Data Addressing Specification

Specification

Revision:	1.2
Status:	Released
Project:	E-ELT Core Integration Infrastructure Software
File:	DataAddressingSpecification.docx
Document ID:	CSL-DOC-17-147264
Owner:	Matej Šekoranja, Cosylab Sweden
Last modification:	March 23, 2018
Created:	August 25, 2017

Prepared by	Reviewed by	Approved by
Matej Šekoranja, CSL SWE Niklas Claesson, CSL SWE	Aljaž Podboršek Gregor Čuk	Gregor Čuk

Document History

Revision	Date	Changed/reviewed	Section(s)	Modification
0.1	2017-08-30	Niklas Claesson	Tango, CSS	
1.0	2017-09-03	Matej Šekoranja	All	
1.1	2017-09-19	Matej Šekoranja	2.	opc.da renamed to opc.rr, note on JSON escaping
1.2	2008-03-22	Matej Šekoranja	2. Appendix A.	IPv6 support

Confidentiality

This document is classified as a confidential document. As such, it or parts thereof must not be made accessible to anyone not listed in the Audience section, neither in electronic nor in any other form.

Scope

This document is a Data Addressing Specification for the E-ELT Core Integration Infrastructure Software project.

Audience

All Cosylab and ESO employees involved with the E-ELT Core Integration Infrastructure Software.

Table of Contents

Figures

No table of figures entries found.

Tables

References

[1] EPICS, http://www.aps.anl.gov/epics/index.php

[2] Channel Access, http://www.aps.anl.gov/epics/docs/ca.php

[3] Tango Controls, http://www.tango-controls.org/

[4] Taurus, http://www.taurus-scada.org/en/latest/

[5] List of valid Taurus URIs, https://github.com/taurus-org/taurus/blob/develop/lib/taurus/core/test/test_taurushelper.py

[6] Control System Studio, http://controlsystemstudio.org

[7] EPICS v4, http://epics-pvdata.sourceforge.net

[8] URI specification, https://www.ietf.org/rfc/rfc2396.txt

[9] Format for Literal IPv6 Addresses in URL's, https://tools.ietf.org/html/rfc2732

Abbreviations

Table 1: List of Abbreviations

Abbreviation	Term
CII	Core Integration Infrastructure
HTTP	Hyper Text Transfer Protocol
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
OPI	OPerator Interface
PV	Process Variable
RPC	Remote Procedure Call
URI	Uniform Resource Identifier

1. Review of existing CS using URI

This chapter reviews usage of URI as data address format as used in different control systems.

1.1. Control System Studio

Control System Studio (CS-Studio) [6] is a front-end GUI widely used in EPICS [1] community, although CS-Studio is highly pluggable platform and therefore not limited to EPICS. CS-Studio uses URIs to address process variables (PV-s) to be control system independent, i.e. a plugin is chosen by URI schema. Supported syntax is:

```
schema://pv-name
```

The syntax above is not a correct way of using an URI. If parsed as URI, `pv-name` would be parsed as authority with no path component (unless it contains slash character(s)). `pv-name` is simply parsed as a string after "://".

Known URL schemas:

- ca://
- file://
- pva://
- loc://
- sim://
- sys://

This chapter briefly describes the different types.

1.1.1. Channel Access

The Channel Access [2] is the default transport protocol for the EPICS control system. The type of data depends on the PV used.

The PV can be either a record name only or a record.field combination (since Channel Access supports connecting to a record.field):

- ca://some:record:name (colon ':' is a valid PV name character)
- ca://some_record_name
- ca://some_record_name.SCAN

1.1.2. File

The `file://` scheme is used for reading the value of the PV from the file on the disk, which again is very useful in testing. It is possible to read the tables, images and lists from a specified file.

1.1.3. pvAccess

The `pva://` scheme is used for accessing the EPICS v4 pvAccess records. pvAccess supports PVs that are structures. The plugin scans (standard string processing) for "?request=" substring in given `pv-name` to extract specification of subfields to be requested.

Example: `pva://adc01?request=field(value,timestamp)`.

1.1.4. Local PVs

The `loc://` scheme is used to create local PVs inside the CS-Studio. CS-Studio screens can then through various mechanisms update the value of the local PVs, while other OPIs can read them.

1.1.5. Simulated PVs

The CS-Studio implements some simulated PVs, which can be used for testing the GUI functionality if the user is developing the OPI screen but does not have the access to the live control system available. Parameters to simulation functions are given as part of a `pv-name`, e.g. `sim://sine(1,10,3)`.

1.1.6. System

The CS-Studio offers access to various system values through a predefined set of system PVs, or to all JVM properties. The scheme used is `sys://`.

- sys://time
- sys://free_mb
- sys://used_mb
- sys://max_mb
- sys://user
- sys://host_name
- sys://qualified_host_name

1.2. Tango

This section describes the URIs that can be used in the Tango control system, mainly in its Taurus tool.

1.2.1. Tango Device Naming

In tango control system a four field namespace has been adopted for specifying a Tango device instance.

[//FACILITY/]DOMAIN/CLASS/MEMBER

- Facility: control system instance (usually omitted)
- Domain: sub-system
- Class (family): class or device family

- Member: device instance

Since facilities usually run only a single instance of the Tango control system, the first part of the device name is usually omitted.

A Tango device runs on a Tango device server, however the Tango control system has a single point of entry to everything related to the Tango control system, and this is the Tango database.

1.2.2. Tango database

The Tango database is exposed to the Tango control system participants through the database device server. Usually all Tango device servers in the system have an environment variable TANGO_HOST defined which contains the hostname and port number of the database device server, separated by the ':' character.

```
TANGO_HOST=host1:10000
```

1.2.3. Tango URIs

URIs are not directly defined as a part of the core Tango control system, however the most widely used Tango control system front-end GUI (Taurus).

In Taurus access to all Tango control system attributes (data points) is made through the database device server. The list of valid tango URIs can be deduced from [5]. The Taurus URI does not support the use of facility in the device name.

The Tango control system URI is defined as follows:

```
  1            2                3               4               5
tango://<database_ds>/<device_instance>/<attribute>[#<attr_property>]
```

1. The URI scheme for the Tango control system is `tango`
2. The hostname and the port of the Tango database device server
3. The three parts of the tango device instance name (the `DOMAIN/CLASS/MEMBER`)
4. The name of the attribute defined for the devices of this class
5. The name of the attribute property

Example:

```
tango://host1:10000/sys/tg_test/1/ampli#label
```

Where the parts of the URI are:

- `tango` – scheme
- `host1:10000` – the Tango database device server
- `sys/tg_test/1` – the device server instance
- `ampli` – the device attribute
- `label` – the attribute property

1.3. EPICS v4 URI

EPICS v4 [7] itself does not use URI, however its command-line tool `eget` has implemented the ability to accept them. The `eget` tool is a tool for value retrieval. It supports pluggable implementations of the pvAccess API, where URI is used to specify a request via command-line. A schema part of an URI determines what implementation to use. Currently there are 2 supported implementations: ”pva” for pvAccess and “ca” for Channel Access. A standard URI server-based naming authority is being used for both of them, i.e. a form of

```
<scheme>:[//<authority>]</path>[?<query>][#fragment]
```

where authority is in a form of:

```
[<user>@]<host>[:<port>]
```

If authority part is omitted then discovery mechanism is used. Channel Access does not support query and fragment part. pvAccess supports RPC mechanism. If query part is specified a RPC call will be made, instead of standard value retrieval. A query syntax follows a standard HTTP GET syntax, e.g. `name1=value1&name2=value2`. Since pvAccess supports structures as data types, fragment part was planned to be used to provide a pvAccess specific string specifying a subset of structured fields, but it was never implemented.

2. CII Data Addressing URI Specification

CII should use URI [8] for data addressing format. URI provides a simple and extensible means for identifying a resource. However, its simplicity and extensibility usually leads to misuse of the URI. CII should strictly use URI as specified in the document IETF RFC 2396 [8]. This preserves compatibility with URI parsers, tools and avoids re-inventing a new URI-like format and special libraries for its processing.

The URI syntax is dependent upon the scheme. In general URI are written as follows:

```
<scheme>:<scheme-specific-part>
```

where <scheme-specific-part> interpretation depends on the <scheme>.

The URI syntax does not require that the <scheme-specific-part> have a specific syntax, however CII URI syntax should follow the following form (absolute hierarchical URI reference):

```
<scheme>:[//<authority>]</path>[?<query>][#fragment]
```

<scheme> is a case-insensitive string that identifies service type and middleware.

The following table presents non-complete set of CII schemas:

Table 2 CII schemas

Scheme Value	Description
dds.ps	DDS middleware publish subscribe profile
dds.rr	DDS middleware request reply profile
opc.rr	OPC/UA middleware data access profile.
cii.log	Artefact of the CII Log (e.g. message or process)
cii.cgf	Artefact of CII Configuration (e.g. root node or leaf of configuration)
cii.oldb	Artefact of CII Online Database (e.g. root node or data point)
zpb.ps	ZMQ/Protobuff middleware publish subscribe profile

<authority> component is a top hierarchical element and is, if specified, either *server-based* or *registry-based*. A server-based authority parses according to the familiar syntax:

```
[user-info@]host[:port]
```

IPv6 addresses are encoded according to the RFC2732 [9] (see last example in Appendix A).

An authority component that does not parse in this way is considered to be registry-based. In this case the authority is specific to the URI scheme (e.g. in case of “dds.ps” scheme, authority can specify a DDS domain).

<path> component contains data, organized in hierarchical form, that appears as a sequence of segments separated by slashes. A path must always begin with a slash (opaque URI syntax should not be supported by CII, due to the limitation that opaque URI is not subject to further processing, i.e. can not have <query> component).

<query> component, if specified, is a string of (non-hierarchical) information to be interpreted by the resource. Its syntax is not defined by URI. Convention is most often a sequence of attribute–value pairs separated by an ampersand delimiter “&”. CII shall follow this convention. In cases more complex query strings (URI for an RPC call that has a complex structure type arguments) JSON-formatted string should be used. To identify this case, query string must start with an open brace ‘{‘ character (in escaped form).

<fragment> component, if specified, contains an additional information used to perform a retrieval action on the identified resource. <path> component already fully identifies a resource, however <fragment> component specifies narrowed view within that resource. Syntax is <scheme> specific.

See Appendix A for examples of CII URIs.

A CII URI form provides unambiguous way of addressing entities in a completely standard way. Standard URI parsers shall be used. There is no need for any additional parser/processing implementation.

Since URIs are scheme specific a tiny layer might be added to help constructing and interpreting specific URI, however this should be a part of specific library for each scheme.

See Appendix B for an example code of such API.

Appendix A. Examples of CII URIs

```
# ZMQ/PB pub/sub, publisher = m4lsv.pl.eso.org:5001, topicName = BrickStatus
zpb.ps://m4lsv.pl.eso.org:5001/BrickStatus

# ZMQ/PB req/rep, server = m4lsv.pl.eso.org:5001, service/device = Mount01,
# ‘move’ operation to given az/dec
zpb.rr://m4lsv.pl.eso.org:5001/Mount01?op=move&az=12.3&dec=34.5

# DDS pub/sub, domain = m1, topicName = CabinetTelemetry,
# sample key CableId = M1-011
dds.ps://m1/CabinetTelemetry?CableId=M1-011

# CII configuration, local (no authority specified)
# path = /m1/SegConc, Id = 232, attribute = numSegments
cii.cfg:/m1/SegConc?Id=232#numSegments

# same as above from remote server instead of local
cii.cfg://configsrv.eso.org/m1/SegConc?Id=232#numSegments

# retrieve a log with given UUID from default Log service
cii.log:/Log?uuid=123e4567-e89b-12d3-a456-426655440000

# retrieve SEVERE logs within given time from default Log service
cii.log:/Log?from=2017-09-01T12:00&to=2017-09-02T08:00&level=SEVERE

# ZMQ/PB pub/sub, publisher with IPv6 address
# FEDC:BA98:7654:3210:FEDC:BA98:7654:3210 and port 5001,
# topicName = BrickStatus
zpb.ps://[FEDC:BA98:7654:3210:FEDC:BA98:7654:3210]:5001/BrickStatus
```

Appendix B. Example of “dds.ps” URI utility API

```
public static URI createPublishSubscribeURI(
        String domainId, String topicName,
        String sampleKeyId, String keyValue)
    throws URISyntaxException {
    return new URI("dds.ps", domainId, "/" + topicName, sampleKeyId + "=" +
keyValue, null);
}


static class PublishSubscribeDataAddress {
    private final String domainId;
    private final String topicName;
    private final String sampleKeyId;
    private final String keyValue;

    public PublishSubscribeDataAddress(
            String domainId, String topicName,
            String sampleKeyId, String keyValue) {
        this.domainId = domainId;
        this.topicName = topicName;
        this.sampleKeyId = sampleKeyId;
        this.keyValue = keyValue;
    }

    // accessors...
}

public static PublishSubscribeDataAddress parsePublishSubscribeURI(String uri)
    throws URISyntaxException {

    URI parsedURI = new URI(uri);
    if (!parsedURI.getScheme().equals("dds.ps")) {
        throw new RuntimeException("invalid scheme");
    }

    String domainId = parsedURI.getAuthority();
    if (domainId == null) {
        throw new RuntimeException("no authority");
    }

    String topicName = parsedURI.getPath();
    if (topicName == null || topicName.isEmpty()) {
        throw new RuntimeException("no path");
    }
    // remove leading slash (required if authority is present)
    topicName = topicName.substring(1);

    String sampleKeyId = null;
    String keyValue = null;
    String query = parsedURI.getQuery();
    if (query != null && !query.isEmpty()) {
        String[] s = query.split("=");
        if (s.length == 2) {
            sampleKeyId = s[0];
            keyValue = s[1];
```

```
        } else {
            throw new RuntimeException("invalid query");
        }
    }

    return new PublishSubscribeDataAddress(
            domainId, topicName,
            sampleKeyId, keyValue
    );
}
```