

# TesLA Device Control Adapter (DCA)

## DCA Implementation Specification



### Document Information

Authors	Cliff Hannel
Last update	12-Jul-09
Version number	Version (file rev 3)
File Name	DCASDK.docx ( <a href="#">most current posted version</a> )
Documents incorporated by reference	<a href="http://www.teslaalliance.org/standards/dca/TesLACommandDefinition.xsd">http://www.teslaalliance.org/standards/dca/TesLACommandDefinition.xsd</a> <a href="http://www.teslaalliance.org/standards/dca/TesLAModuleDefinition.xsd">http://www.teslaalliance.org/standards/dca/TesLAModuleDefinition.xsd</a> <a href="http://www.teslaalliance.org/standards/dca/teslaDCAEnums.xml">http://www.teslaalliance.org/standards/dca/teslaDCAEnums.xml</a> <a href="http://www.teslaalliance.org/standards/dca/AutoDocs/TMDAutoDoc.html">http://www.teslaalliance.org/standards/dca/AutoDocs/TMDAutoDoc.html</a> <a href="http://www.teslaalliance.org/standards/dca/AutoDocs/TCDAutoDoc.html">http://www.teslaalliance.org/standards/dca/AutoDocs/TCDAutoDoc.html</a> <a href="http://teslaalliance.pbworks.com/f/TesLACommonArchitectureRoadmap.pdf">http://teslaalliance.pbworks.com/f/TesLACommonArchitectureRoadmap.pdf</a> (for reference only)
Keywords (tags)	DCA, TesLA, Plug-in, SDK

## Revision History

Version	Date	Who	Description
0.5	4/1/09	Cliff Hannel	Initial in-process draft for review. Incorporates input from Fanfare (Kingston Duffie strawman) and Ixia
0.6	4/9/09	Cliff Hannel	Posted for intermediate TesLA review (in process w/ significant gaps)
0.9	6/22/09	Cliff Hannel	Posted for review, feedback for final revisions and ratification

## Contents

Contents.....	3
Overview: Providers, Consumers & Drivers.....	4
Objectives.....	5
Conventions used in this Document.....	5
Definitions.....	6
The DCA Driver Model.....	6
Installation & Initialization Procedure.....	7
Use of XML.....	8
TesLA XML Namespaces.....	8
TesLA Supported Data Types.....	9
Pre-defined Enumerations.....	10
Use of URIs, Base Path for Relative URIs.....	11
Use of XML Includes.....	11
DCA Help and Icon Files.....	11
Module Types.....	11
What Should Be Included in a DCA.....	12
Customizing or Extending Command Sets.....	13
Vendor (proprietary) Extensions.....	13
The TesLA Module Definition (TMD) File.....	15
Versioning & Compatibility.....	16
Design- and Run-time Version Checking.....	17
Command Aliases.....	18
TesLA Command Definition (TCD) Files.....	19
The Session.....	19
Command File Structure.....	19
Command Classification & Mandatory Commands.....	20
Bindings & Procedure Calls.....	22
TCL Binding Notes (to author of TCL Binding Spec).....	24
Command Groups and Command Definitions.....	24
Command Parameters & Responses.....	26
Interacting with Long-Running Commands.....	29
Code Lookups.....	29

## Overview: Providers, Consumers & Drivers

The Test Lab Automation Alliance (TesLA – [www.teslaalliance.org](http://www.teslaalliance.org)) is a multi-vendor organization that creates standards to facilitate interoperation of the hardware and software used in network test labs. This document defines a model that allows hardware vendors, software manufacturers, third parties and end users (Providers) to expose standardized interfaces so that other applications and components (Consumers) within a Network Test Harness may access their capabilities at run-time without component-specific design-time integration. These TesLA-compliant components and/or applications are referred to in this document as Modules, each of which may be Providers, Consumers or both. This document describes how Providers expose their capabilities through an XML-based Device Control Adapter (DCA) and how Consumers discover and use those services. For an overview of the different types of TesLA Modules and how they interact, see <http://teslaalliance.pbworks.com/Big-Picture-Meeting-Notes>.

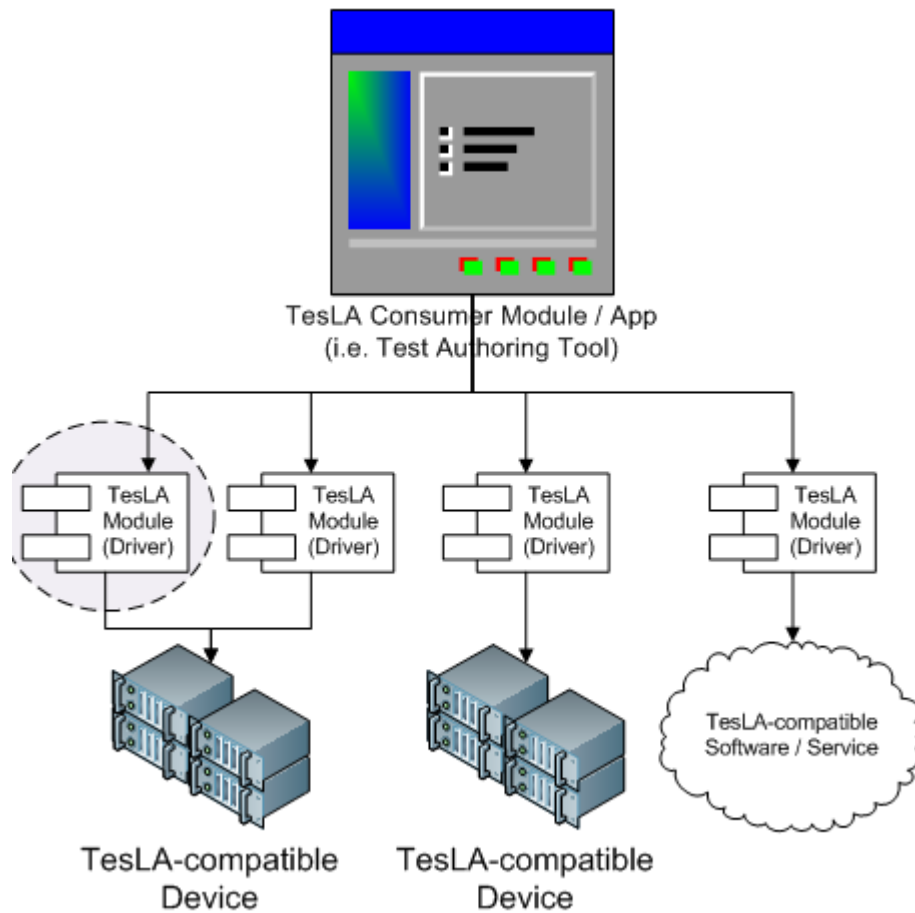


Figure 1: Consumers and Providers

## Objectives

The following is a summary of the high level objectives for the DCA driver model:

- Provide an extensible means for vendors to publish a wide range of TesLA-compatible Modules
- Provide a flexible and extensible means for TesLA Consumers to discover the presence and capabilities of TesLA Providers
- Provide a means for Customers and/or service providers to customize and extend the library of Modules available to TesLA Consumers
  - Provide a means of installing new versions of Modules in a customized environment without overwriting customizations
- Allow for Commands that are common across Modules of all Types
- Allow for Commands that are common across Modules of a specific Type
- Allow for Commands that are unique to a Module vendor
- Allow for differentiation between vendors' implementations of similar functions
- Allow operation on multiple popular platforms
- Facilitate integration with multiple popular programming and scripting languages

## Conventions used in this Document

- **Items requiring discussion or resolution are highlighted in yellow.** [these are being left in the document during final review to provide background and context to some of the decisions made]
- Programmatic elements such as environment variables and XML element or attribute names are shown in Courier font.
- Capitalized terms are defined in this document or in <http://teslaalliance.pbwiki.com/f/TesLACommonArchitectureRoadmap.pdf>
- XML and XSD diagrams were created from source XML documents and incorporated into this specification using Altova XMLSpy version 2009sp1.

In many standards track documents several words are used to signify the requirements in the specification. These words are capitalized. The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", in this document are to be interpreted as described below.

*Note that the force of these words is modified by the requirement level of the document in which they are used. An implementation is not compliant if it fails to satisfy one or more of the MUST requirements for the protocols it implements. An implementation that satisfies all the MUST and all the SHOULD requirements for its protocols is said to be "unconditionally compliant"; one that satisfies all the MUST requirements but not all the SHOULD requirements for its protocols is said to be "conditionally compliant."*

- **MUST:** This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.
- **MUST NOT:** This phrase, or the phrase "SHALL NOT", mean that the definition is an absolute prohibition of the specification.

- SHOULD: This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- SHOULD NOT: This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
- MAY: This word or the adjective "optional" means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.

## Definitions

Term	Definition
Device	Any hardware or software Provider of services that can be exposed by a DCA.
Module	Any component within a Network Test Harness.
DCA	The files that adhere to this standard and make up a Device Control Adapter.
Provider	The Module that provides a service via a DCA.
Consumer	The Module that uses a service via a DCA.
Test	A sequence of DCA commands and the logic that combines them.

## The DCA Driver Model

A Device Control Adapter (DCA) is composed of a combination of configuration and executable files as illustrated in Figure 2: TesLA . Together, these provide a “TesLA Device Driver”, allowing TesLA-compatible *Consumers* (Applications and other Modules) to access the capabilities of the associated *Providers* at run-time without prior design-time knowledge or integration on a Device-by-Device basis.

[NOTE: this procedure may be very similar to what is used for other Module types such as TATs and TRSs – we should consider generalization of the TMD for Module definition and discovery and the TCD for exposing functionality of ALL MODULE TYPES (since any Module may expose functionality in a DCA-like fashion). For example, while a TRS is not a Device per-se, it could be discovered using the same method described here and expose functionality in the same way as well.]

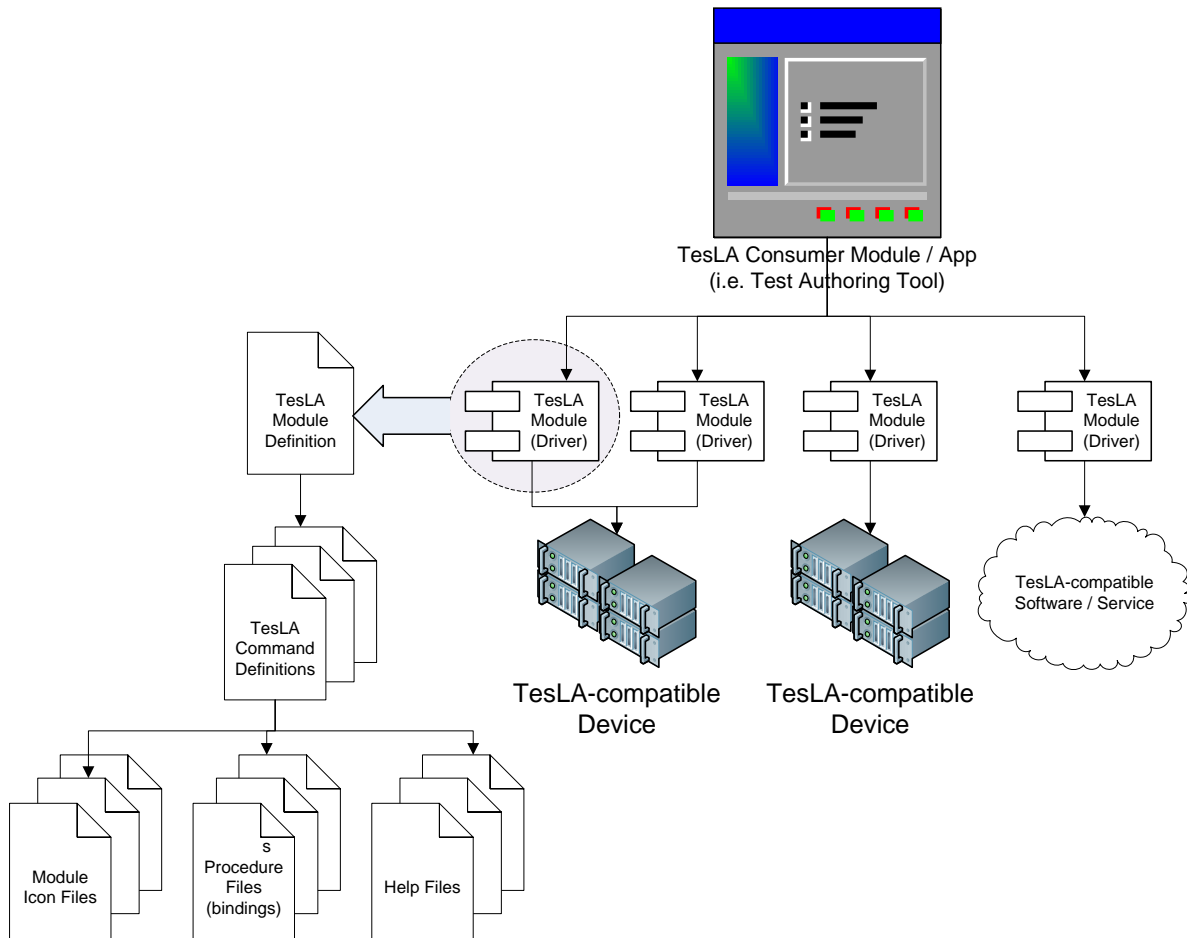


Figure 2: TesLA DCA Driver Files

## Installation & Initialization Procedure

The following procedure **MUST** be used at the time a TesLA-compatible Consumer loads as part of its initialization. Files **MAY** be re-processed upon detection of changes at run-time (without restarting) at the discretion of the DCA Consumer.

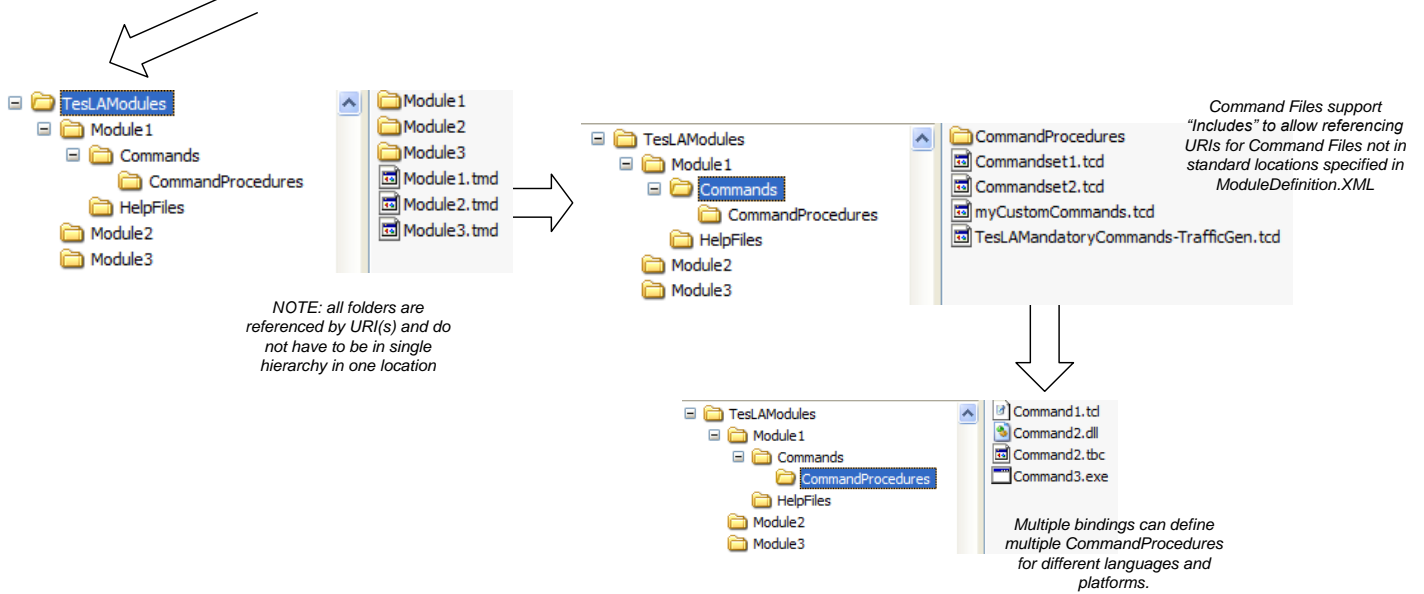
**Module Discovery:** In order to facilitate the ability of any Module to discover any other Module, the following procedure **MUST** be followed. The Consumer or Provider Module **MUST** look for the Environment Variable on the installation target machine named `TesLAModules` that specifies one or more semi-colon delimited URIs for locations of Provider TesLA Module Definition (TMD) files. If missing, the location(s) **MUST** be prompted for and the Environment variable set in a persistent way. For Consumers, Module Definitions at these URIs and all sub-directories **MAY** be loaded at this discretion of the Consumer (i.e. the Consumer is not obligated to expose all installed Modules). For Providers, TesLA Module Definitions **MUST** be installed at one of these locations, an existing or new sub-directory of one of these locations or a new location that **MUST** be appended to the current `TesLAModules` environment variable. **[ISSUE: HOW DO WE LIMIT LENGTH OF ENV VAR/LOCATIONS?]**

**Command Loading:** The TMD contains URI(s) at which TesLA Command Definitions (TCDs) for the Module can be found. The Consumer **SHOULD** process all such files, including files at locations included by reference using `includeCommandURI` element(s) within Command Files at the TMD-specified URIs as described in the Customizing and Extending Commands section below.

```

D:\WINDOWS\system32\cmd.exe
D:\>set TesLAModules
TesLAModules="file:///c:/TesLA/Modules;file:///myserver/sharename/TesLA/Modules"

```



**Figure 3: Module Definitions & Related Files**

## Use of XML

The structures of the TesLA Module Definition and TesLA Command Definition Files are described using [W3C XML Schema Definition Language \(XSD\) 1.1](#). All DCA TesLA Module Definitions MUST use XML per the schema defined in [TesLAModuleDefinition.xsd](#) and [TesLACommandDefinition.xsd](#) and comply with the [W3C XML 1.1 specifications](#). While it is RECOMMENDED that Consumers use this XSD file at run time to validate and interpret the corresponding XML files, this is not required.

## TesLA XML Namespaces

One of the primary motivations for defining an XML namespace is to avoid naming conflicts when using and re-using multiple vocabularies. XML Schema is used to create a vocabulary for an XML instance, and uses namespaces heavily. Thus, having a sound grasp of the namespace concept is essential for understanding XML Schema and instance validation overall. Although a namespace usually looks like a URL, that doesn't mean that it has to be or that one must be connected to the Internet to actually declare and use namespaces. Rather, the namespace is intended to serve as a virtual "container" for vocabulary and un-displayed content that can be shared in the Internet space. In the Internet space URLs are unique—hence you would usually choose to use URLs to uniquely identify namespaces. Typing the namespace URL in a browser doesn't mean it would show all the elements and attributes in that namespace; it's just a concept.<sup>1</sup>

In order to avoid the need centralized name assignment during the independent development of TesLA Modules by multiple vendors, each Module MUST provide a unique namespace that manages independent of the Alliance. It will be up to the Provider to avoid name collisions within that namespace. In addition to the Module-specific namespaces, the Alliance will maintain namespaces for common elements between Modules and specific types of Modules where the standards require them. Providers SHOULD post documentation on the relevant Module at the URI used to define each namespace.

<sup>1</sup> [http://www.oracle.com/technology/pub/articles/srivastava\\_namespaces.html](http://www.oracle.com/technology/pub/articles/srivastava_namespaces.html)

While XML allows the specification of Namespaces at the Element level, it is recommended that TesLA Modules only declare namespaces at the Module level.

Namespace	Description
<a href="http://www.teslaalliance.org/standards/dca/">http://www.teslaalliance.org/standards/dca/</a>	Top level namespace for all TesLA DCA standards compliant documents and APIs. All common elements across all TesLA Modules must have unique names within this domain.
<a href="http://www.teslaalliance.org/standards/dca/[ModuleType]">http://www.teslaalliance.org/standards/dca/[ModuleType]</a>	Each Module Type will have a namespace for elements that are common to all Modules of that Type (such as PacketGenerator, AuthoringTool or NetworkEmulator). A directory of ModuleTypes will be maintained at <a href="http://www.teslaalliance.org/standards/dca/teslaDCAEnums.xml">http://www.teslaalliance.org/standards/dca/teslaDCAEnums.xml</a>
<a href="http://[VendorDomain]/TesLA/standards/[DCAName]/[DCAVersion]">http://[VendorDomain]/TesLA/standards/[DCAName]/[DCAVersion]</a>	Each Vendor will have a namespace for each DCA they deliver. Element names must not be duplicated within a Module Type for a Vendor.

### ***TesLA Supported Data Types***

Parameters and Responses as defined in this specification use XML. It is expected that both Providers and Consumers will leverage available software libraries for XML parsing. Instead of defining our own set or subset of data types, TesLA will support the existing [W3C XML Schema built-in datatypes](#) as illustrated below.

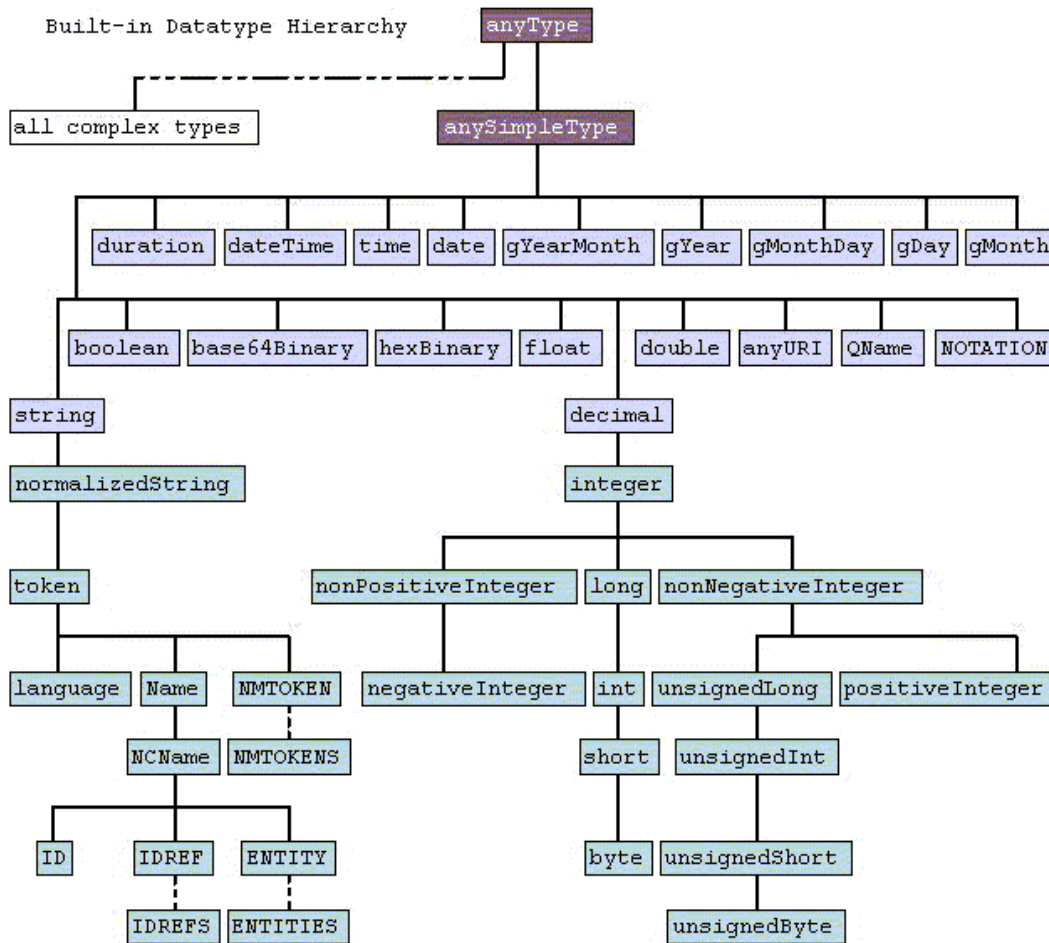


Figure 4: W3C built-in XML Schema data types

[Do we support all/some of [W3C built-in datatypes](#) or choose TCL/other? Per discussion 4/16/09, as XML libraries support all types defined in the standard and most are trivial derivatives of basic types, limiting TesLA to some subset or deviating from the XML standard will likely create more issues than are justified by any reduction in effort. In addition, we should endeavor to leverage as many existing, accepted standards as possible instead of creating new ones.]

## Pre-defined Enumerations

Pre-defined values and/or lookup tables (to correlate text to values) are used in several locations in DCA files. Where the values are part of the DCA schema, they are specified within the relevant XSD files as enumerations within the element or attribute definition as illustrated below. In the cases where the values cannot be pre-defined within the DCA schema (such as the XML used to pass parameters or responses between the DCA provider and consumer), or where additional information is needed, the enumerations are defined in the XML document at

<http://www.teslaalliance.org/standards/dca/teslaDCAEnums.xml>, which is incorporated into this specification by reference. [TODO: REVIEW/FINALIZE ENUMS WITH COMMITTEE]

```

<xs:attribute name="supportClass" default="GA">
  <xs:annotation>
    <xs:documentation>Commands may be included that are in various stages of
development - and may never be fully supported. This could be alpha, beta, custom,
customer-specific, internal, deprecated or GA (for example). Consumers should ONLY
use GA commands unless otherwise arranged with Publisher and/or customer.</
xs:documentation>
  </xs:annotation>
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="alpha"/>
      <xs:enumeration value="beta"/>
      <xs:enumeration value="GA"/>
      <xs:enumeration value="deprecated"/>
      <xs:enumeration value="internal"/>
      <xs:enumeration value="custom"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>

```

Figure 5: enumeration example - supportClass attribute

## Use of URIs, Base Path for Relative URIs

URIs are used in several locations in the TMD and TCD files to refer to other files that may be on local file systems or on remote systems and accessed via any method (scheme) supported within URIs as described in the [W3C Recommendation](#) and in [RFC2396](#). Where relative URIs are specified, the location of the DCA file from which the URI was read should be considered the default Base Path (URL) as described in [RFC1808](#). A Base Path MAY also be specified using the `dcaBasePath` attribute of the `TesLAModuleDefinition` element in the TMD file.

DCA 1.0 requires that Consumers MUST support file URIs. All other URI types are optional and may not be supported, so DCA Providers should be very careful as Consumers may not be able to resolve non-file-type URIs.

[should basePath be allowed in each Command? Other locations? FOR NOW ONLY AT MODULE LEVEL]

## Use of XML Includes

Many parameters, responses or other elements within TCD files may be the same across multiple commands. In these cases, the use of XML Inclusions is suggested per the [W3C Recommendation](#) to reduce redundant typing and improve readability ([fragment inclusion example](#)).

## DCA Help and Icon Files

DCAs MAY provide URIs to Help and Icon files at multiple context-sensitive locations in the TMD and TCD files. These MAY be supplied by Providers and used by Consumers for improved usability and presentation, but their use is not mandatory. Help files SHOULD be in HTML format and Consumers SHOULD be capable of displaying these files, although other formats MAY be used. Icon files SHOULD be in ICO or PNG format no larger than 32x32 pixels and Consumers SHOULD be capable of displaying these files, although other formats MAY be used.

## Module Types

The following Types of DCA Modules are defined for the purpose of organizing Modules (Devices) and facilitating a minimum set of common commands. It also provides a means for Consumers to organize and present functionality. The common command set is not designed to cover all functionality, but only the most basic common capabilities. It is anticipated that more sophisticated commands will be vendor-specific. Third parties may create their own DCAs “on top

of” vendor-provided DCAs or native APIs to their own command set(s) if desired. All TesLA Modules MUST be one of the Types defined in <http://www.teslaalliance.org/standards/dca/teslaDCAEnums.xml>:

dca:moduleTypes		
List of TesLA pre-defined DCA Module Types		
dca:moduleType (12)		
name	infoURI	description
1 PacketGeneratorAnalyzer	<a href="http://www.teslaalliance.org/standards/dca/PacketGenerator">http://www.teslaalliance.org/standards/dca/PacketGenerator</a>	A hardware and/or software device/program that can be configured to transmit, receive and analyze frames/packets over a network interface without maintaining application state or user context.
2 UserSimulator	<a href="http://www.teslaalliance.org/standards/dca/UserSimulator">http://www.teslaalliance.org/standards/dca/UserSimulator</a>	A hardware and/or software device/program that can be configured to simulate and analyze user interaction with an application over a network interface, maintaining application state and user context.
3 NetworkEmulator	<a href="http://www.teslaalliance.org/standards/dca/NetworkEmulator">http://www.teslaalliance.org/standards/dca/NetworkEmulator</a>	A hardware and/or software device/program that can be configured to emulate a network (i.e. inject impairments, control bandwidth, etc.)
4 Layer1Switch	<a href="http://www.teslaalliance.org/standards/dca/Layer1Switch">http://www.teslaalliance.org/standards/dca/Layer1Switch</a>	
5 Layer2Switch	<a href="http://www.teslaalliance.org/standards/dca/Layer2Switch">http://www.teslaalliance.org/standards/dca/Layer2Switch</a>	
6 Router	<a href="http://www.teslaalliance.org/standards/dca/Router">http://www.teslaalliance.org/standards/dca/Router</a>	
7 TAT	<a href="http://www.teslaalliance.org/standards/dca/TAT">http://www.teslaalliance.org/standards/dca/TAT</a>	
8 TRS	<a href="http://www.teslaalliance.org/standards/dca/TRS">http://www.teslaalliance.org/standards/dca/TRS</a>	
9 DataAnalyzer	<a href="http://www.teslaalliance.org/standards/dca/DataAnalyzer">http://www.teslaalliance.org/standards/dca/DataAnalyzer</a>	
10 DataMonitor	<a href="http://www.teslaalliance.org/standards/dca/DataMonitor">http://www.teslaalliance.org/standards/dca/DataMonitor</a>	
11 ReportGenerator	<a href="http://www.teslaalliance.org/standards/dca/ReportGenerator">http://www.teslaalliance.org/standards/dca/ReportGenerator</a>	
12 Other	<a href="http://www.teslaalliance.org/standards/dca/Other">http://www.teslaalliance.org/standards/dca/Other</a>	DCAs can be created for Devices that have not been defined within TesLA, but only a small set of universal standardized commands will be available. Usage will depend on some vendor-specific knowledge

Figure 6: Module Types (per <http://www.teslaalliance.org/standards/dca/teslaDCAEnums.xml>)

[TBC: REVIEW/FINALIZE LIST OF MODULE TYPES WITH COMMITTEE]

## What Should Be Included in a DCA

The DCA will generally expose a subset of the capabilities of a Device. While it is impractical (and perhaps undesirable) to expose *all* of the capabilities of a Device at a granular level (i.e. a one-to-one mapping to the native vendor API), adequate high level commands should be exposed to allow useful multi-vendor automation scripts to be created. TesLA defines a minimal mandatory common command set (described in the TCD section), but it is expected that Providers will support additional commands as determined by common customer use cases. A DCA that does not expose adequate functionality in a way that makes their Device useful to third parties would not be following the intent of the TesLA standards. TesLA does not enforce usability but expects commercial considerations to drive usability and usefulness. While a single DCA could be created that connects to and controls multiple Devices, this is strongly discouraged. Each DCA SHOULD control a single Device. It should be noted that in this context, a “Device” is ANY Module that exposes functionality to other TesLA Modules.

TesLA-compatible Test Authoring Tools (TATs) will be able to create automated Tests that coordinate the activities of multiple Modules from multiple vendors. They will not, in general, be capable of creating intricate device-specific configurations, though. It is more likely that vendor-specific tools will be used to create these configurations, and that these configurations will be invoked by the TAT.

## Customizing or Extending Command Sets

Third parties (other than the Device manufacturer) may create TesLA Command Definition files (referencing vendors' native APIs via a "wrapper function" according to the Binding Specification) and place them at any of the TMD-defined Command File URIs. Thus, they can extend the command set without modifying any files installed by the Provider.

```
<!-- any number of locations for this DCA's TCD files can be listed /-->
<commandFileURI>file://./commands</commandFileURI>
<commandFileURI>file://server/share/commands</commandFileURI>
```

Figure 7: TMD pre-defined TCD URIs

In order to direct the Consumer to process TCD files that are NOT in locations specified by the TMD, a TCD file must be placed at one of the TMD-defined URIs that includes any number of includeCommandURI elements as shown in the example below. This is convenient, for example, in a networked environment where many workstations need to load the same customized TCD files. Third parties SHOULD NOT modify the TMD or Vendor-provided Command Files, but instead should create their own TCD files and optionally insert includeCommandURI elements within those files.

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- This is an example of a TCD file that, if placed at any URI specified in the
TesLAModules environment variable, will direct a Consumer to discover
additional commands at the includeCommandURI location(s) below -->

<tdca:TesLACommandDefinition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.TesLAAlliance.org/stds/dca TesLACommandDefinition.xsd"
xmlns:tdca="http://www.TesLAAlliance.org/stds/dca" >

  <tdca:includeCommandURI>file://myServer/myTCDFiles</tdca:includeCommandURI>
</tdca:TesLACommandDefinition>
```

Figure 8: Example of a TCD file that can be placed at any TMD-specified URI to include another location

## Vendor (proprietary) Extensions

Many elements in this specification contain a child element named `vendorExtensions`. This optional element may contain any number of attributes or elements of any type. This provides:

- A means of maintaining proprietary support for features that TesLA mechanisms are not well-suited for
- A means of exposing functionality per vendor-specific interfaces that go beyond what is mandated by the TesLA standards
- A migration path toward TesLA support (by allowing legacy elements from proprietary implementations)

Consumer applications and Modules should ignore vendorExtensions EXCEPT:

- If the Consumer vendor is also the Provider vendor
- If there is prior agreement between the supplier of the Consumer and the supplier of the Provider (for example, if custom modifications were made to a DCA for a specific customers)

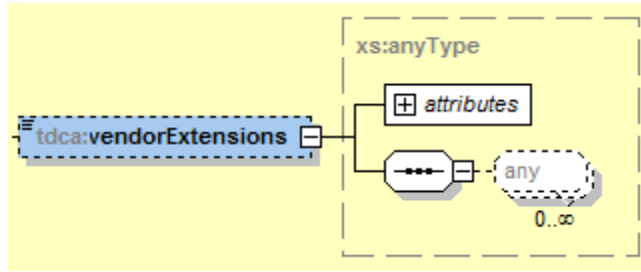


Figure 9: Consumers should ignore vendorExtensions unless you have agreed to usage with the Provider

## The TesLA Module Definition (TMD) File

The TMD is an XML file complying with the schema defined in

<http://www.teslaalliance.org/standards/dca/TesLAModuleDefinition.xsd> and whose individual elements and attributes are defined in detail in <http://www.teslaalliance.org/standards/dca/AutoDocs/TMDAutoDoc.html>. It provides high level information *about* the Module and the information required to discover the details of Module capabilities and operation. For extensibility and maintainability, command definitions are not embedded in the TesLA Module Definition but are in Command Files at the URIs listed in the TMD. The TMD SHOULD NOT be modified by users; modification of the TMD is not needed to extend or customize Device capabilities.

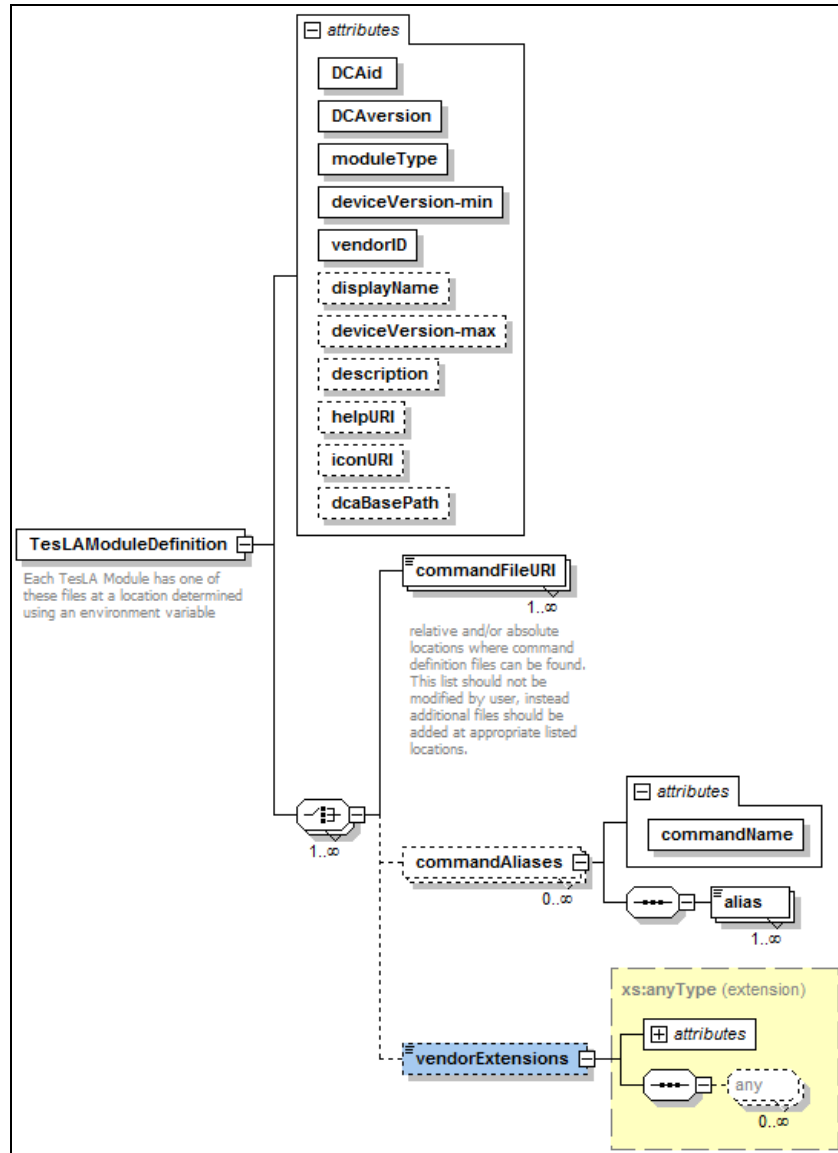


Figure 10: TMD Schema (<http://www.teslaalliance.org/standards/dca/TesLAModuleDefinition.xsd>)

```

<?xml version="1.0" encoding="UTF-8"?>
<TesLAModuleDefinition
  xsi:noNamespaceSchemaLocation="TesLAModuleDefinition.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:dca="http://www.teslaalliance.org/standards/dca"
  displayName="IxExplorer DCA"
  description="TesLA DCA Driver for Ixia IxExplorer"
  moduleType="PacketGeneratorAnalyzer"
  vendorID="Ixia"
  DCAid="http://www.ixiacom.com/tesla/dca/ixexplorer"
  DCAversion="1.0"
  deviceVersion-min="3.2"
  iconURI="file://IxExplorer.png"
  helpURI="file://IxExplorerDCAhelp.html" >

  <!-- any number of locations for this DCA's TCD files can be listed -->
  <commandFileURI>file://./commands</commandFileURI>
  <commandFileURI>file://server/share/commands</commandFileURI>

  <!-- any number of command aliases for core commands that might be
  used during initialization may be listed here.  commandName is a
  command name from a TCD file -->
  <commandAliases commandName="Open">
    <alias>startSession</alias>
    <alias>start</alias>
    <alias>connect</alias>
  </commandAliases>

  <vendorExtensions>
    <!-- any attributes and/or elements a vendor wants can go here for
    proprietary use -->
  </vendorExtensions>

  <commandAliases commandName="Close">
    <alias>stopSession</alias>
    <alias>stop</alias>
    <alias>disconnect</alias>
  </commandAliases>
</TesLAModuleDefinition>

```

Figure 11: TMD Example (<http://www.teslaalliance.org/standards/dca/TMDSample.xml>)

## Versioning & Compatibility

There are two relevant versions for each DCA:

1. The version of the DCA
2. The version range of the Device it supports

A new DCA version may be released without a new Device version and vice versa. New commands or parameters can be exposed with no change in the underlying functionality of the supported Device. Similarly, a Device version may be changed, but an earlier DCA can still be used without exposing new functionality.

While it might be possible for a Consumer to load multiple versions of the same DCA within a single Test, it would be preferable to only use the latest version available. TesLA mandates that backward compatibility MUST be maintained by all TesLA-compliant Devices and DCAs. So, for example:

- A new DCA release MUST run Tests written with prior versions of the same DCA
- A new release of a Device MUST run Tests written with prior versions of supported DCAs

In practice, what this implies is that if the number or meaning of mandatory parameters of a command changes, either the calling syntax must handle this or a new command should be defined. It also implies that the underlying device control libraries must continue to support older DCA commands indefinitely.

In the rare event that breaking compatibility with a previously defined command is unavoidable, the “deprecated” supportClass MUST be used for at least one release and a utility and/or procedure should be provided to translate or adapt existing Tests to the replacement Command.

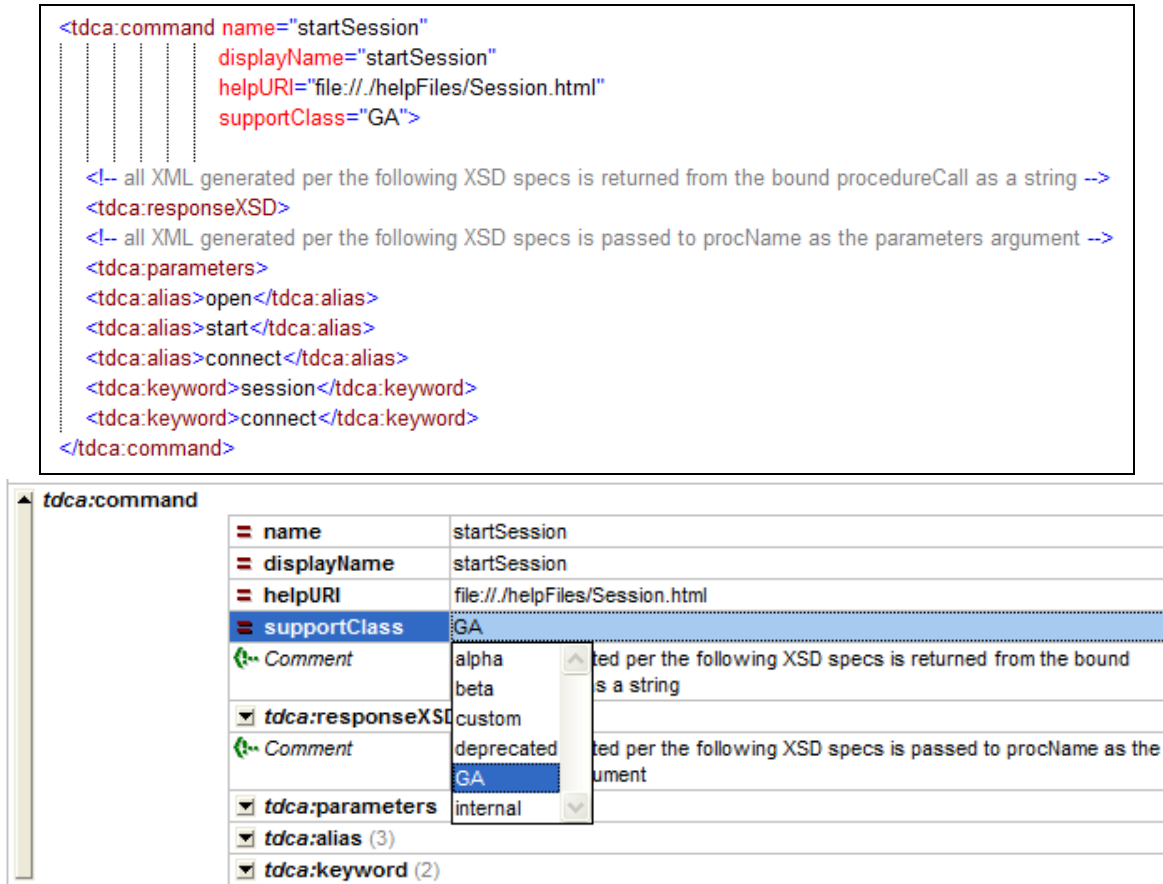


Figure 12: Command supportClass attribute and enumerated values in an XML editor

## Design- and Run-time Version Checking

DCA and supported Device version information is provided in the TMD files. This information is important at run-time, as a Test may be run in a different environment that it was created and tested in. The DCA MUST verify that the Device or Server to which it connects is between the deviceVersion-min and deviceVersion-max (if specified in the TMD).

```

<TesLAModuleDefinition
  xsi:noNamespaceSchemaLocation="TesLAModuleDefinition.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:dca="http://www.teslaalliance.org/standards/dca"
  displayName="IxExplorer DCA"
  description="TesLA DCA Driver for Ixia IxExplorer"
  moduleType="PacketGeneratorAnalyzer"
  vendorID="Ixia"
  DCAid="http://www.ixiacom.com/tesla/dca/ixexplorer"
  DCAversion="1.0"
  deviceVersion-min="3.2"
  iconURI="file://IxExplorer.png"
  helpURI="file://IxExplorerDCAhelp.html" >

```

Figure 13: Device Version Compatibility specification in the TMD file

This also implies the following recommendations to other TesLA committees:

- TAT: standardized Test metadata (either in a header of “companion file”) SHOULD contain the DCA ID and Device version information with which it was created.
- TRS: DCA and Device version information SHOULD be considered when resolving an Abstract Topology to a Concrete Topology.

TesLA versions are specified as a[[[.b].c].d] where a, b, c and d are any positive integer. Version numbers are compared from left-to-right, with the leftmost integer being the most significant and the rightmost integer being least significant. Where minimum and maximum supported versions are specified, they should be interpreted according to the version scheme above.

## Command Aliases

Due to historical implementations, there are certain common commands that are expected to have a certain name. In order to support these applications, the TMD allows the creation of any number of “Command Aliases”. Consumers MUST treat all references to a Command Aliases for a given command as equivalent. These are provided in the TMD (and not the TCD) as certain Consumers may need to execute one of these commands in order to complete the loading and validation process.

```

<!-- any number of command aliases for core commands that might be
      used during initialization may be listed here.  commandName is a
      command name from a TCD file /-->
<commandAliases commandName="Open">
  <alias>startSession</alias>
  <alias>start</alias>
  <alias>connect</alias>
</commandAliases>
<commandAliases commandName="Close">
  <alias>stopSession</alias>
  <alias>stop</alias>
  <alias>disconnect</alias>
</commandAliases>

```

Figure 14: Command Alias definitions in a TMD file

## TesLA Command Definition (TCD) Files

The TesLA Command Definition (TCD) File is an XML file complying with the schema at <http://www.teslaalliance.org/standards/dca/TesLACommandDefinition.xsd> and whose individual elements and attributes are defined in detail in <http://www.teslaalliance.org/standards/dca/AutoDocs/TCDAutoDoc.html>. All TesLA Command Definitions at any of the URIs listed in the Module Definition MUST be processed, but it is up to the Consumer to determine which Commands are exposed or invoked. Each TesLA Command Definition (TCD) file defines one or more commands including naming, parameters, help files and binding/execution information (see details below).

[Not all commands will be supported at all times by all Device configurations. How do we deal with which of the commands are “enabled” for a given configuration? Do we make provisions for “dynamic commands” that the device creates/configures at run-time?] *Decision: 1.0 handles same as if you were writing a script today – the script expects certain commands to be available and must deal with errors when they occur.*

### ***The Session***

Commands are executed within the context of a long-lived communication channel and interaction with a Device. When creating a Test, at least one “Session” must be opened with each Device being controlled. All Commands executed are associated with this single Session. Sessions are created with the Open command (which may have aliases assigned in the TMD file), which MUST be the first command executed with a Device. When all Commands are completed for a Device within a Test, the Close Command should be used to terminate the Session and release the Device Resources for use with another Test.

[How does the TAT or AXE know if a session goes down unexpectedly? Is there any kind of “handle” or is the context just the shell that is being used?]

### ***Command File Structure***

Command files MUST adhere to the structure defined in the [TesLACommandDefinition.xsd](http://www.teslaalliance.org/standards/dca/TesLACommandDefinition.xsd). The basic structure is illustrated below. Detailed documentation of each Element and Attribute may be found at <http://www.teslaalliance.org/standards/dca/AutoDocs/TCDAutoDoc.html>. A sample TCD file may be found at <http://www.teslaalliance.org/standards/dca/TCDSample.xml>.

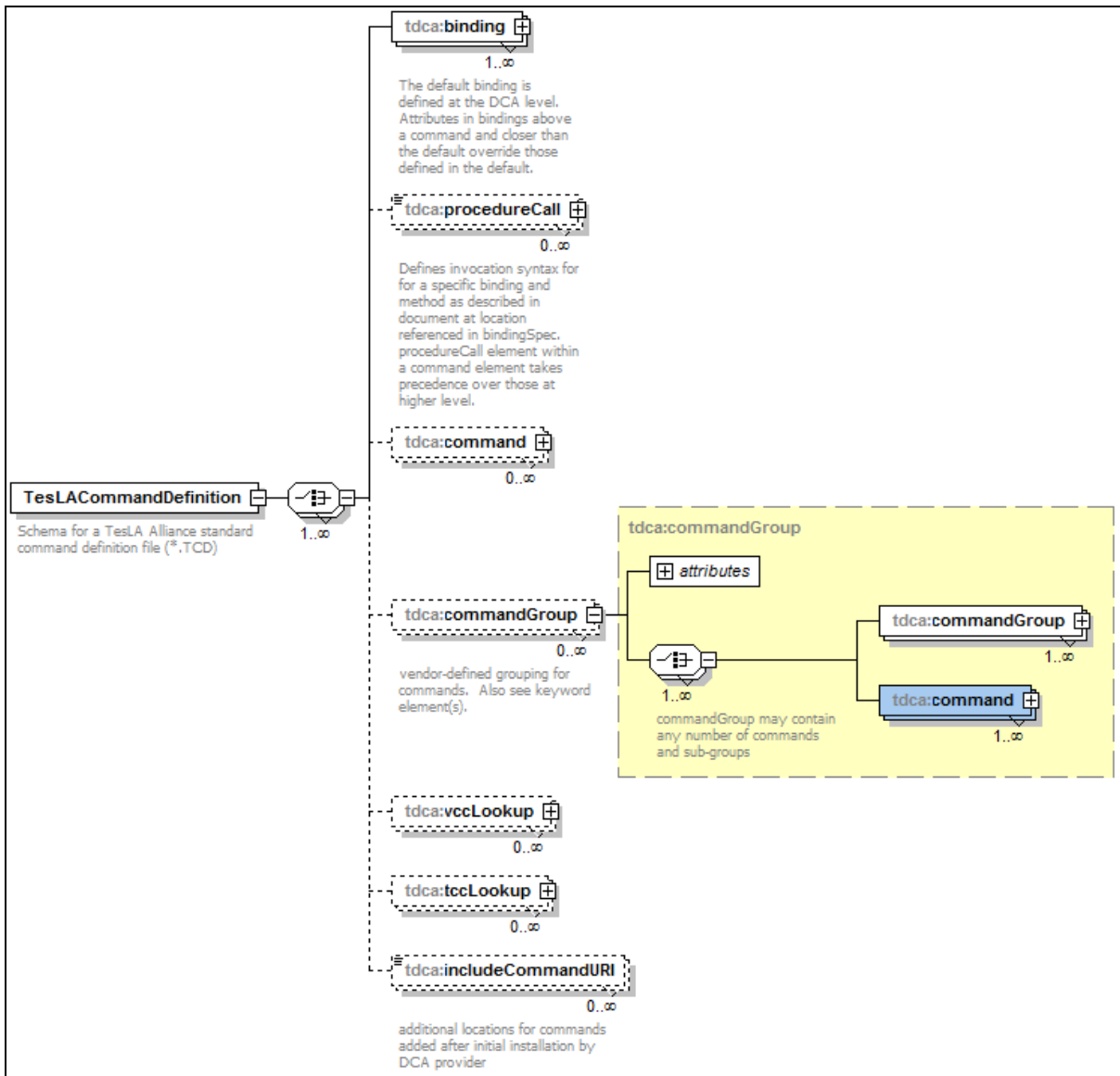


Figure 15: TCD High Level Schema

Note the following above (each is described in more detail in the sections below):

- commands may be standalone or put into commandGroups
- commandGroups may be nested arbitrarily deep
- all supported bindings are defined as children to the root element and referenced within procedureCall elements
- a default procedureCall element may be defined as a child to the root element; this is only used if there is a single procedureCall per binding, otherwise there should be a procedureCall per command
- a command may have multiple child procedureCall elements for different languages or methods. It is up to the Consumer to identify and select the preferred supported binding.

### Command Classification & Mandatory Commands

Each command implemented in a DCA can be classified as:

- DCA-Common: all Devices of any Type MUST support these Commands with their corresponding Parameters as described in this specification.
- Device-Type-Common: all Devices of the defined Type MUST support these Commands with their corresponding Parameters as described in this specification.
- Device-Specific-Supported: these optional Commands are defined by the Provider and are intended for external use. They will have a `supportClass` attribute of “GA” in the command element.
- Qualified Support: commands may be included at the discretion of the Provider that are for internal use, for specific customers or are in pre-release form. The `supportClass` attribute of the command element is used to describe the Support level.

[TBC: TCD files containing DCA-Common and Device-Type-Common command definitions for inclusion into appropriate DCAs]

Each DCA MUST have a command (or an alias) that corresponds to the exact name of the commands listed in this table.

Command	Classification	Description
Open	DCA-Common	Creates a uniquely named Device Session, which is used for associating commands and maintaining state.
Close	DCA-Common	Terminates a Device Session.
Start	DCA-Common	Initiates pending activities associated with resources/handles provided
Stop	DCA-Common	Initiates pending activities associated with resources/handles provided
RunCommand	DCA-Common	Pass through (opaque) command/script string for interpretation by Device
LoadConfiguration	Device-Type-Common: PacketGenerator UserEmulator Others that support external configuration files	Loads an application-specific configuration file passed by URI.
GetDeviceInformation	DCA-Common	Provides version, type, ID and other information about a Device. [TBC: LIST OF MANDATORY AND OPTIONAL INFORMATION/STATUS ITEMS]
TransmitStart	Device-Type-Common: PacketGenerator UserEmulator	
TransmitStop	Device-Type-Common: PacketGenerator UserEmulator	

Command	Classification	Description
GetResources	DCA-Common	[this needs to be specified by TRS Committee]
ListGeneratedFiles	DCA-Common	Provides list of result, log and other files generated by session associated with the Open command.
OTHER DEVICE-TYPE-SPECIFIC COMMANDS	Device-Type-Common:	[NEED INPUT FROM OTHER DEVICE AND TAT VENDORS]
ALL OTHER COMMANDS NOT DEFINED ABOVE	Device-Specific-Supported:	

## ***Bindings & Procedure Calls***

Each binding will have its own specification to allow additional bindings without modifying this specification. Each Binding Specification will define the required XML elements and attributes required in the `bindingExtensions` and `procedureCall` elements. The paragraphs below describe how bindings are used but are not intended to fully define the binding.

A Binding and Procedure Call together describe how a command is invoked from a specific programming environment. This environment includes either the programming or scripting language used and the method used from in that language. Some possible bindings include:

- TCL Package & Local Procedure Call
- TCL Remote Procedure Call
- XMLRPC
- SOAP
- .NET

TesLA DCA Commands are able to support multiple Bindings and multiple Methods per Binding. At least one Binding element **MUST** be included under the `TesLACommandDefinition` Element. The `procedureCall` needed to implement a Command for each supported Binding **MUST** be provided. In the simplest case, where all Commands within a TCD file use the same `binding` and `procedureCall`, the single Binding and `procedureCall` may be defined as child elements of the `TesLACommandDefinition` element. Where there is a single Binding, but multiple `procedureCalls`, a `procedureCall` element **MUST** be included under each Command that does not use the default defined within the `TesLACommandDefinition`. Where there are multiple Bindings, every Command **MUST** have a `procedureCall` Element that references each supported `binding`.

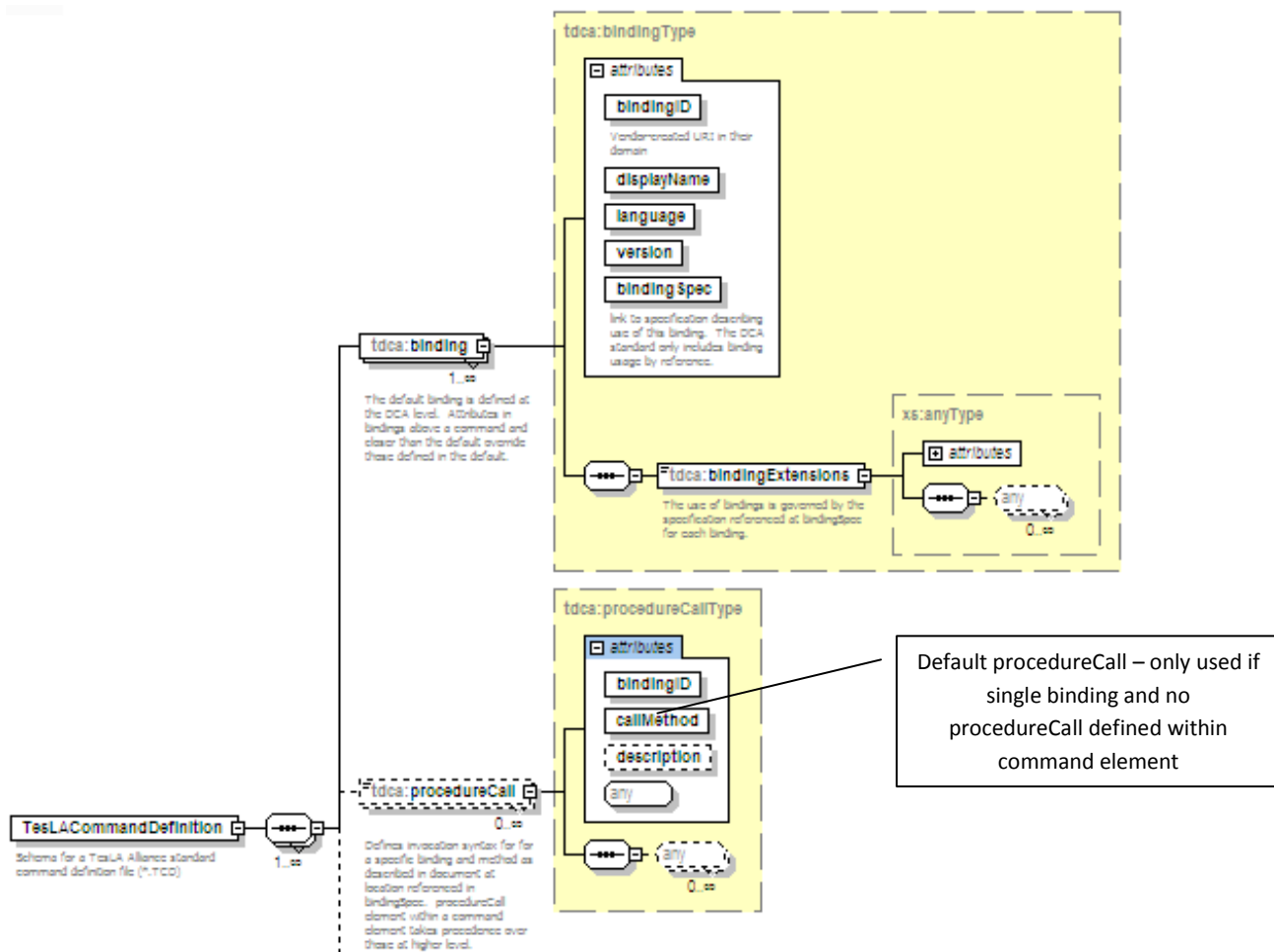


Figure 16: bindings and default procedureCall for each binding

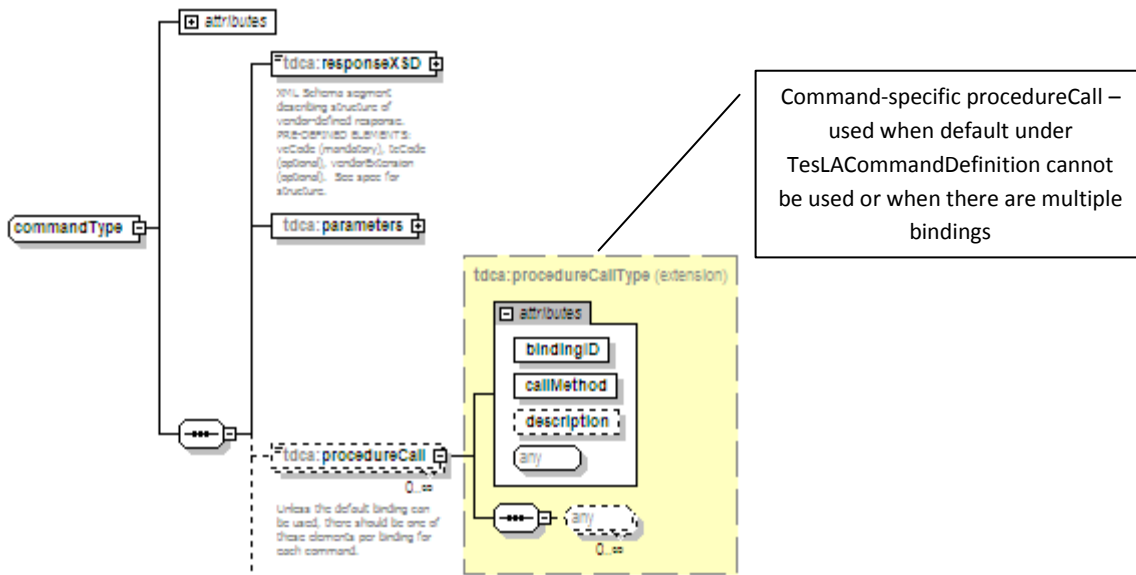


Figure 17: binding- or command-specific procedureCall

All supported DCA commands MUST provide a TCL binding that complies with the **XXX specification (examples in this document prior to the release of the XXX specification contain assumptions about TCL Binding implementation)**. DCA Commands MAY support additional bindings (such as .NET or XMLRPC). As the methods used to invoke commands vary significantly from one language and remote access protocol to another, this specification does not attempt to pre-define

a method of describing all possible bindings. Instead, ANY XML Schema elements and attributes may be added to the binding and procedureCall elements and their usage is described in an additional referenced specification.

[should operating system/platforms supported by a binding be part of TCD file outside of XML defined by the binding spec?]

## TCL Binding Notes (to author of TCL Binding Spec)

- Describe pre-requisites (TCL client installation)
- Fully define (using XSD) the content of the binding and procedureCall elements within the TesLACommandDefinition
- Describe shell version requirements
- Describe passing of parameters (as XML string) and response (as XML string return value)
- Describe calling convention
  - Use of XPATH (relative to command element) for specifying parameters for procedure call
- Platform-specific issues
- Sample suggesting how this might work and use of XPATH notation to construct API call:

```
<!-- BINDING IS DEFINED IN THE TCL BINDING SPEC (THIS IS PRE-SPEC)
| procedureCall procName is name of TCL procedure to call and
| paramSequence describes procedure parameters using XPATH syntax
| procedureCall closest to command function should be used; procedureCall outside command block is default. -->

<tdca:binding bindingID="http://www.teslaalliance.org/standards/dca/bindings/tcl/"
| version="1.0.2"
| language="TCL"
| displayName="TCL Package"
| bindingSpec="http://www.teslaalliance.org/standards/dca/bindings/tcl/DCATclBinding.html">
| <tdca:bindingExtensions xmlns:btcl="http://www.teslaalliance.org/standards/dca/bindings/tcl/">
|   <btcl:tclBinding sourceFile="file:///CommandProcedures/ixexplorerDCAProcs.tcl" shellFile="wish80" minShellVersion="8.0">
|     </btcl:tclBinding>
|   </tdca:bindingExtensions>
| </tdca:binding>

<tdca:procedureCall bindingID="http://www.TesLAAlliance.org/stds/bindings/tcl/"
| callMethod="tclProc"
| procName=":tesLA::XMLAPI"
| paramSequence="@name parameters/*">
| </tdca:procedureCall>
```

## Command Groups and Command Definitions

Each exposed command is described in a `command` element, the structure of which is illustrated in Figure 18: `command` element. As in the TCD, aliases are supported. Consumers MUST treat aliases as equivalent to the `name` attribute.

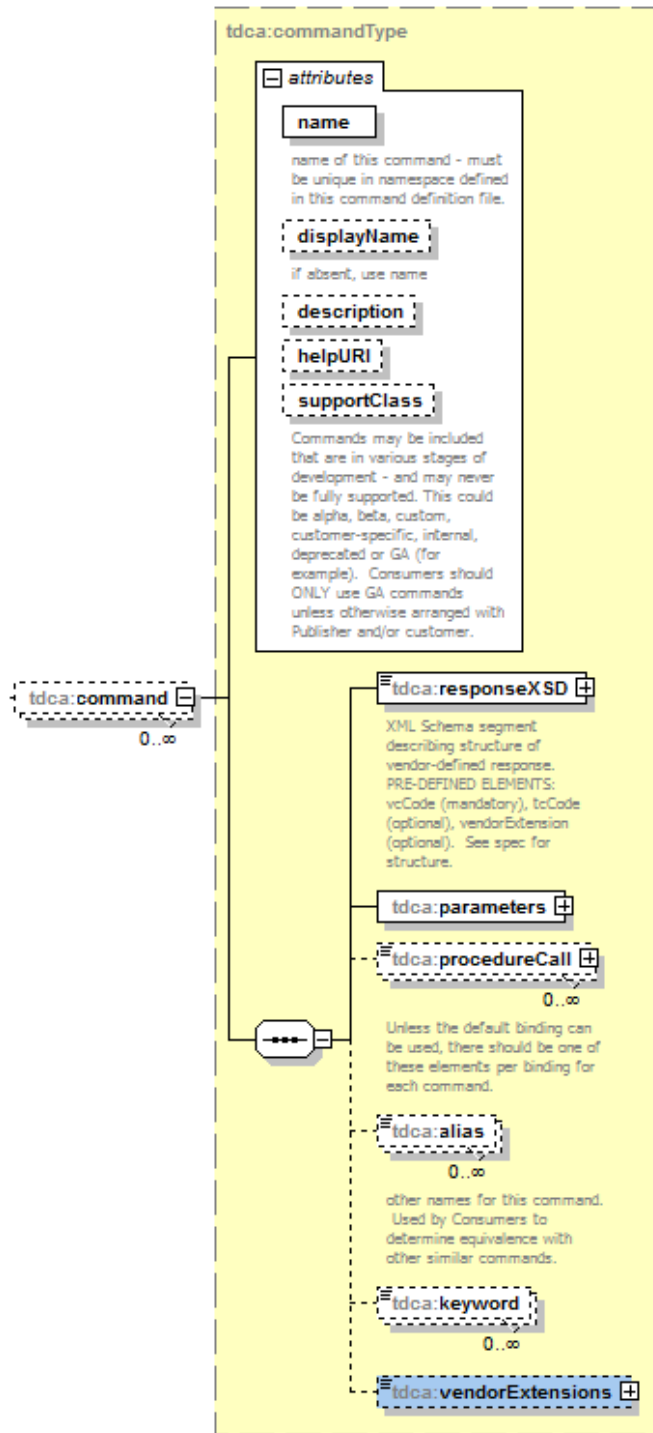


Figure 18: command element schema

command elements may be logically grouped into commandGroups, the structure of which is illustrated in Figure 19: commandGroup. commandGroups may contain both commands and additional commandGroups, which may be nested arbitrarily deep.

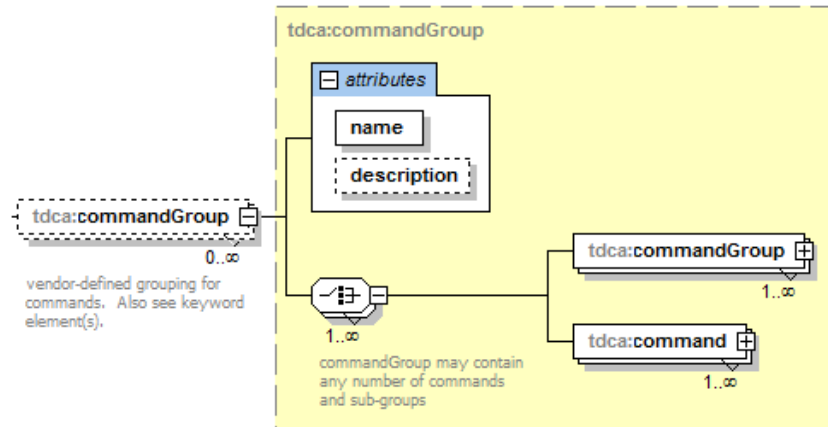


Figure 19: commandGroup element schema

In addition to `commandGroups`, the keyword element is provided to “tag” commands to allow an arbitrary number of alternative groupings, filters or searches.

```
<tdca:command name="startSession" displayName="startSession" helpURI="file:///helpFiles/Session.html" supportClass="GA">
  <tdca:keyword>session</tdca:keyword>
  <tdca:keyword>connect</tdca:keyword>
</tdca:command>
```

Figure 20: Example of command keywords (tags)

## Command Parameters & Responses

The basic DCA model is based on a request-response model. The Consumer makes a request to the Provider. The Provider performs the request and produces a response. The request identifies a command and provides a set of parameters for that command.

Both the parameters and responses (return parameters) for commands can become quite complex. Instead of creating TesLA-defined elements and attributes to describe them, TesLA will use XML Schema itself to describe parameters (within the `paramXSD` element) and responses (within the `responseXSD` element). Accordingly, both the `parameters` and `response` elements have child elements to contain XSD (XML Schema) text that contains any valid elements and attributes that comply with XML Schema. It is up to the Consumer to collect or create and format ALL of the XML described by ALL of the `parameter XSD` segments within the `parameters` element. The `displayName` and `helpURI` attributes for each `parameter` are for Consumer consumption and serve no purpose to the API.

The parameters are passed as a small XML document that conforms to the XSD specified in the TCD element for that command. The response is another XML document that returns information from the Provider to the Consumer that is appropriate to that command -- and that response must conform to the XSD specified in the TCD describing responses for that command. This all works well -- and provides a lot of latitude to the DCA designer to handle complex and well-defined requests and responses. The more specific the XSD is for the request and response, the better the Consumer can do in presenting this information to the User.

As many parameters are common to many commands, it will often be useful to include parameter-specific XSD files referenced using an XML Inclusion to improve readability and reduce repetition. Note that parameters and responses are the same for all bindings under the assumption that the API implementations can adapt to the calling conventions required by the binding.

`vendorExtensions` are provided for use as described in the Vendor Extensions section above.

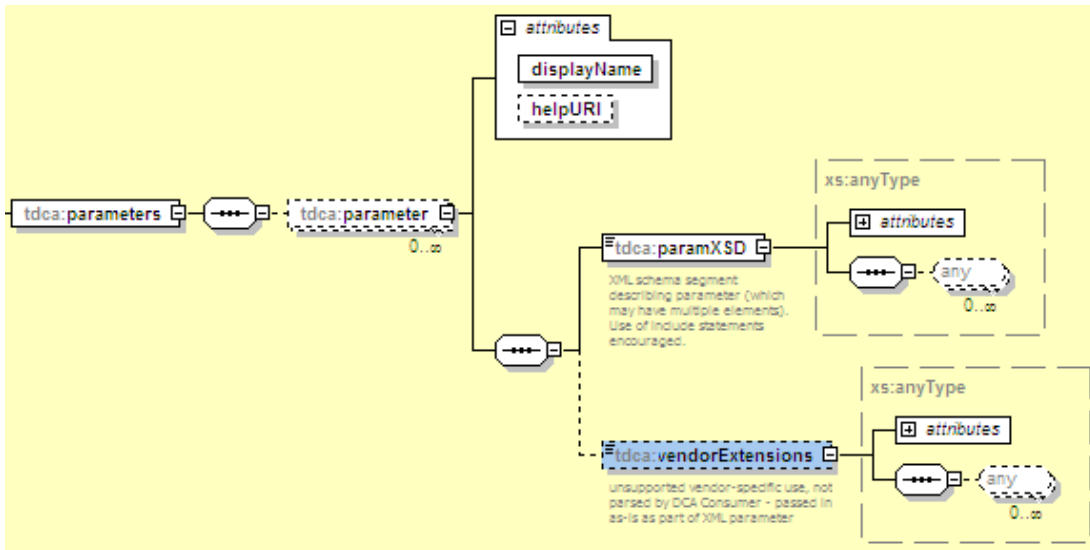


Figure 21: paramXSD contains XML Schema describing each parameter

<!-- all XML generated per the following XSD specs is passed to procName as the parameters argument -->

```

<tdca:parameters>
  <tdca:parameter displayName="Chassis" helpURI="file:///HelpFiles/session.html">
    <tdca:paramXSD>
      <xs:element name="chassis">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:pattern value="[0-2][0-5][0-5].[0-2][0-5][0-5].[0-2][0-5][0-5].[0-2][0-5][0-5]"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </tdca:paramXSD>
  </tdca:parameter>

  <tdca:parameter displayName="Ports" helpURI="file:///HelpFiles/session.html">
    <tdca:paramXSD>
      <xs:element name="ports">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:pattern value="\{*(\{*\d*\d*\d*\})+*\}/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </tdca:paramXSD>
  </tdca:parameter>

  <tdca:parameter displayName="Ownership Mode" helpURI="file:///HelpFiles/session.html">
    <tdca:paramXSD>
      <xs:element name="OwnershipMode" default="force">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="force"/>
            <xs:enumeration value="noForce"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </tdca:paramXSD>
  </tdca:parameter>
</tdca:parameters>

```

Figure 22: example of a parameters block with three parameters defined using embedded XSD

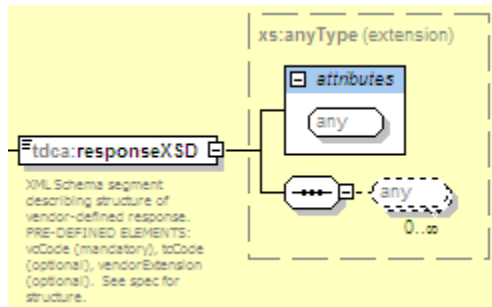


Figure 23: responseXSD contains XML Schema describing the API response

```

<!-- all XML generated per the following XSD specs is returned from the bound procedureCall as a string -->
<tdca:responseXSD>
  <!-- XML Schema segment describing response provided by command (return value) -->
  <xs:element name="tResponse">
    <!-- MANDATORY: Tesla Response MUST be in a tResponse element -->
    <xs:complexType>
      <xs:sequence>
        <xs:element name="logFile" minOccurs="0" maxOccurs="unbounded">
          <!-- RECOMMENDED: list of log files generated with optional format label -->
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:anyURI">
                <xs:attribute name="fileFormat" type="xs:string" use="optional"/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="vcCode" type="xs:int" use="required"/>
      <!-- MANDATORY: MUST have a vendor completion code -->
      <xs:attribute name="tcCode" type="xs:int" use="optional"/>
      <!-- RECOMMENDED: Tesla completion code per http://www.teslaalliance.org/standards/dca/teslaDCAEnums.txt -->
      <xs:attribute name="vcMsg" type="xs:string" use="optional"/>
      <!-- RECOMMENDED: vendor-defined descriptive completion text -->
    </xs:complexType>
  </xs:element>
</tdca:responseXSD>

```

Figure 24: example of responseXSD showing pre-defined attributes

Note that in the example above, there are three attributes and an element that are pre-defined and are not specific to any API. These are specified here but there is no mechanism for including them in TeslaCommandDefinition.xsd (the TCD schema), since there is no way to specify that the XML file being described includes XML schema elements. These are:

- logFile: an element SHOULD be created per log or result file that is generated by the API call.
- vcCode: a Vendor Completion Code MUST be provided with each response. The vcLookup elements MAY be used to provide descriptions for the codes.
- vcMsg: a Vendor Completion Message SHOULD be provided with each response to provide more details of execution, errors and completion.
- tcCode: a Tesla Completion Code MAY be provided with each response. These will have pre-defined meanings as described in <http://www.teslaalliance.org/standards/dca/teslaDCAEnums.xml> and optionally within the TCD file using tcLookup elements.

[Does spec need to address the use of collected elements or attributes or from a response within a Test?] *The assignment of collected parameters or responses to variables for re-use is a feature that can be provided by the DCA Consumer, not part of the DCA standard.*

[need to specify how root element is "generated" by Consumer for XSD within 'parameters' and 'response?']

## ***Interacting with Long-Running Commands***

Occasionally a command may run for a long period of time. In these cases, it is desirable to be able to interact with the running command. For example:

- requesting command status
- requesting command progress
- aborting the command

This interaction model is not compatible with the strict request-response that is common in most native Device APIs. The TesLA DCA specification will be updated in the future to recommend a method of implementing these asynchronous operations. As this will describe what, in most cases, is new functionality that is not available in the underlying native Device APIs, it will not be mandatory.

[Alternative methods are still under consideration. One proposal can be found at <http://teslaalliance.pbworks.com/DCA> ("Monitor object concept").]

## ***Code Lookups***

The TCD file MAY provide text values to be associated with TesLA and Vendor Completion Codes as shown in the example below.

```
<!-- Vendor Completion Codes -->
<tdca:vccLookup code="0" msg="Fail"/>
<tdca:vccLookup code="1" msg="Success"/>

<!-- tesLA Completion Codes TBC in standard, this could be in all DCA files via XInclude -->
<tdca:tccLookup code="0" msg="Fail"/>
<tdca:tccLookup code="1" msg="Success"/>
```

**Figure 25: Completion Code Messages**

[TBC: Committee needs to define all tcc values/messages]