoch byte
C – Clock value

Clock value: the most significant (leftmost) 7 bytes of the standard TOD clock.

Epoch TOD Clock – Format



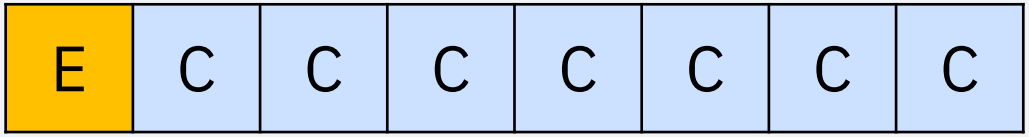
Epoch TOD Clock – Technical Details



E – Epoch byte
C – Clock value

- Its data type is **tpf_TODExtHi_type**.
- This format is equivalent to the first 8 bytes of the 16-byte extended TOD clock.
- The range of timestamps it can represent is identical to the extended TOD clock's: the year 1900 to tens of thousands of years into the future.

Epoch TOD Clock – Technical Details



E – Epoch byte
C – Clock value

- Its main drawback is that it only has a 62.5 nanosecond granularity.
  - The values between two instances of this format could be identical.
- Additional investigation might be required to determine if this format's granularity is an issue for timestamps in your database.
- Data migration may also be required if a field is updated from a standard TOD clock value. This is because the internal data of the standard TOD clock isn't equivalent to the epoch TOD clock's data.

Standard 8-byte TOD Clock

- TOD clock values are grouped by the value of their most significant bit.
- We call each contiguous range of clock values a “bracket”.

Spans approx. 71 years

Bit 0 is off

Timestamp (approx.)	TOD Clock Value
1900-01-01	X’**0**000 0000 0000 0000
1935-09-06	X’**4**000 0000 0000 0000
1971-05-11	X’**7**FFF FFFF FFFF FFFF
1971-05-11	X’8000 0000 0000 0000
2007-01-13	X’C000 0000 0000 0000
2042-09-17	X’FFFF FFFF FFFF FFFF

The leftmost bit is ***off*** for the first bracket.

Standard 8-byte TOD Clock

- TOD clock values are grouped by the value of their most significant bit.
- We call each contiguous range of clock values a “bracket”.

Timestamp (approx.)	TOD Clock Value
1900-01-01	X’0000 0000 0000 0000
1935-09-06	X’4000 0000 0000 0000
1971-05-11	X’7FFF FFFF FFFF FFFF
1971-05-11	X’**8**000 0000 0000 0000
2007-01-13	X’**C**000 0000 0000 0000
2042-09-17	X’**F**FFF FFFF FFFF FFFF

Spans approx. 71 years

Bit 0 is on

The leftmost bit is ***on*** for the second bracket.

Standard 8-byte TOD Clock

	Timestamp (approx.)	TOD Clock Value
Spans approx. 71 years Bit 0 is off	1900-01-01	X'0000 0000 0000 0000
	1935-09-06	X'4000 0000 0000 0000
	1971-05-11	X'7FFF FFFF FFFF FFFF
Spans approx. 71 years Bit 0 is on	1971-05-11	X'8000 0000 0000 0000
	2007-01-13	X'C000 0000 0000 0000
	2042-09-17	X'FFFF FFFF FFFF FFFF

The standard 8-byte TOD Clock represents timestamps from 1900 to 2042, so it spans these two brackets.

Bracketed TOD Clock – Introduction

Timestamp (approx.)	TOD Clock Value
1900-01-01	X'0000 0000 0000 0000
1935-09-06	X'4000 0000 0000 0000
1971-05-11	X'7FFF FFFF FFFF FFFF
1971-05-11	
2007-01-13	
2042-09-17	X'FFFF FFFF FFFF FFFF
2042-09-17	???
2078-05-23	???
2114-01-26	???

What if we wanted to represent the years 2042 to 2114 with a third bracket?

Spans approx. 71 years

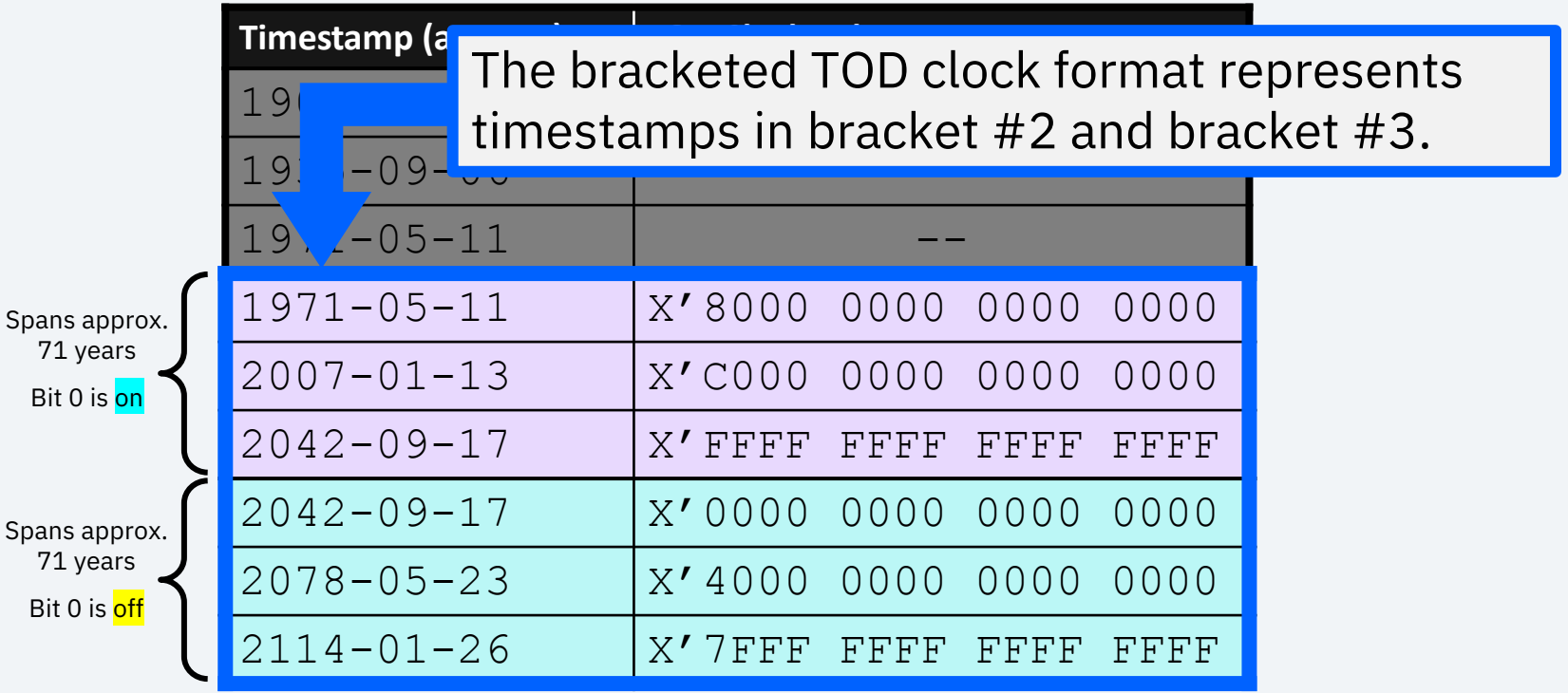
Bit 0 is off

Bracketed TOD Clock – Introduction

Timestamp (approx.)	TOD Clock Value
1900-01-01	X'0000 0000 0000 0000
1935-09-06	X'4000 0000 0000 0000
1971-05-11	X'7FFF FFFF FFFF FFFF
1971-05-11	X'8000 0000 0000 0000
	00 00 0
	F FFFF
2042-09-17	???
2078-05-23	???
2114-01-26	???

We can repurpose the first bracket's values and use them for the third bracket.

Bracketed TOD Clock – Introduction



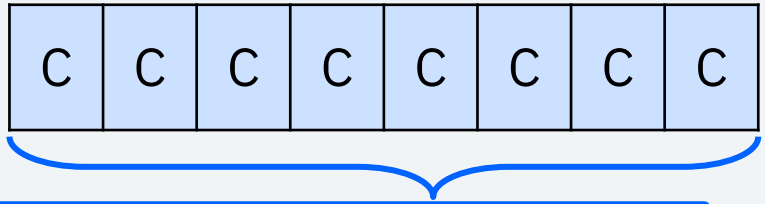
Bracketed TOD Clock – Introduction

Timestamp (approx.)	TOD Clock Value
1900-01-01	--
1935-09-06	--
1971-05-11	--
1971-05-11	X'8000 0000 0000 0000
2007-01-13	
2042-09-17	X'FFFF FFFF FFFF FFFF
2042-09-17	X'0000 0000 0000 0000
2078-05-23	X'4000 0000 0000 0000
2114-01-26	X'7FFF FFFF FFFF FFFF

If we do this, timestamps from the first bracket are impossible to represent.

Bracketed TOD Clock – Format

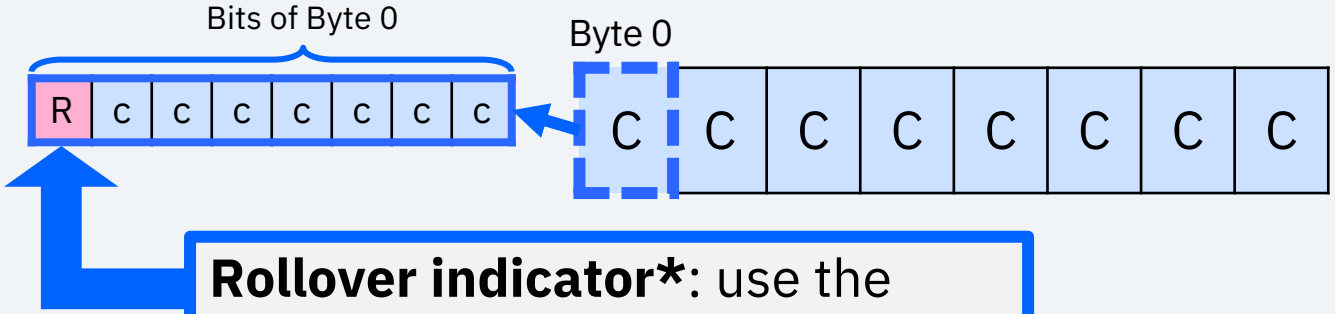
C – Clock value



Clock value: identical to the standard TOD clock, 8 bytes.

Bracketed TOD Clock – Format

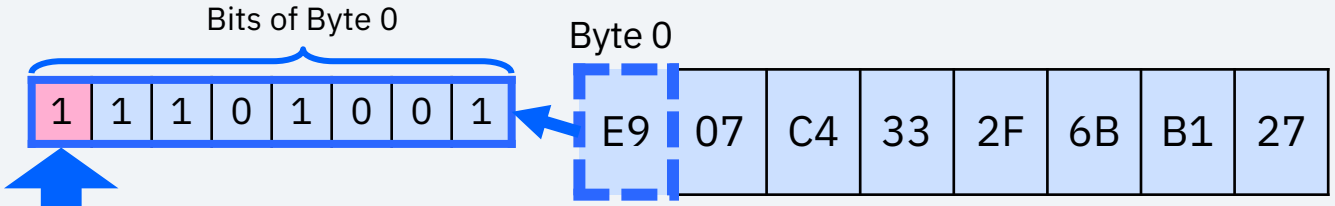
C – Clock value
R – Rollover indicator



Rollover indicator*: use the most significant (leftmost) bit to determine if a rollover occurred.

** The rollover indicator is considered part of the clock value!*

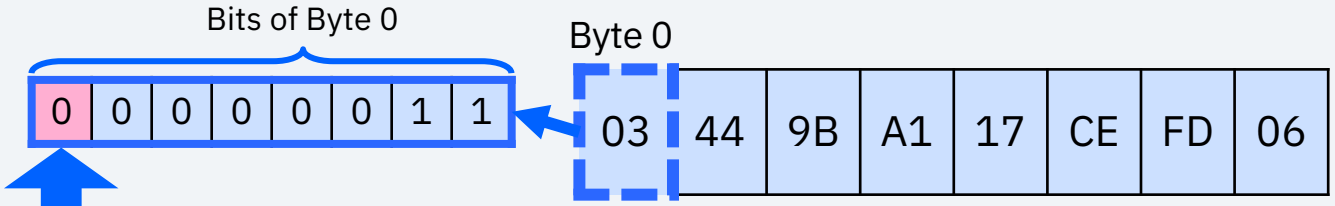
Bracketed TOD Clock – Example #1



If the rollover indicator is 1, we assume that the TOD clock ***did not*** overflow.

This is a timestamp in the year range of 1971 to 2042.

Bracketed TOD Clock – Example #2

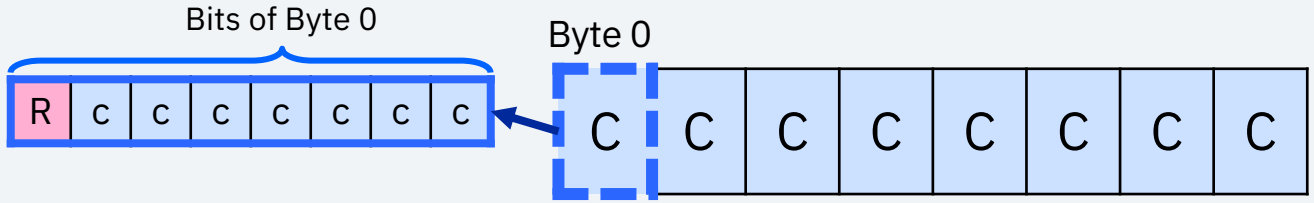


If the rollover indicator is 0, we assume that the TOD clock ***did*** overflow.

This is a timestamp in the year range of 2042 to 2114.

Bracketed TOD Clock – Technical Details

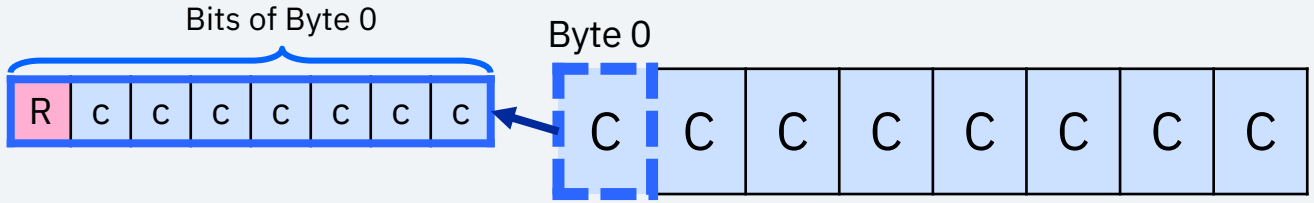
C – Clock value
R – Rollover indicator



- Its data type is **`tpf_BrkTOD_type`**.
- This format extends the duration of the standard TOD clock by approximately 71 years.
- Instead of representing timestamps during the years 1900 to 2042, this format represents timestamps during the years 1971 to 2114.

Bracketed TOD Clock – Technical Details

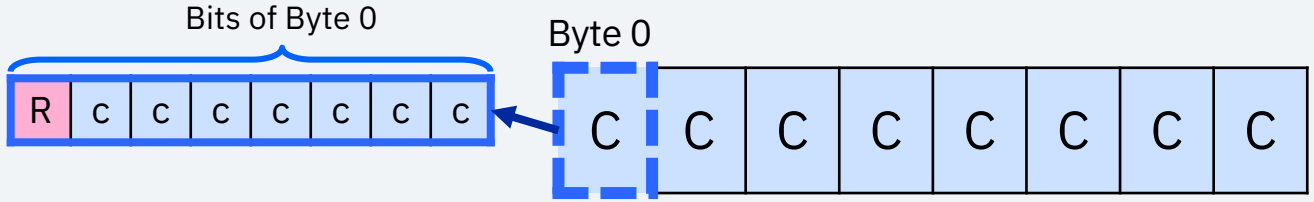
C – Clock value
R – Rollover indicator



- The most significant bit of the clock value is treated as a rollover indicator:
  - If the indicator is 0, the clock value represents the years 2042 to 2114.
  - If the indicator is 1, the clock value represents the years 1971 to 2042.

Bracketed TOD Clock – Technical Details

C – Clock value
R – Rollover indicator



- The main drawback when using the bracketed TOD clock format is that it is impossible to represent the years 1900 to 1971 with it.
  - This is because the raw clock values for 1900 to 1971 represent the years 2042 to 2114 instead.
- One benefit is that a database reorganization is not required if data prior to 1971 is not present.
- Another benefit is that using the STCK instruction still guarantees unique tokens.

Technical Details – API Functions

This project introduced 8 new functions as part of the z/TPF API:

Conversion to/from time_t:

- tpf_timet_to_BrkTOD
- tpf_timet_to_TODExtHi
- tpf_BrkTOD_to_timet
- tpf_TODExtHi_to_timet

Conversion to/from the extended TOD clock:

- tpf_STCKE_to_BrkTOD
- tpf_STCKE_to_TODExtHi
- tpf_BrkTOD_to_STCKE
- tpf_TODExtHi_to_STCKE

Example – tpf_BrkTOD_to_STCKE

```
#include <tpf/c_stcke.h>

tpf_BrkTOD_type brktod_val;
tpf_TOD_ext_type extod_result;

// Convert bracketed TOD to extended TOD
int rc = tpf_BrkTOD_to_STCKE(
   &brktod_val,
   &extod_result
);
```

Technical Details – Comparisons

There are also 3 new sets of C/C++ macros in c_stcke.h:

- `STCKE_xx (a, b)`
- `TODExtHi_xx (a, b)`
- `BrkTOD_xx (a, b)`

Each **'xx'** can be:

- **'EQ'**: `a == b`
- **'NE'**: `a != b`
- **'LT'**: `a < b`
- **'LE'**: `a <= b`
- **'GT'**: `a > b`
- **'GE'**: `a >= b`
- **'MV'**: `memcpy(a, b)`

Example – BrkTOD_LT

```
#include <tpf/c_stcke.h>

tpf_BrkTOD_type now_tod;
StructWithBrkTOD* bkts;

// Is the current TOD before the struct's TOD?
if (BrkTOD_LT(&now_tod, &bkts->brkTOD)) {
  updateStruct(&bkts, &now_tod);
}
```

Technical Details - Comparison

Bracketed TOD	Epoch TOD
• Represents the years 1971 – 2114.	• Represents the years 1900 – 10,000+
• Clock reads in this format are guaranteed to be unique.	• Uniqueness is **never** guaranteed.
• Internal data is equivalent to the standard 8-byte TOD clock format.	• Internal data is equivalent to the first 8 bytes of the extended TOD clock format.
• Another rollover issue will occur in 2114.	• No rollover issues will occur for tens of thousands of years.

To-be Scenario

Anna
application architect

After discussing what to do with each instance of an 8-byte standard TOD clock value with her team for the 2042 rollover, they reach a consensus on how to handle each one.

Anna leverages the bracketed TOD clock and epoch TOD clock formats to minimize the reorganization of her company’s database.

Value Statement

Both the bracketed and epoch TOD clock formats are solutions to the 2042 overflow issue.

The epoch TOD clock prevents overflow issues from occurring for the foreseeable future.

The bracketed TOD clock does not require changes to your database.

Conclusion

APAR PJ47142 (September 2023) delivered support for the bracketed and epoch TOD clock formats.

IBM is investigating which strategies to apply for each TOD clock value in the z/TPF system code.

Thank you