The Performance Data Collector for HP OpenVMS

TDC V2.2 Technical Overview

March 2006

Contents

Figures

Tables

Background

Insufficient OpenVMS Data Providers

- OpenVMS delivers a set of System Services to provide system metrics
- OpenVMS System Services provide lots of useful information, but
  - Don't cover all aspects of system performance;
  - Often provide data at too high a level;
  - May lag behind OpenVMS functional and performance enhancements;
  - Can be cumbersome to use.

Multiple OpenVMS Data Collectors

- Different projects have different data needs, often met by supplementing OpenVMS System Services by:
  - Using undocumented system APIs;
  - Directly accessing system data structures.
- Potential problems:
  - Changes to undocumented APIs
  - Doing it "right" (*e.g.,* synchronization, etc.)
  - Reliability and validity of data
  - Conflicts (*e.g.*, IOPERFORM)
  - Staying current
  - Drag on engineering resources

Data Consumers

- May require multiple data collectors for all needed metrics
- Forced to wait for data collectors to add new metrics
- "Live" *vs.* "Stored" data

The Performance Data Collector for HP OpenVMS (TDC V2)

- Provides "raw" data, much of which is not available through OpenVMS System Services
- Tracks OpenVMS enhancements
- Documented API
- Reduces potential problems associated with data collection
- Can be extended by data consumers
- Supports both "live" and "stored" data

What is TDC V2?

TDC V2 is…

- A provider of system configuration and performance data for use by other software applications.
- A "bulk" data provider, that allows programmers to request groups of performance-related system metrics.
- A tool to manage the collection of system performance data.
- A tool to manage the extraction of data from a data file.
- A component to be integrated into software applications.

TDC V2 is not…

- A "system service" that offers programmers flexibility to request individual performance metrics of interest
- A complete system performance analysis solution.
- A standalone application like MONITOR, T4, ECP, etc.

TDC V2 is targeted at…

- Developers of system performance-monitoring software.

TDC V2 is not targeted at…

- System managers.
- End users.

TDC V2 is best-suited to…

- Facilitate deployment and integration of system performance-analysis and performance-management software applications.

What are the principal TDC V2 Components?

- An “Engine” that:
    - Manages the collection of data and extraction of data from a file.
    - Coordinates the activities of Processor Modules that collect and/or process system data.
    - Is fully accessible through a supported Application Programming Interface (API).

- A set of “Processor Modules” that:
    - Perform data collection and/or extraction tasks, timing, file access, etc. using the Engine’s API.
    - Together, provide more than 1000 performance-related metrics.
    - Can be extended…
        - In the field
        - By ISVs, etc.
        - By customers

- An Application that uses the Engine’s API to:
    - Collect data and store it in a file;
    - Extract data from a file;
    - Convert data to various formats.

- A Software Developer’s Kit (SDK) that provides:
    - A specification of the API;
    - A description of metrics provided “out-of-the-box” by TDC;
    - A Programmer’s Guide and this technical overview;
    - Header files;
    - Code samples.

TDC V2 Architectural Highlights

The architecture is “open”

The API can be used to control all aspects of a data collection or extraction operation, thereby facilitating integration of the Engine and API-selectable PROCESSOR MODULEs into software applications.

The architecture is “extensible”

The real work of collecting and processing data is performed by individual PROCESSOR MODULEs under control of the TDC Engine. Processor Modules are loaded by the Engine at run time, under control of the API. Thirty-five PROCESSOR MODULEs are provided with TDC (see Table 1); the SDK allows third parties to develop and deploy additional PROCESSOR MODULEs to collaborate in collecting and processing data.

Data can be provided “live”

The architecture separates the production and storage of data records. Once data records have been produced, they can optionally be stored in a file for offline processing, but they can also be directed “live” to other software through appropriate Processor Modules.

Multiplatform support

- OpenVMS Version 7.3-2, or Version 8.2 and later (Alpha systems)
- OpenVMS Version 8.2 and later (Integrity systems)

An overview of the TDC V2 architecture is provided in Figure 1, which shows the principal components and their interactions.

Basic Operations

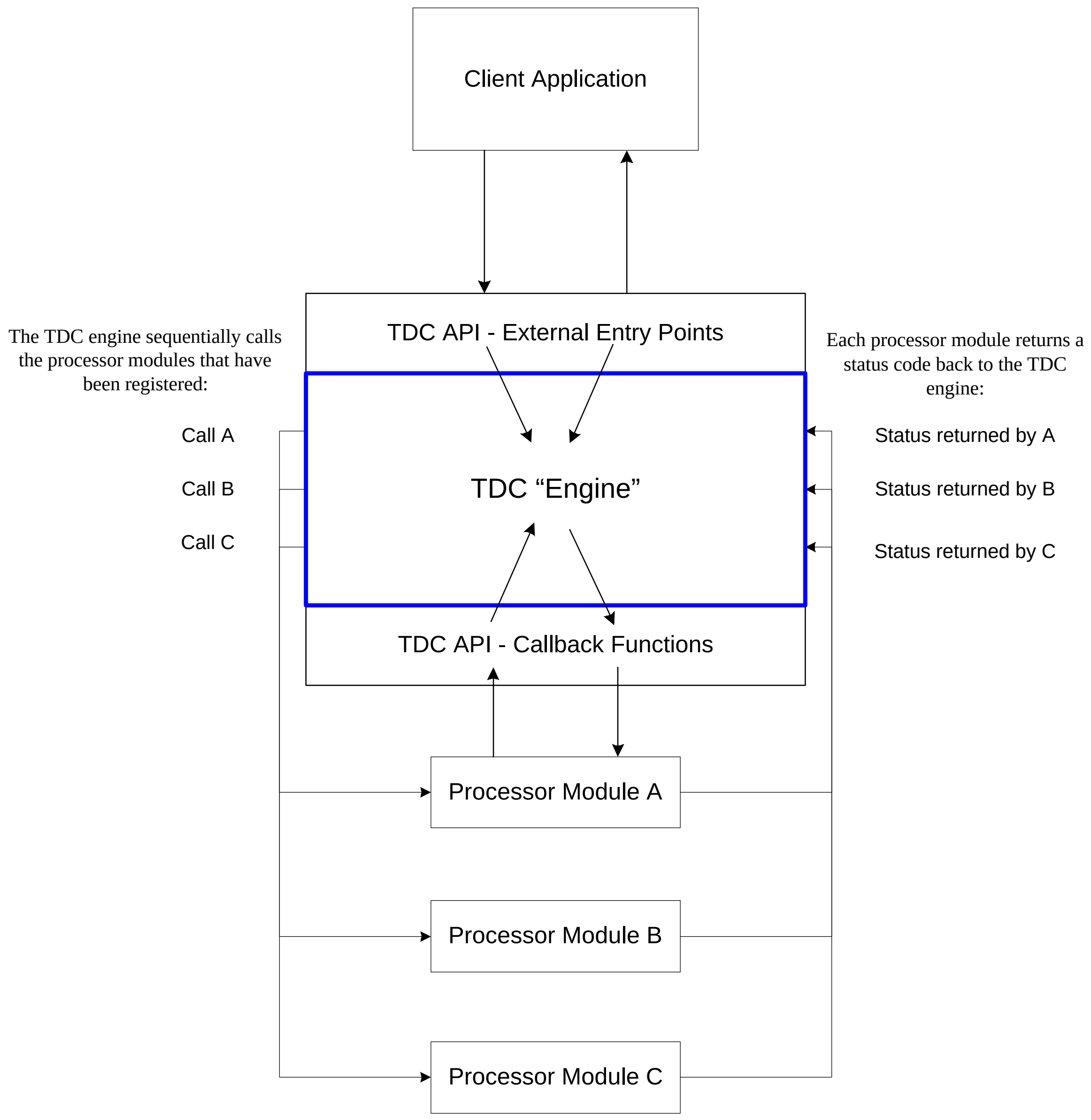
Collection

A collection process is the gathering and optional processing of data, followed by the consumption of that data, typically, but not always, by writing it to a file.

Extraction

An extraction process is the retrieval and optional processing of data from a source other than the OpenVMS operating system. The source will typically be a file created during a collection process.

Figure 1: Overview of interactions among TDC V2 components


Client Application
TDC API - External Entry Points
The TDC engine sequentially calls the processor modules that have been registered:
Call A
Call B
Call C
TDC "Engine"
Each processor module returns a status code back to the TDC engine:
Status returned by A
Status returned by B
Status returned by C
TDC API - Callback Functions
Processor Module A
Processor Module B
Processor Module C

What are Processor Modules?

- The TDC Engine delegates responsibility for producing and processing data records to Processor Modules.
- A Processor Module (PM) is a functional unit that performs some task of interest in the context of collecting or processing system performance data.
- Processor Modules are typically packaged in shareable images. A shareable image that contains one or more PMs is referred to as a processor module library.
- Processor Modules are loaded at runtime, via `LIB$FIND_IMAGE_SYMBOL()`.
- A Processor Module presents itself to TDC, through a TDC-defined interface, as an autonomous entity; while PMs packaged within a library are free to interact among themselves "in the background," each must be capable of independently interacting with TDC.

Processor Module Capabilities

The TDC V2 architecture broadly defines a number of operational capabilities for Processor Modules. An individual Processor Module specifies its actual capabilities after it has been loaded by the TDC Engine. The capabilities specified by a Processor Module may be changed from one operation to another.

The principal Processor Module capabilities defined and recognized by the TDC Engine are:

- **PRODUCER**: the Processor Module provides data records. The data records that it provides can result from information it collects from the operating system, or they can result from its processing of data records provided by other PMs. A producer PM can be invoked during collection operations, and provides exactly one type of data record. 24 producer PMs are provided with TDC Version 2.2.

- **CONSUMER**: the Processor Module consumes data records. A consumer non-destructively utilizes data records produced by other PMs.

  Two consumers are provided with TDC Version 2.2:

  - One (`TDC_COLFIL`) has responsibility for managing the storage of data in a collection file for subsequent processing; it can be disabled to inhibit storage of the data in a file.

  - A second (`TDC_TXTCVT`) handles conversion of data to various text representations (`LIST`, `LISTALL`: display data items in a snapshot/record set/data record/data item listing, with identifying information for data items and data aggregates).

  Several other consumers are provided in source-code form as a part of the TDC V2.2 SDK.

A consumer PM can perform other operations on the data records, perhaps enabling "live" provision of data to other software: broadcasting data records over a network connection; using them to update web pages or a system status display; otherwise feeding them to another application, for example. Any number of consumers can be simultaneously involved in either a collection or an extraction operation.

Because a consumer PM typically concerns itself only with the sets of data records made available to it by the TDC Engine, it should function equally well whether participating in a collection or in an extraction operation.

- **TIMER**: the Processor Module controls the timing of data collection activities. One timer (`TDC_TIMR`) is provided with TDC; it can be overridden by a timer PM supplied by you or by a third party. Exactly one timer must be active during a data-collection process. A timer is optional during data-extraction operations; the TDC-provided timer disables itself for data-extraction operations.

- **FILTER**: the Processor Module examines data records produced by one or more other PMs, deciding which are useful and which can be discarded before being made available to other PMs. No filters are provided with TDC, though any number of filters can be active during either a collection or an extraction operation.

- **EXTRACTOR**: the Processor Module provides the ability to extract (read) data from a file or other data source. Extractors are invoked by the TDC engine only during data-extraction operations. One extractor is provided with TDC; it has responsibility for reading data from the file created by the consumer PM provided with TDC (actually, one PM – `TDC_COLFIL` – provides both functions). Generally, use of a file-building consumer PM other than that provided with TDC implies the necessity of also providing an extractor PM.

- **GROUPER**: the Processor Module forces the inclusion and loading of a set of other PMs; it does not otherwise participate in collection or extraction processes. Three groupers are provided for general use with TDC: `DEFAULT`, `CLUSTER`, and `ALL`; several others are provided for use with specific applications.

- **SUPPORTER**: the Processor Module provides support services for one or more other processor modules, generally in the nature of acquisition/initialization and tear-down/release of shared resources. Supporters are not called during processing of snapshots. Two supporters are provided with TDC: `TDC_LWS` and `TDC_EXEC`.

- **MONITOR**: the Processor Module monitors the state of collection and extraction operations, generally for the purpose of making that information available to other applications.

An individual Processor Module may provide a combination of the capabilities defined above, as is the case with the default PM provided with TDC for creating (consumer capability) and reading (extractor capability) files.

Processor Module Synchronization

The TDC V2 architecture defines a set of points at which the Engine can invoke a Processor Module. All PMs are invoked prior to the start of a collection or extraction process, providing them an opportunity to self-initialize; all PMs are also invoked at the end of a collection or extraction process, providing them the opportunity to free resources.

The following invocation points are also defined by the architecture. After it has been loaded by the Engine, a Processor Module may specify at which of the following synchronization points it should be invoked:

- **Pre-Baseline**: A Processor Module which specifies pre-baseline synchronization will be invoked before the Engine begins periodic data collections. This synchronization point, applicable only to `TIMER`s and `MONITOR`s, provides the ability to defer data collection until a specified time. This synchronization point is not available during extraction operations.

- **Snapshot**: A Processor Module which specifies snapshot synchronization will be invoked as a part of each of the Engine's periodic data collections. It is during these invocations that a PRODUCER would actually provide data records based on its scanning of the system hardware or software. During extraction operations, a processor module that specifies snapshot synchronization will be called at each snapshot, after all of its data records have been read and stored for the current snapshot (processor modules which contributed no data records to the current snapshot will not be called). Relevant FILTER PMs would also be invoked to filter data records provided by PRODUCERS.

- **Snapshot-End**: A Processor Module which specifies snapshot-end synchronization would be invoked after all snapshot-synchronized PRODUCER PMs. Such a PM might itself be a PRODUCER, which produces records derived from records produced by other PMs. CONSUMER PMs would typically specify snapshot-end synchronization. During extraction operations, a processor module that specifies snapshot-end synchronization will be called at each snapshot, after all data records (of all types) have been read and stored for the current snapshot (non-CONSUMER processor modules which contributed no data records to the current snapshot will not be called).

- **Collection-End**: A Processor Module which specifies collection-end synchronization would be invoked after all collection activities have completed, and the collection file, if any, has been closed. A CONSUMER PM synchronized on collection-end might alert a software application that a collection file is available for processing, or that it should expect no more data to be sent to it from the PM.

A Processor Module may specify any combination of synchronization options. Figure 2 shows the synchronization of PM invocations by the TDC Engine during a collection operation.

Extraction operations differ from collection operations in that an EXTRACTOR PM is invoked to provide a single system snapshot; processor modules are then invoked to process that data, and a TIMER PM, if present, is invoked to time extraction of the next system snapshot; a MONITOR PM, if present, would be invoked at each step. Figure 3 shows the sequence of events during an extraction operation.

Other Processor Module Options

A number of other options are available to assist Processor Modules in their tasks:

- A Processor Module can, after it has been loaded by the TDC Engine, specify that it has a dependency on one or more other PMs. A FILTER does this, for example, to indicate the types of records that it filters. The Engine uses dependency information to determine the order in which PMs should be invoked.

- A Processor Module can, after it is loaded by the Engine, specify any system privileges required for its use. The TDC Engine will not start an operation unless all privileges specified by all PMs are enabled by the user (or by the client application that drives the TDC Engine).

- The API allows Processor Modules to access the Engine through a set of callback functions.

- A Processor Module can, by means of the API, examine or copy data records provided by a different PM.

- A Processor Module can specify that it be invoked to reformat its data records as they are read from a file during an extraction operation.

- A Processor Module can support “external calls.” An external call, placed through an API callback function, allows one PM to invoke another, perhaps to provide some service not otherwise available through the API. The TDC API does not specify the types of external calls that can be made (that is a matter for agreement between the developers of the calling and target PMs). Only a PM that, after having been loaded by the Engine, specifies that it will accept external calls, is a potential target of an external call. A PM that is the target of an external call is provided with information to identify the caller, allowing it to screen external calls.

- A Processor Module can provide multiple views of its data records, sorted in various ways, with appropriate search functions.

- A Processor Module that provides multiple data records at each snapshot can provide a callback function to be used to match records from the current and previous snapshots (the TDC Engine provides this matching capability on behalf of PMs that produce only one record per snapshot).

- A Processor Module can provide a callback function that creates "delta" records whose data items reflect differences between the corresponding data items from two raw data records.

- A Processor Module can disable itself, if it discovers that it can no longer appropriately perform its task, without necessarily affecting the operation of other PMs. Note that PMs that have declared a dependency on the disabled PM will receive notification of its changed state, and may, as a result, modify their own operational states.

- A Processor Module can use the API to request that an operation halt prematurely.

Table 1: Processor Modules Supplied with TDC

Name	Purpose	Data Items [*: some items are arrays]
TDC_TIMR	Default TIMER	N/A
TDC_PDSC	Manages processor modules	N/A
TDC_DDSC	Manages data descriptions provided by producers.	N/A
TDC_TXTCVT	Provides text conversions	N/A
TDC_COLFIL	Manages collection file	N/A
TDC_COLHDR	Collection parameters & system HW/OS characteristics	45 HW/OS characteristics; 17 collection parameters
TDC_COLHDR2	Supplementary system HW/OS characteristics	24 HW/OS characteristics
TDC_EXEC	SUPPORTER, provides an interface to the execlet that is used for some process and disk data	N/A
TDC_LWS	SUPPORTER, handles initialization and cleanup of resources used be several other processor modules.	N/A
DEFAULT	GROUPER, forces collection of ADP, CPU, CTL, DEV, DSK, FCP, MEM, NTI, PAR, PRO, & SYS records	N/A
ALL	GROUPER, forces collection of DEFAULT records plus CLU, CVC, CPS, DLM, DTM, GLX, INET, SRV, XFC, & XVC	N/A
CLUSTER	GROUPER forces collection of CLU, CPS, CVC, DLM, DTM, GLX, PAR, and SRV records.	N/A
ADP	Describes H/W adapters	14
CLU	Describes VMSCLUSTER configuration	23
CPU	Describes CPU utilization	20

CPUCFG	Describes characteristics of CPUs	27
CPS	Describes performance of Clusterwide Process Services	10
CTL	Describes I/O controllers & ports	5
CVC	Describes VMSCLUSTER communications (SCS)	19
DEV	Fixed characteristics of storage devices	50
DLM	Describes Distributed Lock Manager performance	50
DSK	Variable characteristics and performance of disk devices	40
DTM	Describes performance of the Distributed Transaction Manager	37
FCP	Describes FCP performance	34
GLX	Describes resources used for Galaxy	11
INET	Describes network software performance	146
MEM	Describes memory utilization	34
NTI	Describes network hardware performance	67
PAR	Provides all SYSGEN parameter values	384
PGFL	Describes page/swap-file utilization	19*
PRO	Describes processes	88
SRV	Describes server (MSCP and TMSCP) performance	58
SYS	Describes miscellaneous system metrics	75
XFC	Describes cache performance	44*
XVC	Describes volume-caching	37*

Figure 2: Synchronization of calls to Processor Modules by TDC engine during a collection.

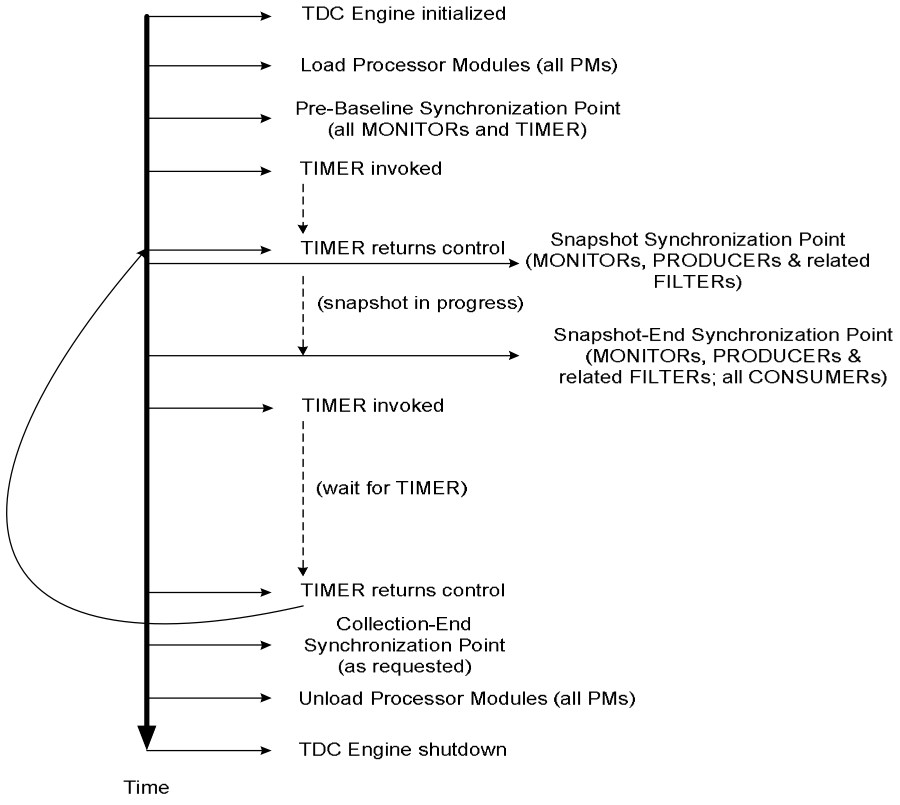
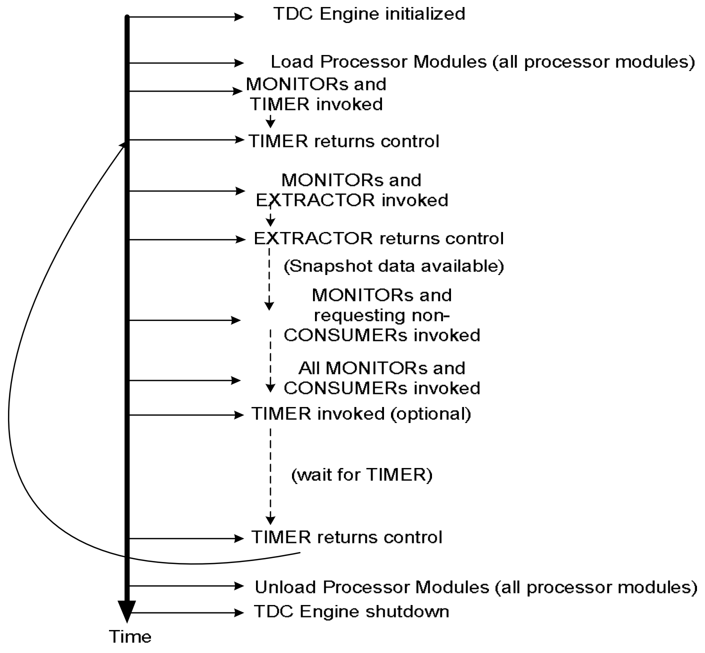

TDC Engine initialized
Load Processor Modules (all PMs)
Pre-Baseline Synchronization Point (all MONITORs and TIMER)
TIMER invoked
TIMER returns control
Snapshot Synchronization Point (MONITORs, PRODUCERs & related FILTERs)
(snapshot in progress)
Snapshot-End Synchronization Point (MONITORs, PRODUCERs & related FILTERs; all CONSUMERs)
TIMER invoked
(wait for TIMER)
TIMER returns control
Collection-End Synchronization Point (as requested)
Unload Processor Modules (all PMs)
TDC Engine shutdown
Time

Figure 3: Overview of the extraction process.


TDC Engine initialized
Load Processor Modules (all processor modules)
MONITORs and TIMER invoked
TIMER returns control
MONITORs and EXTRACTOR invoked
EXTRACTOR returns control
(Snapshot data available)
MONITORs and requesting non-CONSUMERs invoked
All MONITORs and CONSUMERs invoked
TIMER invoked (optional)
(wait for TIMER)
TIMER returns control
Unload Processor Modules (all processor modules)
TDC Engine shutdown
Time

How does TDC V2 communicate with Processor Modules?

Global context structure

A global context structure stores all information relevant to the current data-collection or data-extraction operation, including the inter-snapshot timer, the number of snapshots to be taken, the start and end times for the collection, and various other options. The global context also contains a pointer to a public data buffer and to the current and previous snapshots.

The public data buffer is used to pass data records between the TDC API and Processor Modules.

The current snapshot represents data currently being collected to hand over to the consumer PMs, and the previous snapshot represents the data most recently processed by those consumers. A producer PM could generate “delta” records, representing the change in some set of metrics between snapshots, by comparing corresponding records in the two available snapshots.

A pointer to the global context is provided to the Processor Module at each invocation.

Private context structure

A private context is created by the TDC Engine for each PM that is participating in an operation. That private context is used to exchange PM-specific information between the Engine and the Processor Module: required privileges, supported data formats, capabilities, synchronization options, error codes, etc.

The private context contains a pointer for exclusive use of the Processor Module. The Processor Module should generally use it to reserve a block of heap memory for its own global data, thus avoiding the use of global variables, which would present a problem in a multi-threaded implementation.

The private context also contains a pointer that can be used by the Processor Module in order to share data with other processor modules.

Public and Private data records

The private context also contains a data buffer pointer. That will be used to exchange “private” data records between the TDC Engine and the Processor Module.

Each PRODUCER Processor Module is responsible for exactly one type of data record. The data record may, however, have two or more different representations.

The public form of the record is designed for efficient and convenient run-time access. That is the form that is stored in snapshots and that is made available, through those snapshots, to other

Processor Modules. The public form of the record is passed between the Engine and the PM through a pointer in the global context.

The optional private form of the record should be designed to conserve storage space in order to minimize the disk footprint of a data collection file. It is passed between the Engine and PM through a pointer in the private context. It is the responsibility of a PRODUCER PM that supports both public and private data records to convert between the two formats.

Processor Module invocations

Arguments passed

The TDC Engine passes three arguments when calling a Processor Module:

- Pointer to the global context, which provides a "Do What" code from which the PM can determine the actions expected of it at that call; other data provides the "global" context in which the operation is occurring, and can be read, or sometimes modified, by the PM. This argument is guaranteed to be non-`NULL`.

- Pointer to the PM's private context, which provides access to context information maintained by the PM, and by the TDC Engine on behalf of the PM. This argument is guaranteed to be non-`NULL`.

- Pointer to a record set which represents a set of records of a particular type, usually the data records maintained by the PM. This argument may be `NULL`.

A Processor Module which accepts "external" calls will receive an additional three arguments during an external call:

- A pointer to a processor description record for the calling PM.

- A pointer to use to receive input arguments from that caller. The specific way in which this pointer is used is not defined by the TDC architecture.

- A pointer to use to return output values to that caller. The specific way in which this pointer is used is not defined by the TDC architecture.

Return values

A Processor Module returns one of a defined set of values as its return value at each invocation. If the PM has encountered an unexpected condition during that invocation, it should appropriately describe that condition using the set of status code items in its private context.

If a PM's return value indicates a "fatal" error, the TDC Engine will, as quickly as is convenient, begin to abort the current operation.

A PM should never call `exit()` or `LIB$STOP()` or signal a fatal condition, as those sorts of abort actions might not allow the Engine to shut down properly, and the operating system could, during a collection operation, be left in a non-optimal state (perhaps because data-collection execlets may not be unloaded).

Messaging and Event-Logging

The TDC Engine provides for uniform reporting of "interesting" situations through a callback function accessed through the global context.

A set of macros is defined, in TDC_COMMON.H, for use with that callback function. In general, the following should be used; besides invoking the callback function, they will set appropriate status values in the PM's private context:

- **TDC_SET_PCTX_FTLERR** to report that the PM has encountered a fatal error. It is appropriate to report a fatal error only when the situation is such that the TDC Engine should be shut down, perhaps because the PM has detected that its data has been compromised and that corruption of other elements of the TDC Engine's address space is a possibility. This will most often occur as a result of a programming error.

- **TDC_SET_PCTX_ERROR** to report that the PM has encountered an error. It is appropriate to report an error when the PM decides that it cannot provide the services expected of it, perhaps because the set of devices or software monitored by a PRODUCER PM has gone offline. The macro will mark the PM as disabled.

- **TDC_SET_PCTX_WARN** to report a less severe condition, that should, nonetheless be logged.

A set of macros in addition to those listed above provides for a more general messaging capability, including the filtering of messages according to a "trace level" specified within the global context.

A Processor Module should never generate output that would be visible to an end-user except through the macros and callback function provided for that purpose.

What is a Client Application?

An application is an executable image that performs some set of tasks. A TDC Client Application is an application that makes use of the TDC Engine, through its API, to perform data-collection and/or –extraction tasks.

A Client Application might be something built specifically to use TDC to collect or read data, or it might be an existing application to which the capability to interact with TDC has been added, again for collecting or extracting data.

A Client Application can coordinate with Processor Modules (provided with the Client Application) to better perform some task, or it can be entirely independent of any particular Processor Modules, relying on those provided with TDC or made available by third parties.

TDC does not support direct interaction between Client Applications and Processor Modules, but neither does it attempt to prevent such interactions. TDC does, however, support the ability to include one Processor Module as a part of the client application executable image.

The TDC API provides the following principal functions for a Client Application:

- Initialize the API (`TDC_INIT`)
- Load processor modules (`TDC_REGISTER`)
- Prepare to start an operation (`TDC_PREPARE`)
- Start a read or collection operation (`TDC_START`) and run it to completion
- Read data from a collection file one snapshot at a time (`TDC_READ_SNAPSHOT`)
- Collect data one snapshot at a time (`TDC_COLLECT_SNAPSHOT`)
- Transfer control to the TDC Engine to run the current operation to completion after one or more calls to `TDC_READ_SNAPSHOT` or `TDC_COLLECT_SNAPSHOT` (`TDC_CONTINUE`)
- Retrieve information about the TDC execution environment (`TDC_GET_INFO`)
- End an operation (`TDC_END`)
- Shut down the API (`TDC_FINISH`)
- Other, more specialized, functions

The interaction between a client application and the TDC Engine is illustrated in Figure 4.

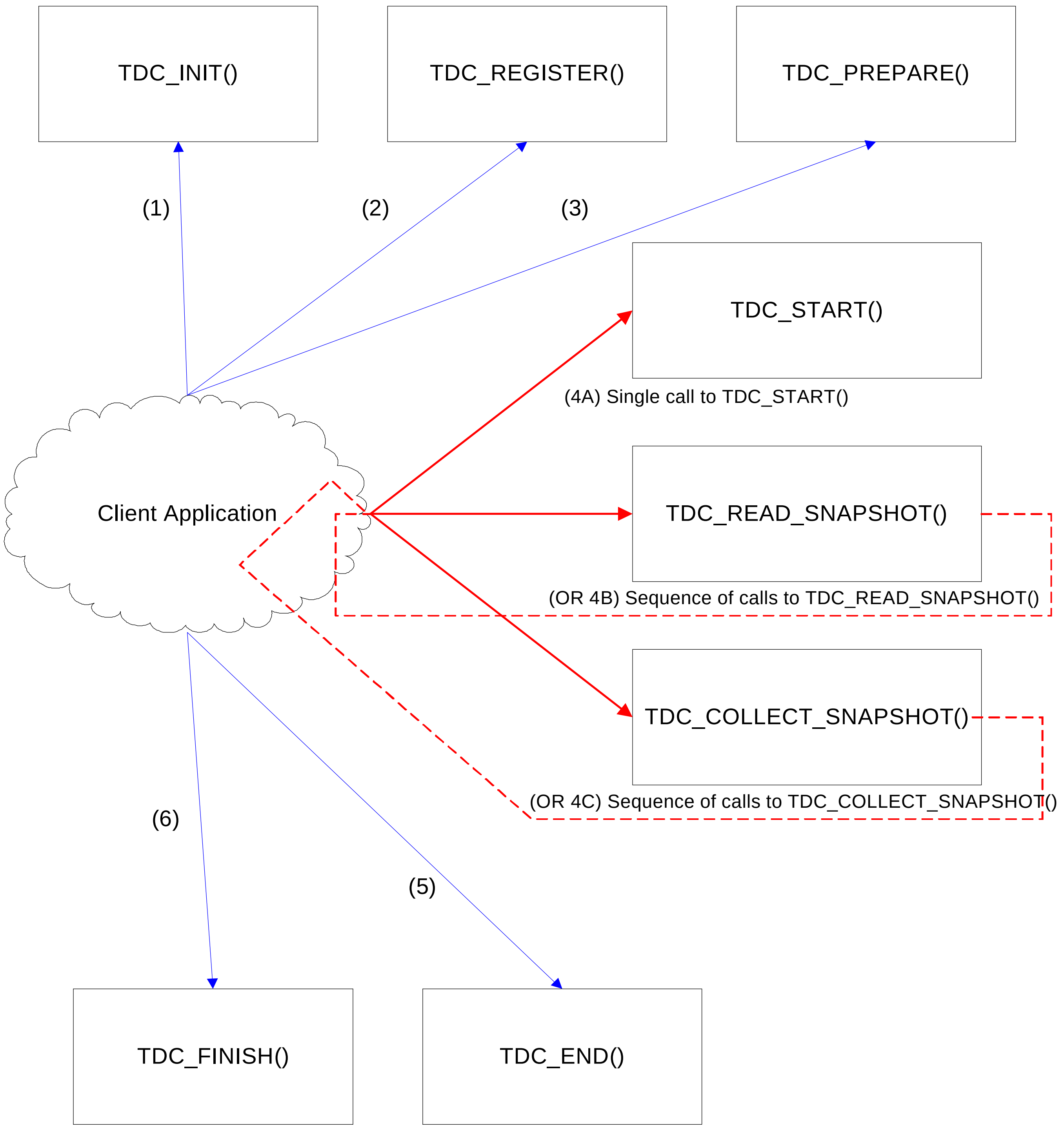


Figure 4: Client application interactions with TDC Engine

How does TDC V2 manage data?

Snapshots

During a data-collection operation, the TDC Engine periodically calls the set of loaded PRODUCER Processor Modules to sample the state of the hardware and operating system. This is a snapshot operation. As a result of a snapshot operation, each of the relevant PRODUCERs will have created a set of data records collectively representing the state of monitored aspects of the computer at that snapshot.

That set of data is aggregated into a snapshot data structure. A snapshot data structure has a header, of type `TDC_SnapshotHeader_t`, through which the snapshot data can be accessed.

The TDC Engine's global context contains a pointer to the "current" snapshot, the one most recently created, at member `TDC_CTX_CurrentSnapshot`. It is that snapshot which PMs will utilize to collect and later process the current data.

The global context also contains a pointer to the previous snapshot at member `TDC_CTX_PreviousSnapshot`.

Snapshot structures can be copied using a callback function accessible through the global context.

It is also possible for software to mark an ongoing interest in a particular snapshot to prevent its being deleted. This is done by "reserving" the snapshot, and, later, "releasing" it. This mechanism is more efficient than copying, as a snapshot can potentially contain a large number of data records.

Record Sets

The snapshot structure contains pointers to record sets. A record set, accessed through a record set header of type `TDC_Set_t`, contains all the data records of a particular type that were created as a part of a snapshot operation. If there are 10 PRODUCER PMs participating in a collection operation, the resulting snapshot is likely to contain 10 record sets, each contributed by one of those PRODUCERs.

A record set represents all the records of a particular type that were generated during a single snapshot operation. The record set header contains pointers to (the public forms of) those data records. PRODUCER PMs will add data records to a snapshot by means of a callback function accessed through the global context; adding records through other means is not supported and may result in undesirable consequences, including corruption of a data file.

Because the most recent snapshots and their associated record sets are accessible through the TDC Engine's global context, they can be examined by any of the Processor Modules that are currently loaded. They can also be examined by a Client Application.

It may at various times be desirable to have different views of a record set. This ability is provided through a mechanism referred to as “cloning.” The TDC Engine’s global context provides a pointer to a function that will clone a record set. During the process of cloning, the record set header is copied, and the clone receives a set of pointers to the original data records rather than pointers to copies of the records. The set of pointers in the clone record set can be reordered in any way without affecting the original data records.

Data records

All records, including public and private data records, begin with a “record prefix” which serves to identify the record type and other characteristics, including time of creation and size.

Ensuring that a record has the correct prefix data is primarily the responsibility of the Processor Module responsible for that data record.

The TDC Engine assigns each PM a record ID value when the PM is loaded. The PM has to insert that ID into the prefix of each record, whether public or private, that it produces. The PM also has to track the size of each record, insuring that the size value in the prefix is correct. There are some prefix flags that the PM can set or clear to describe the state of a record.

The TDC Engine will add an appropriate timestamp value to data records as they are inserted into a snapshot, though that task can instead be handled by the PM.

Organization

Figure 5 shows the principal data structures and their inter-relationships. In general, data access begins with a snapshot - typically the current snapshot, which is reached through a pointer within the API’s global context. The snapshot header provides a pointer to an array of pointers to record set headers, with one record set header per data record type represented within the snapshot. Each record set header provides a pointer to an array of data pointers, with one data pointer per data record.

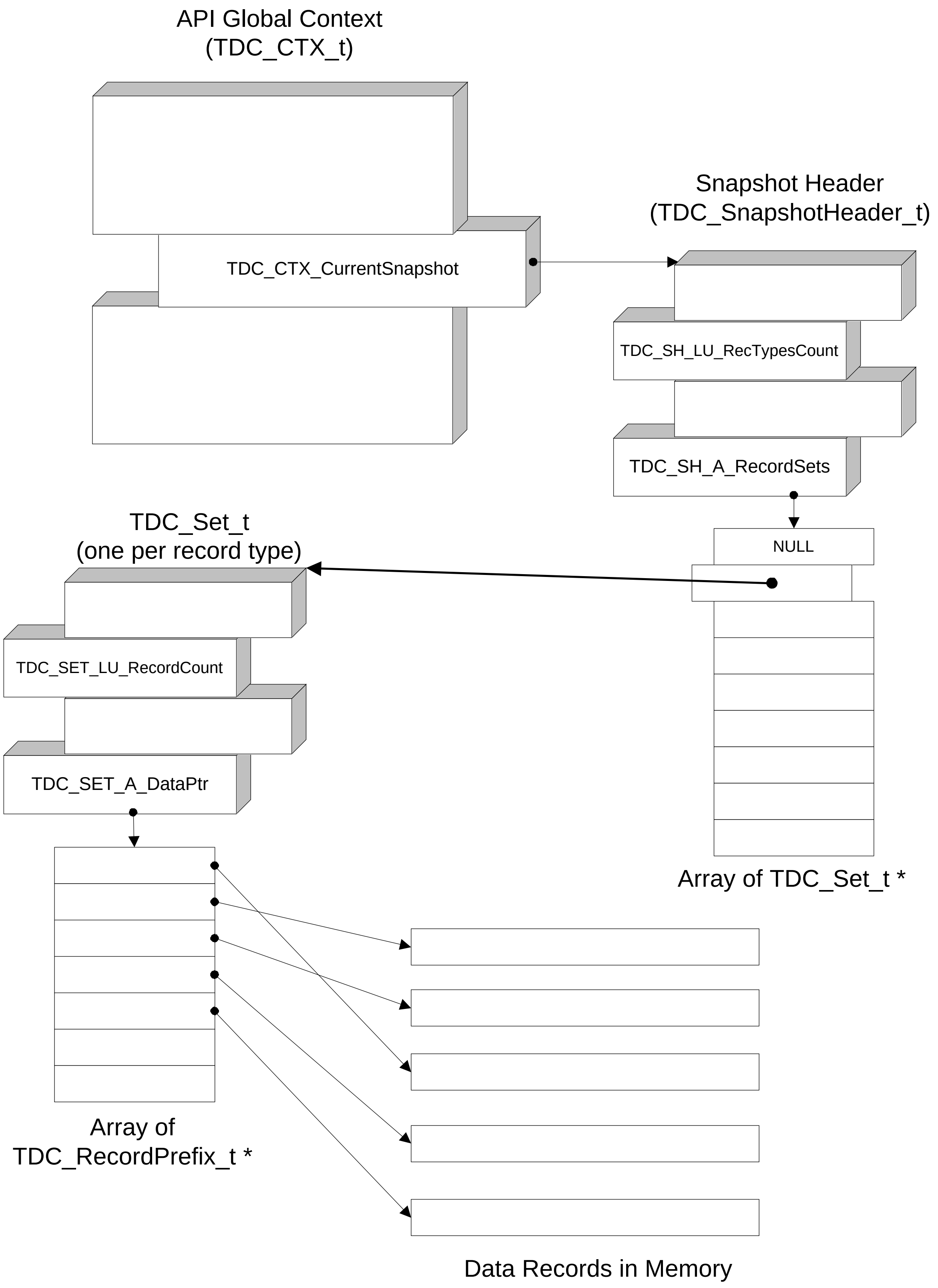


Figure 5: Data Organization and Access