ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THSDSC03

INTEGRATE EPICS 7 WITH MATLAB USING PVACCESS FOR PYTHON (P4P) MODULE

K. Kim*, E. Williams, J. Bellister, K. Kim, M. Zelazny
SLAC National Accelerator Laboratory, Menlo Park, CA, USA

Abstract

MATLAB is essential for accelerator scientists en-gaged in data analysis and processing across diverse fields, including particle physics experiments, syn-chrotron light sources, X-ray Free Electron Lasers (XFELs), and telescopes, due to its extensive range of built-in functions and tools. Scientists also depend on Experimental Physics and Industrial Control Systems (EPICS) 7 to control and monitor complex systems. SLAC has developed Matpva, a Python interface to integrate EPICS 7 with MATLAB. Matpva utilizes the PVAccess for Python (P4P) module and EPICS 7 to offer a robust and reliable interface for MATLAB users that employ EPICS 7. The EPICS 7 PVAccess API al-lows higher-level scientific applications to get/set/monitor simple and complex structures from an EPICS 7-based control system. Moreover, Matpva simplifies the process by handling the data type con-version from Python to MATLAB, making it easier for researchers to focus on their analyses and innovative ideas instead of technical data conversion. By leverag-ing Matpva, researchers can work more efficiently and make discoveries in diverse fields, including particle physics and astronomy.

INTRODUCTION

MATLAB stands as an indispensable tool for accelerator scientists deeply immersed in the intricate realms of data analysis and processing, spanning an array of scientific domains such as particle physics experiments, synchrotron light sources, X-ray Free Electron Lasers (XFELs), and telescopic observations. Its indispensability is rooted in the vast array of built-in functions and tools it offers, empowering scientists to decipher complex data and derive meaningful insights.

In parallel, the world of scientific instrumentation and control systems relies heavily on Experimental Physics and Industrial Control Systems (EPICS) 7, a set of software tools and libraries widely used for building control systems in scientific research and industrial facilities [1]. Several attempts have been made to integrate EPICS into MATLAB, including projects like labCA and MATLAB Channel Access (MATLAB CA, MCA) [2, 3]. However, these were specifically designed for the EPICS Channel Access (CA) interface within MATLAB [4]. Consequently, they did not provide comprehensive support for EPICS PVAccess, which offers distinct advantages over CA [5]. These benefits include structured data types, enhanced security measures, efficient data transmission, and the ability to handle structured datasets, such as NTTable [6-8].

* ktkim@slac.stanford.edu

Efforts have been made to integrate PVAccess into MATLAB at SLAC using epicsCoreJava based on Java. However, this integration faced limitations because MATLAB still relies on Java 1.8.x, while the EPICS Java PVAccess support defaults to Java 11, making it incompatible with MATLAB. Moreover, Oracle JDK 8 with the premier support reached its end of life in March 2022. OpenJDK 8 is also slated to end support in November 2026.

Thus, SLAC has decided to integrate EPICS PVAccess with MATLAB using a well-supported language that aligns with their requirements, Python.

Python has emerged within the scientific community due to its numerous advantages, including readability, simplicity, versatility, a robust standard library, a vast community, compatibility, and seamless integration. As a result, numerous institutions and facilities have been transitioning to Python to leverage these benefits.

Acknowledging this shift, SLAC took a pioneering step by developing Matpva, a Python interface to integrate EPICS 7 with MATLAB. This work was also inspired by the labCA, which is the most widely used tool for integrating EPICS 3 with MATLAB. This innovative solution bridges the gap between two scientific powerhouses and provides researchers with a unified platform for their endeavors. Matpva utilizes Python's capabilities and integrates the PVAccess for Python (P4P) module and EPICS 7 [9]. This creates a sturdy and reliable interface for MATLAB users navigating the intricate realms of EPICS 7.

What sets Matpva apart, however, is its exceptional capability to streamline the cumbersome process of converting data types from Python to MATLAB, and vice versa. By doing this, Matpva empowers scientists to delve into their research with unwavering focus and efficiency. It serves as a catalyst for groundbreaking discoveries in the multifaceted realms of particle physics and astronomy propelling the boundaries of scientific knowledge ever further.

IMPLEMENTATION

The PVAccess data type comprises Normative Types, which are defined as structures consisting of both required and optional fields. Matpva stands out for its ability to convert PVAccess Normative Types from Python to MATLAB. In Listing 1, you'll find a code snippet that demonstrates how PVAccess Normative Types can be converted into MATLAB data types in the mpvaGet function.

Listings 1: Snippet of mpvaGet function as an implementation example.

```
% Bring P4P python module into MATLAB
MatP4P = py.p4p.client.thread.Context('pva',
pyargs('nt', false));
PV = MatP4P.get(pvname);
```

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THSDSC03

```
% Check the ID of PV
nt_id = string(getID(PV));
% Check the type of PV
t = struct(py.dict(type(PV))).value;

% To have timeStamp information in human readable form
TimeInSeconds = dou-
ble(int64(struct(struct(todict(PV)).timeStamp).sec-
ondsPastEpoch));
ts = datetime(TimeInSeconds, 'ConvertFrom',
'epochtime', 'Epoch', '1970-01-01', ...
    'Format', 'MMM dd, yyyy HH:mm:ss.SSS', 'TimeZone',
'UTC');
% To have alarm information
alarm.severity =
int32(int64(struct(struct(todict(PV)).alarm).sever-
ity));
alarm.status   =
int32(int64(struct(struct(todict(PV)).alarm).status));
alarm.message  =
string(struct(struct(todict(PV)).alarm).message);

% NTScalarArrays data type PVs
if (contains(nt_id, "NTScalarArray"))
    if (t == "ai")
        ret = int32(py.array.array('i',
struct(todict(PV)).value));
    elseif (t == "aI")
        ret = uint32(py.array.array('I',
struct(todict(PV)).value));
    elseif (t == "ab")
        ret = int8(py.array.array('b',
struct(todict(PV)).value));
```

MPVAGET

Calling Sequence

Listing 2: Syntax of mpvaGet function.

```
% When PV is NTScalar or NTScalarArray type
[PV, ts, alarm] = mpvaGet(pvname)

% When PV is NTTable type
[NTTable, ts, alarm, NTStruct] = mpvaGet(pvname)

% Skip unwanted outputs using tilde(~)
[PV, ~, ~] = mpvaGet(pvname);
```

Description

The mpvaGet function is used to retrieve the values of specified EPICS Process Variable (PV) names. When querying PVs are NTScalar or NTScalarArray type, the output consists of PV value, timestamp, and alarm. For PVs of NTTable type, the output includes PV value, timestamp, alarm, and NTStruct as illustrated in Listing 2.

In MATLAB, simply typing mpvaGet(pvname) displays the first output of the function. If you assign a single output, such as PV = mpvaGet(pvname), it captures the first value among the outputs. To selectively capture specific outputs, you can utilize the tilde (~) notation to skip unwanted outputs, as demonstrated in Listing 2. For detailed examples, refer to Listing 3.

Parameters

- PV: The values of specified EPICS PV names. It is one of the mpvaGet outputs when targeting PV is NTScalar or NTScalarArray type. When PV is NTScalar, it is a MATLAB numeric, string, or logical data type. When PV is NTScalarArray, it is a MATLAB numeric array, string array, or logical array data type.
- pvname: A Name of PV. It must be specified as a string data type. The pvname should be enclosed within double quotation marks ("").
- NTTable: The table values of given EPICS PV names. If the EPICS PV names targeted belongs to the NTTable type, the data type of acquired values is a MATLAB Table type.
- ts: Timestamp. It denotes the moment when the EPICS record was last processed. This timestamp is derived from the EPICS timestamps, which measure the elapsed time in seconds and fractional nanoseconds since 00:00:00 UTC, Jan.1, 1970.
- alarm: This encompasses the alarm status (HIHI, HIGH, LOW, and LOLO) and severity associated with the PV.
- NTStruct: The structure values of given EPICS PV names. When targeted EPICS PV falls under the NTTable type, mpvpaGet provides another output option, MATLAB Structure type for user.

MpvaGet Examples

Listing 3: mpvaGet function examples.

```
% When PV is NTScalar type
>> [PV, ts, alarm] = mpvaGet("TEST:PVA:IntValue")

PV =
  int32

   10

ts =

  datetime

   Sep 29, 2023 00:23:46.233

alarm =
  struct with fields:

    severity: 0
      status: 0
     message: ""

% When PV is NTScalarArray type and unwanted outputs,
ts and alarm, are skipped
>> [PV, ~, ~] = mpvaGet("TEST:PVA:FloatArray")

PV =

  1×6 single row vector

    1.5000    2.5000    3.5000    4.5000    5.5000
6.7000

% When PV is NTTable type and unwanted outputs, ts and
alarm, are skipped
>> [NTTable, ~, ~, NTStruct] =
mpvaGet("TEST:PVA:NTTable")

NTTable =

  4×2 table

     name      number
    _______    _______

    "test1"      1
    "test2"      2
    "test3"      3
    "test4"      4

NTStruct =

  struct with fields:
```

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THSDSC03


```
     name: ["test1"    "test2"    "test3"    "test4"]
   number: [1 2 3 4]
```

MPVAPUT

Calling Sequency

Listing 4: Syntax of mpvaPut function.

```
% When PV is NTScalar or NTScalarArray type
mpvaPut(pvname, value, "mpvaDebugOn");

% When PV is NTTable type
mpvaPut(pvname, field1, value1, field2, value2, ..,
"mpvaDebugOn");


% When PV is NTTable type, MATLAB table or structure
can be used as an input value
mpvaPut(pvname, struct/table, "mpvaDebugOn");
```

Description

The mpvaPut function is designed to place input values into the specified field within the given EPICs PV name. By default, this operation occurs silently without any display. However, if you want to view the previous and updated PV values, you can include the optional parameter mpvaDebugOn as the final input argument in the function, as demonstrated in Listing 4. Examples of its usage are provided in Listing 5 for reference.

Parameters

- pvname: A Name of PV. The data type should be a string.
- value: A value that is a numeric, string, logical, numeric array, string array, or logical array data type. The character vector is not allowed. Please use a string ("double quotation marks") instead of a character vector. Boolean values are either true or false. They don't require quotation marks and every word is lower cases.
- field1, field2, field3, …: Fields of given EPICS NTTable PV.
- value1, value2, value3, …: Values that are numeric array, string array, or logical array data types for the specific field in the given EPICS NTTable PV. The character vector is not allowed. Please use a string array ("double quotation marks") instead of a character vector. Bool values are either true or false. They don't require quotation marks and every word is lower cases.
- You can also use a MATLAB structure or table data type as input.

MpvaPut Examples

Listing 5: mpvaPut function examples.

```
% When PV is NTScalar type
>> IntValue = 0;
>> mpvaPut("TEST:PVA:IntValue", IntValue)


% When PV is NTScalarArray type
>> ab = [false, true, false, false];
>> mpvaPut("TEST:PVA:BoolArray", ab)

% When PV is NTTable type and mpvaDebugOn is selected
>> Name = ["device_1", "device_2"];
>> Number = [1.1, 1.2];
>> mpvaPut("TEST:PVA:NTTable", "name", Name, "number",
Number, "mpvaDebugOn")
The old PV is
old_PV =
  4×2 table
     name      number
    _______    ______

    "test1"      1
    "test2"      2
    "test3"      3
    "test4"      4

The update PV is
updated_PV =

  2×2 table

       name       number
    ___________    ______

    "device_1"     1.1
    "device_2"     1.2

% When MATLAB structure or table is used as a mpvaPut
input for the NTTable data type
>> NTTable = [NTTable; {"String5", 50};{"Test",10}];
>> mpvaPut("TEST:PVA:NTTable", NTTable)
```

MPVAMONITOR

Calling Sequency

Listing 6: Syntax of mpvaMonitor function.

```
mpvaMonitor(pvname);
```

Description

The mpvaMonitor displays the values of specified EPICS PV names as soon as they are updated. This monitoring process persists until the user decides to terminate it. You can find the syntax of this function in Listing 6, along with practical examples provided in Listing 7.

Parameters

- pvname: A Name of PV. The data type should be a string. Type a pvname between double quotation marks (").

MpvaMonitor Examples

Listing 7: mpvaMonitor function examples.

```
% When PV is NTScalar type
>> mpvaMonitor("KTEST:PVA:IntValue")
NEW: KTEST:PVA:IntValue   Sat Sep 23 22:22:16 2023   36
NEW: KTEST:PVA:IntValue   Sat Sep 23 22:22:17 2023   2
NEW: KTEST:PVA:IntValue   Sat Sep 23 22:22:18 2023   22

% When PV is NTTable type
>> mpvaMonitor("KTEST:PVA:NTTable")
NEW: KTEST:PVA:NTTable   Sat Sep 23 22:21:42 2023
name        number
--------  --------
device_1   1.66642
device_2   9.3474
NEW: KTEST:PVA:NTTable   Sat Sep 23 22:21:43 2023
name        number
--------  --------
device_1   7.89731
device_2   9.07301
```

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THSDSC03

MPVASETMONITOR

Calling Sequency

Listing 8: Syntax of mpvaSetMonitor function.

```
PV = mpvaSetMonitor(pvname);
```

Description

mpvaSetMonitor subscribes to the given EPICS PV using Python class and stores values in the cache when it is updated. The module includes a method (mpvaNewMonitorValue) to check if the PV is updated since it has been subscribed. The function's syntax is outlined in Listing 8.

Parameters

- PV: Instantiated Python class in mpvaSetMonitor.py module to monitor the pvname.
- pvname: A Name of PV. The data type should be a string. Type a pvname between double quotation marks (").

MPVANEWMONITORVALUE

Calling Sequency

Listing 9: Syntax of mpvaNewMonitorValue function.

```
mpvaNewMonitorValue(PV)
```

Description

mpvaNewMonitorValue returns false if the PV is not updated and returns true if it is updated. This function is especially valuable when the read operation consumes a significant amount of time, such as when dealing with large arrays or NTTables. mpvaNewMonitorValue function only works when mpvaSetMonitor is instantiated. The syntax of the function is in Listing 9, along with practical examples of mpvaSetMonitor and mpvaNewMonitorValue provided in Listing 10.

Parameters

- PV: Instantiated class in mpvaSetMonitor.py Python module to monitor a specified PV.

MpvaSetMonitor and MpvaNewMonitorValue Examples

Listing 10: mpvaSetMonitor and mpvaNewMonitorValue examples.

```
% Instantiate class in mpvaSetMonitor.py Python module
to monitor a pvname, TEST:PVA:IntValue
>> PV = mpvaSetMonitor("TEST:PVA:IntValue");

% Check if PV is updated
>> mpvaNewMonitorValue(PV)

ans =

  logical

   0

% Check the current PV value
>> mpvaGet("TEST:PVA:IntValue")

ans =

  int32

   777


% Put a new PV value
>> mpvaPut("TEST:PVA:IntValue", 10)

% Check if PV is updated
>> mpvaNewMonitorValue(PV)

ans =

  logical

   1

% Confirm the updated PV value
>> mpvaGet("TEST:PVA:IntValue")

ans =

   int32

   10
```

CONCLUSION

In summary, this paper introduces Matpva, a robust tool that seamlessly connects EPICS 7 and MATLAB using Python. Previous integration attempts lacked full support for EPICS PVAccess, which prevents scientists from efficient large data analysis.

Matpva fills this gap, providing a smooth interface between EPICS 7 and MATLAB through Python. It simplifies complex data type conversions, allowing scientists to concentrate on their research without getting entangled in technical intricacies.

The paper presents key functions like mpvaGet, mpvaPut, mpvaMonitor, mpvaSetMonitor, and mpvaNewMonitorValue. Through illustrative examples, these functions enable scientists to interact effortlessly with various types of EPICS PVs, ranging from NTScalar to NTTable.

Furthermore, it's worth noting that Matpva remains a dynamic and continuously evolving solution. Its source code is actively maintained and constantly improved, fostering a vibrant community of contributors and users. Anyone interested can access the source code freely from the GitHub repository [10], making it not only a powerful tool but also a collaborative effort within the scientific community.

In this paper, we've implemented a monitoring flag for detecting updates of PV after monitoring invocation. To enhance this functionality, future work will integrate a callback mechanism, leveraging MATLAB's built-in support for callbacks. This enhancement will enable specific actions when detecting changes in PV values.

REFERENCES

[1] EPICS 7, https://epics-controls.org/resources-and-support/base/epics-7

[2] labCA, https://till-s.github.io/epics-labca/

[3] Matlab CA, https://github.com/epics-extensions/matlab_ca

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THSDSC03

[4] EPICS Channel Access, https://epics-controls.org/resources-and-support/documents/ca

[5] PVAccess, https://epics-controls.org/resources-and-support/documents/pvaccess

[6] EPICS 7, pvAccess and pvData, https://docs.epics-controls.org/en/latest/pv-access/OverviewOfpvData.html

[7] G. White *et al.*, "The epics software framework moves from controls to physics," in *Proc. IPAC'19*, Melbourne, Australia, May 2018, pp. 1216-1218. doi:10.18429/JACoW-IPAC2019-TUZZPLM3

[8] K.-U. Kasemir, G. S. Guyotte, and M. R. Pearson, "EPICS V4 Evaluation for SNS Neutron Data", in *Proc. ICALEPCS'15*, Melbourne, Australia, Oct. 2015, pp. 947-949. doi:10.18429/JACoW-ICALEPCS2015-WEPGF105

[9] PVAccess for Python (P4P), https://mdavidsaver.github.io/p4p

[10] Matpva, https://github.com/slaclab/matpva/tree/master