ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEPD027

EPICS IOC EXTENSION POINTS: OLD, RECENT, AND PROPOSED*

A. N. Johnson†
Argonne National Laboratory, Lemont, IL, USA

Abstract

The EPICS Input/Output Controller (IOC) has always been extensible, enabling applications to add functionality without modifying the core software. Since the early EPICS 3.14 releases in 2002 the Core Developers have introduced ten new extension mechanisms that IOC application developers can use in individual IOCs or shared support modules. This paper reviews the plugin interfaces available in EPICS 7.0.9 and suggests a couple of areas where new extension points could be added in the future.

INTRODUCTION

EPICS (the Experimental Physics and Industrial Control System [1]) is a set of open-source software tools, libraries and applications used worldwide for building distributed soft real-time control systems for particle accelerators, telescopes and other large scientific instruments. From its beginnings in the early 1990s, the IOC software was designed for extensibility, so developers could add the site-specific functionality needed at their installation without changing the core [2].

Extensibility

To support extensibility, the common IOC components use standard interfaces so new parts can connect to existing ones without needing special glue code. Network links between IOCs use the same protocols and libraries that Graphical User Interfaces (GUIs) and other client programs use to access IOC data.

Over time the IOC's interfaces have been extended, and new ones added to support user needs and provide extra functionality. This paper highlights the plugin interfaces that exist, showing where the IOC can be extended without modifying the core software at all, and suggesting where some additional interfaces could be added.

HISTORY OF IOC EXTENSIONS

Early Releases

Versions of EPICS released before 3.14 in 2002 were configurable and extensible in several ways. The most frequently used was the ability to add new device support to the existing record types, allowing IOCs to be able to read and write to many kinds of hardware. Some sites created new record types for locally tailored functionality, but most found it quicker to combine sets of standard records (using existing hardware device support) than to develop new record types in C.

The following extension points existed prior to the development of the EPICS 3.14 release series:

- Database Definition (DBD) file: The IOC discovers component types from this configuration file.
- Database scanning: IOC-specific periodic scan rates, software events, and I/O event scanning.
- Database structures: Record, device and driver support from Base, support modules or IOC applications.
- User-defined breakpoint unit conversion tables.
- User C code called from subroutine records.
- Hook routines called during iocInit.
- Registerable error status codes with descriptions.
- Register a listener for error log messages.
- Locally added IOC and application templates.

EPICS 3.14: IOC Portability

The primary goal of the early 3.14 releases was to port the IOC code from VxWorks to run on other Operating Systems (OSs) [3-5], initially RTEMS, Solaris, Windows NT and Red Hat Linux. At the time most IOCs were based on VME and similar bus standards that those OSs didn't support, so no attempts were made to port device drivers.

Many changes replicated features provided by VxWorks, but with a generic interface. For example, the IOC Shell [6] replaced Wind River's Target Shell, adding an extension point for plugins to register commands and support for access to variables used for debugging and configuration.

IOC Construction and Configuration

To link all the necessary components into an IOC executable, the 3.14 build system generated C++ source code from information in the IOC's DBD file. That file specifies the components to be included, giving symbol names of the functions or data structures needed to access each component.

Table 1 shows examples of every DBD statement type that provides symbolic information, and the EPICS version that introduced it.

Table 1: DBD File Statements With Symbols

DBD Statement	Introduced
`recordtype(mbbi) { … }`	< 3.13
`driver(drvAsyn)`	< 3.13
`device(ao, INST_IO, devAoStats, "IOC stats")`	< 3.13
`registrar(rsrvRegistrar)`	3.14.2
`variable(dbRecordsOnceOnly, int)`	3.14.3
`function(generalTimeCheck)`	3.14.6
`link(const, lnkConstIf)`	3.16.1

* Work supported by the U.S. Department of Energy Office of Science, Office of Basic Energy Sciences, under Contract No. DE-AC02-06CH11357.
† Andrew Johnson <anj@anl.gov>

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEPD027

Compiling the C++ code into the IOC ensures that all the named components will be included, either directly or by linking to their shared libraries. The generated code allows the IOC to find the address of each symbol at run-time and thus initialize the associated support code.

Linking plugins into the executable like this provided an efficient way to include extensions. The DBD `link` statement was added in 3.16.1 to register JSON link types, but generic statements like `registrar(regFunc)` will be preferred for future extension points — this names a function to be called at initialization that adds the component into the IOC software.

Dynamic Module Loading Once OS support for runtime module loading and symbol lookup became available, the IOC was given generic interfaces to these and a `dlload` command. These allow facilities to start IOCs as a generic executable that loads the specific modules it needs from its startup script.

Extending Device Support

Prior to 3.14.8 the device support code that connects a record to its hardware and performs I/O could only be initialized once, when the IOC started. It would identify its hardware from the INP or OUT field, and maintain that connection until the IOC was shut down. Other link fields connected via database (DB) or channel access (CA) links could be updated at runtime though, as could soft device support using the INP or OUT link as a DB or CA link.

The 3.14.8 release added an interface called the Device Support Extension Table (DSXT) that device support code may implement to support address changes at runtime (see Fig. 1). The IOC checks the DSXT when something writes to the INP or OUT field of a record to determine whether to accept that write. The device provides an `add_record()` routine if it can accept new records after startup, and a `del_record()` routine if it can disconnect an active record from its hardware. Either routine may decline a change by returning an error code.

```
/** Device Support Entry Table */
typedef struct typed_dset {
  long number;
  long (*report)(int level);
  long (*init)(int after);
  long (*init_record)(struct dbCommon *prec);
  long (*get_ioint_info)(int detach,
     struct dbCommon *prec, IOSCANPVT *pscan);
  /* Reserved for record type additions */
} typed_dset;

/** Device Support Extension Table */
typedef struct dsxt {
  long (*add_record)(struct dbCommon *prec);
  long (*del_record)(struct dbCommon *prec);
  /* Reserved for future EPICS releases */
} dsxt;

DBCORE_API void devExtend(dsxt *pdsxt);
```

Figure 1: Definitions for the IOC's Device Support Entry Table and Device Support Extension Table.

Time and Event Sources

By 3.14.10 the IOC's old interface for fetching the time and synchronizing embedded OS clocks to a Network Time Protocol (NTP) server had proved unsuitable for Astronomy users and the new workstation OSs. Parts of the IOC were rewritten to allow time synchronization and event distribution plugins, creating a subsystem now called General Time. This allows multiple prioritized time and event providers to be running simultaneously. If the first time-provider is unable to respond the next will be asked until an answer is obtained. It enforces monotonic time (preventing it from moving backwards). The subsystem code tracks errors, and reports on the provider status.

Other Additions in 3.14

Array Subroutine records (aSub) The original subroutine record type allowed users to implement complex algorithms in C with minimal additional code, but only a limited amount of data could be provided to and returned from it. Gemini's genSub record type provided multiple array input and output links to relieve these limits, and was adopted with minor changes as the aSub record.

Field Modifiers The IOC's code that parses Process Variable (PV) names from client applications was extended [7] to recognize specific markers given after the field name and modify some aspects of those client connections. The first field modifier was released in 3.14.11 to circumvent a limitation of the Channel Access API, allowing existing client applications to read and write strings over 40 characters long without breaking network compatibility.

VMEbus Support (devLib) Earlier EPICS releases came with many drivers that directly called VxWorks APIs to discover and connect to their VMEbus I/O devices. A wrapper API called devLib was created with extra features, but remained unpopular until 3.14.3 when it was also implemented for RTEMS. Many VME drivers now call the devLib APIs and can be used on either embedded OS.

In 3.14.12 the devLib code was refactored to use a function table, allowing for plugins implementations. A plugin for a PCI to VMEbus bridge that had Linux support then let EPICS drivers for VME I/O boards be run on Linux.

RECENT EXTENSIONS

Channel Filters as JSON Field Modifiers

The field modifiers described in [7] suggested using JavaScript Object Notation (JSON) [8] to encode complex filtering requests for the server that could be unique to each client [9]. The framework to allow JSON filters to be registered and added to a channel on request was delivered in EPICS 3.15.1 with a set of 4 filters [10]. More recently two more filters have been added, and the functionality of one of the original filters was expanded.

Figure 2 shows some examples of a channel with various filters applied. The array filter has a shorthand notation that the IOC recognizes and translates.

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEPD027

```
channel.{"arr":{s:2,i:2,e:8}}
channel.[3:5]
channel.[3:2:-3]

channel.{"dbnd":{"abs":10}}
channel.{"dbnd":{"d":1.5}}

channel.{dec:{n:60}}

channel.{sync:{after:"e-"}}
channel.{sync:{last:"lap"}}
channel.{sync:{while:"blue"}}

channel.{"ts":{}}
channel.{"ts": {"str": "iso"}}
channel.{ts: {"num": "dbl"}}

channel.{utag:{m:1, v:0}}
```

Figure 2: Examples of filtered channels.

The Database Server API

The IOC's process database is only loosely bound to the RSRV CA server that implements communication with CA client applications and calls the database to connect to its process variables. The IOC calls the server to identify client users for debugging purposes though, and until 3.15.1 it could only call the RSRV server even though an IOC server for PV Access was already in development.

A dbServer plugin API was added so server layers could provide routines for client identification and to print status reports. The API was later expanded with routines for server initialization, startup and shutdown, and is now used by RSRV, the PV Access server QSRV in EPICS 7, and the QSRV2 server in the PVXS module.

JSON Link Types

The other major recent extension point allows new kinds of links to be created and used for record input-, output- and forward-link fields. Link instances are specified as a JSON map; the map key sets the link type, and its value can be any JSON value, including an array or nested map to accept multiple parameters.

One link type included gives users the ability to initialize array fields when the IOC starts up, a feature not previously possible. A "const" link as shown in Fig. 3 can accept numeric, string, or array values; then when the record initializes a field from it, the link provides the number, string or array to that field.

```
{const: 3.14159265359}
{const: "Pi"}
{const: [1, 2.7182818284, 3.14159265359]}
{const: ["One", "e", "Pi"]}
{const:[Inf, -Inf]})
```

Figure 3: Constant JSON link examples.

Link types can only get (input links) or put (output links) values for the record, or trigger processing (forward links), but links can be nested to combine or fan out values. A calculation link type can fetch values from multiple input links, perform calculations with the values, test alarm conditions, and send the result through an output link. A total of 5 JSON link types now come with 7.0.9; examples of the others can be seen in Fig. 4.

```
{calc: {expr:"PI*A*B", prec:3,
        args:[{pva:"dia"},
              {pva:"len"}]}}
{state:"redBeam"}
{state:"!simEnable"}
{debug:{state:"redBeam"}}
{trace:{state:"!simEnable"}}
```

Figure 4: Other link type examples.

Earlier ideas discussed in [11] to allow device support layers to define their own address formats have seen little demand recently and are unlikely to be pursued.

IDEAS FOR EXTENSION POINTS

Event Support

The current IOC's periodic scan rates are fixed at initialization time. New rates can't be changed or added later, and records can only be configured using the rates configured. The dbScan subsystem could be redesigned, but backwards compatibility for existing IOCs is paramount and would significantly complicate that task.

One change in 3.15.1 changed the dbCommon EVNT field from a short integer to a string, allowing Soft Events that trigger record processing to be named instead of allocating numbers for each different event. The string is free format other than a fixed length limit.

As the IOC's database file parser now accepts field values in JSON, an EVNT field can be set to a JSON object that gets stored in the string. Some development of that capability could create an event interface to support plugins that would be able to trigger record processing alongside the existing periodic scan mechanism.

Plugins could implement various record processing trigger mechanisms, either time-based or specific to some device support layers or network protocols. Some examples of JSON values for the EVNT field that could be implemented are shown in Fig. 5.

```
field(EVNT, {seconds: 1.5})
field(EVNT, {minutes: 10})
field(EVNT, {cron:{minute:[4,19,34,49],
                   hour:[9,12,15]}})
field(EVNT, {asyn:{port:"$(P)", event:"conn"}})
field(EVNT, {mqtt:{topic:"sr/bpm/+/alarm"}})
```

Figure 5: Suggested JSON event specifications.

Reserving soft-event names that start with an open brace character would avoid future conflicts with these JSON event specifications.

Record-type Metadata

Client applications currently have no way to discover what fields a record provides. The EPICS "capr.pl" script reads the DBD file built for its soft IOC but can't handle IOCs from another EPICS version or with external record types. The pvTree tool from CSS/Phoebus embeds a list of link fields from the standard record types but is similarly limited by external record types or changes to EPICS. Sites can configure both client tools for local use, but the ability to fetch metadata directly from the IOC would be better.

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEPD027

There are many ways for the IOC to provide that over a network channel. Encoding the metadata as a JSON list or map at build-time would allow publishing it as a static record-type attribute. The current attribute value limited of 40 characters could be overcome, and the JSON value published as a long string.

CONCLUSION

The IOC has expanded its extensibility while being porting to new OSs and application domains. Since 3.14 the core has gained many new extension points, allowing projects and developers to add local capabilities to IOCs without changing iocCore: from extending device support for device retargeting and allowing pluggable time and event providers in General Time, to long strings, better array processing using aSub records, and channel filters.

The dbServer abstraction and JSON link type interfaces allow the IOC to integrate with existing and future network protocols. Together, the extension mechanisms provided in EPICS 7.0.9 allow sites to tailor IOCs for local requirements and share support modules, while preserving backward compatibility and modernizing interfaces and IOC configuration.

The ideas proposed here for Event Support and client-accessible record-type metadata will be discussed with the community for potential inclusion in future releases.

REFERENCES

[1] EPICS, https://www.epics-controls.org

[2] M. Knott, D. Gurd, S. Lewis, and M. Thuot, "EPICS: A control system software co-development success story", *Nucl. Instr. and Meth. in Phys. Res. A,* vol. 352, iss. 1-2, pp. 486-491, 1994. doi:10.1016/0168-9002(94)91577-6

[3] M. Kraimer, "EPICS: Porting iocCore to Multiple Operating Systems," in *Proc. ICALEPCS'99*, Trieste, Italy, Oct 1999, paper WC1P01, pp. 33-35.

[4] M. R. Kraimer, J. B. Anderson, J. O. Hill, and W. E. Norum, "EPICS: A Retrospective on Porting iocCore to Multiple Operating Systems", in *Proc. ICALEPCS'01*, San Jose, CA, USA, Nov. 2001, paper WEBT002, pp. 238-240.

[5] R. Lange *et al.*, "EPICS: Recent Developments and Future Perspectives", in *Proc. ICALEPCS'03*, Gyeongju, Korea, Oct. 2003, paper TU601, pp. 278-281.

[6] W. E. Norum, "New Capabilities of the EPICS IOC Shell", in *Proc. ICALEPCS'03*, Gyeongju, Korea, Oct. 2003, paper MP504, pp. 60-62.

[7] A. N. Johnson and R. Lange, "Evolutionary Plans for EPICS Version 3", in *Proc. ICALEPCS'09*, Kobe, Japan, Oct. 2009, paper WEA003, pp. 364-366.

[8] D. Crockford, "The application/json Media Type for JavaScript Object Notation (JSON)," RFC 4627, July 2006, https://www.rfc-editor.org/info/rfc4627.

[9] R. Lange, L. R. Dalesio, and A. N. Johnson, "Advanced Monitor/Subscription Mechanisms for EPICS", in *Proc. ICALEPCS'09*, Kobe, Japan, Oct. 2009, paper THP090, pp. 847-849.

[10] R. Lange, L. R. Dalesio, and A. N. Johnson, "Adding Flexible Subscription Options to EPICS", in *Proc. ICALEPCS'11*, Grenoble, France, Oct. 2011, paper WEPKS020, pp. 827-829.

[11] A. Johnson, "Link Support: The future of Device Support & Soft Links?", APS, Argonne National Laboratory, Lemont, IL, USA, May 1999, https://epics.anl.gov/docs/lnkSup.html