Introduction

To instrument application software metrics on NonStop servers you use the ASAPX application programmer interface (`ASAP_<functions>`) as documented in the Independent Products TIM manual. Like a great many low-level system interfaces and because it is based on direct memory sharing, your use of this interface will benefit greatly from forethought with respect to:

- Encapsulation
- Error recovery
- Online replacement of ASAP

Encapsulation

By encapsulating all your ASAPX calls you will assure your instrumentation is consistent. Encapsulation also provides a single interface into the ASAPX API calls allowing improvement to be shared among applications.

Error recovery

If you encounter a problem with the `ASAP` API, you'd like recovery to be consistent among your applications. By having a consistent set of error recovery routines, you can be more careful and sophisticated about how you retry operations. For example, re-doing memory operation indefinitely could cause looping processes, but retrying a subsequent sample internal will likely prove successful.

Also if you have a common set of error recovery routines, and you encounter errors on calls such as `ASAP_REGISTER_` API, you can avoid mistakes like accidentally attempting too many registration (`FILE_OPEN_`) messages.

Online ASAP replacement

ASAP is an independent product, which means: no Sysgen is required for replacement.

If the ASAP shared segment has not changed format, you can upgrade your ASAP environment without stopping your application!

Thus, application downtime because of an ASAP update is completely avoidable if you have a consistent set of ASAPX API wrapper methods.

Based on experiences with the above topics, Rabobank has built an API on top of ASAP. The higher-level API takes responsibility for error-recovery, throttling and online server replacement.

Using this API means unified (and controllable behavior of applications using ASAP). This document contains the description of those API's (DOMAIN API's). This package contains 7 API's:

- `ASAPENV_INI()` → setup ASAP environment
- `DOMAIN_INI()` → setup measurement
- `DOMAIN_STOP()` → stop measurement
- `DOMAIN_START()` → start measurement
- `DOMAIN_SET_DATAITEM()` → update user defined counters
- `DOMAIN_SET_STATE()` → update system defined state field
- `DOMAIN_SET()` → flush user defined counters and state

ASAPENV_INI

This API sets up the ASAP-environment. The ASAP-environment will be carried forward in an environment structure. The environment structure must be passed as an input parameter to `DOMAIN_INI` API. The environment structure contains primarily the name of the ASAP-environment and it chains all registered domains (`DOMAIN_INI` invocations). This function should only be called once (and before any `DOMAIN_INI` invocation).

The chaining is used in case of a controlled shutdown of ASAP for online replacement. Controlled shutdown is ignited when the operator gives the disabled stats command from the ASAPX-command prompt. On the next invocation of a DOMAIN API (it doesn't matter which one) by the application, all chained domains will be de-registered automatically and the last domain in the chain will release the shared ASAP segment file.

```
{CALL } ASAPENV_INI      (      environment         ! o
                         ,      asapid:asapidlen    ! i:i
                         ,      segmentid           ! i
                         ,      segmentbase         ! i
                         ,    [ resetinterval ]     ! i
                         ,    [ version ]           ! i
                         ,    [ timeout ]           ! i
                         );
```

environment *output*

`STRUCT:ref:EXT:*`

Environment structure - `typedef asapenv_def` - which will be passed as input parameter to the `DOMAIN_INI` API.

asapid:asapidlen *input:input*

`STRING:ref:EXT:*,INT:value`

Is the character prefix of the ASAP-environment to be used when opening the ASAPXMON process. Prefix contains 4 characters at maximum (including a leading $). Prefix should be specified in local form.

segmentid *input*

`INT:value`

Identifier to be used by the `ASAP_REGISTRATION_` procedure in its call to `SEGMENT_ALLOCALTE_`. Default value is 0.

segmentbase *input*

`INT(32):value`

The segment base address to be used by the `ASAP_REGISTRATION_` procedure. Default value is 0D.

`resetinterval` *`input`*

`INT:value`

Specifies the amount of time (in seconds) that `DOMAIN_INI` API should take in account in case an error that requires a re-registration within ASAP (using `ASAP_REGISTER_` API). Default value is 180 seconds. This parameter throttles down the number of messages send to the ASAPXMON process in case of an erroneous situation.

`version` `input`

`INT:value`

Identifies the version of the application using the DOMAIN API's. Version will be passed opaque to `ASAP_REGISTER_` call.

`timeout` *`input`*

`INT(32):value`

Specifies the maximum amount of time the `DOMAIN_INI` API's will wait for domain registration in ASAP. Timeout is specified in milliseconds. Default value 100 msec. If a timeout occurs, re-registration occurs at the next DOMAIN API invocation (and with reset interval expired).

Considerations

Since the DOMAIN API's are wrappers around the ASAP API's, the (fault) behavior of the ASAP API's is encapsulated in the DOMAIN API's (and not visible for the developer). To trace the behavior of the ASAP API in case of problems, it is possible to initialize a trace log by setting the DEFINE `=MTRIC_LOG_FILE`.

If the define `=MTRIC_LOG_FILE` is set, `ASAPENV_INI` will open this file using its define name. It should be noted that `ASAPENV_INI` doesn't create a log file. Transaction propagation to this logfile is turned off since `ASAPENV_INI` invokes `SETMODE(<logfilefnum>,117)`

The log file option is intended for use during application development, using a terminal as a logging device.

```
#SET #INFORMAT TACL
ADD DEFINE =MTRIC_LOG_FILE,class map,file [#MYTERM]
RUN<D> <app> /run opts/
```

DOMAIN_INI

This API initializes the domain structure. The domain structure contains a maximum of 12 slots measurements (dataitems). ASAP uses these dataitems to calculate software metrics. The domain structure is passed as input/output parameter to all the other DOMAIN API's. The `DOMAIN_INI` API should only be called once. If `DOMAIN_STOP_` is called with removal flag set, the domain structure will be void. A renewed call to `DOMAIN_INI` is necessary.

```
{ CALL } DOMAIN_INI     (     domain                    ! o
                        ,     domainid:domainidlen      ! i:i
                        ,     environment               ! i
                        ,     flags                     ! i
                        ,   [ dataitem_0,math_0 ]       ! i,i
                        ,   [ dataitem_1,math_1 ]       ! i,i
                        ,   [ dataitem_2,math_2 ]       ! i,i
                        ,   [ dataitem_3,math_3 ]       ! i,i
                        ,   [ dataitem_4,math_4 ]       ! i,i
                        ,   [ dataitem_5,math_5 ]       ! i,i
                        ,   [ dataitem_6,math_6 ]       ! i,i
                        ,   [ dataitem_7,math_7 ]       ! i,i
                        ,   [ dataitem_8,math_8 ]       ! i,i
                        ,   [ dataitem_9,math_9 ]       ! i,i
                        ,   [ dataitem_A,math_A ]       ! i,i
                        ,   [ dataitem_B,math_B ]       ! i,i
                        );
```

domain *input/output*

`STRUCT:ref:EXT:*`

Domain structure that contains the description of the measurement to be activated. Domain structure is passed as input/output parameter to all other DOMAIN API's.

domainid:domainidlen *input:input*

`STRING:ref:EXT:*,INT:val`

a 1 to 5 level domain name. The name is of the form `<1>\<2>\<3>\<4>\<5>`, where each level can be any length up to the maximum of 64 characters. If bit 14 of `<flags>` is reset, the process name of the caller will act as the 5th level. In that case the maximum length of the domain name will be 6 characters less.

environment *input*

`STRUCT:ref:EXT:*`

Environment structure returned by `ASAPENV_INI` API.

flags *input*

`INT:val`

a control word indicating the following:

`flags.<13>`	allow replace operation on non-constant dataitems
`flags.<14>`	no auto concatenation of the callers processname to the domain name
`flags.<15>`	domain should be started immediately, which makes `DOMAIN_START` invocation unnecessary.

dataitem_<x> *input*

`INT:val`

Identifies the dataitem that will be stored in the domain structure. A maximum of 12 different dataitems can be specified

math_<x> *input*

`INT:val`

Is the type of math to be used by default in `DOMAIN_SET_DATAITEM` API. The options are: Add (0), Replace (1), ReplaceText (2).

DOMAIN_START

This API activates a DOMAIN within the ASAP-environment. Activation means that every sample interval the ASAP-sensors will be polled by the ASAP-management environment, compared against objectives and reported to operations (only started domains are actively monitored).

```
      { CALL } DOMAIN_START   (     domain      ! i/o
                                    );
```

domain   *input/output*

    STRUCT:ref:EXT:*

Domain structure that contains the description of the measurement to be activated.

DOMAIN_STOP

This API deactivates a DOMAIN within the ASAP-environment. Deactivated domains won't be monitored by the ASAP-management environment, but remain visible for the operations staff.

There is a special flag to remove a domain physically from the ASAP-management environment. It should only be used when an application is removed from production (since is removes the notion of existence).

```
{ CALL } DOMAIN_STOP(   domain       ! i/o
                   ,  [ flags ]      ! i
                        );
```

domain *input/output*

`STRUCT:ref:EXT:*`

Domain structure that contains the description of the measurement to be deactivated/removed.

flags *input*

`INT:val`

Boolean that determines whether a DOMAIN should be removed physically from ASAP-environment. Default value is FALSE (0), which means no removal.

DOMAIN_SET_DATA_ITEM

This API stores the value of a dataitem (sensor) in the domain structure. The domain structure is a container for 12 different dataitems (sensors). `DOMAIN_SET_DATA_ITEM` doesn't invoke an ASAP API, the `DOMAIN_SET` API does. Hence invoking `DOMAIN_SET` will flush the domain structure to the ASAP-environment.

Depending on the math, the value of the dataitem will be added to or replace the current value in the domain structure. `DOMAIN_SET` will reset the dataitems after invoking the `ASAP_UPDATE_` or `ASAP_UPDATELIST_ to` start again with a clean sheet.

```
{ value := } DOMAIN_SET_DATA_ITEM   (     domain      ! i/o
                                    ,     dataitem    ! i
                                    ,     value       ! i
                                    ,   [ math ]      ! i
                                          );
```

value *return*

`FIXED`

Returns the accumulated total (from the domain structure) for the specified `<dataitem>`

domain *input/output*

`STRUCT:ref:EXT:*`

Domain structure that contains the description of the measurement to be modified. A specific dataitem `<dataitem>` will be updated with value `<value>`. The kind of update (add or replace) operation depends on the value of `<math>`.

dataitem *input*

`INT:val`

Identifies the dataitem that should be updated. An application can use a maximum of 12 dataitems. Value between 0 and 11 should therefore be specified.

value *input*

`FIXED:val`

is the amount to add/replace to the dataitem specified in `<domain>`

math *input*

`INT:val`

is the type of math to be used in `DOMAIN_SET_DATA_ITEM` API. The options are: Add (0), Replace (1), ReplaceText (2). Replace can also be used for non-constant dataitems if the register flag – bit 13 – is set in `DOMAIN_INI`.

If specified, it temporary overrides the math set in `DOMAIN_INI`.

DOMAIN_SET

This API flushes the collected dataitems in the domain structure to the ASAP-managment environment. Normally this API is called with just 1 parameter - the domain structure - where the contents of the domain structure is set by `DOMAIN_SET_DATA_ITEM` and `DOMAIN_SET_STATE` API's. However it is possible to flush your data immediately by setting those values using this API (and bypassing `DOMAIN_SET_DATA_ITEM`).

```
{ CALL } DOMAIN_SET     (     domain                  ! i/o
                        ,  [ dataitem_0,value_0 ] ! i,i
                        ,  [ dataitem_1,value_1 ] ! i,i
                        ,  [ dataitem_2,value_2 ] ! i,i
                        ,  [ dataitem_3,value_3 ] ! i,i
                        ,  [ dataitem_4,value_4 ] ! i,i
                        ,  [ dataitem_5,value_5 ] ! i,i
                        ,  [ dataitem_6,value_6 ] ! i,i
                        ,  [ dataitem_7,value_7 ] ! i,i
                        ,  [ dataitem_8,value_8 ] ! i,i
                        ,  [ dataitem_9,value_9 ] ! i,i
                        ,  [ dataitem_A,value_A ] ! i,i
                        ,  [ dataitem_B,value_B ] ! i,i
                        );
```

domain *input/output*

`STRUCT:ref:EXT:*`

Domain structure that contains the description of the measurement to be modified. A specific dataitem `<dataitem>` will be updated with value `<value>`. The kind of update (add or replace) operation depends on the value of `<math>`.

dataitem_<x> *input*

`INT:val`

Identifies the `<x>`st dataitem that should be updated. An application can use a maximum of 12 dataitems. Value between 0 and 11 should therefore be specified.

value_<x> *input*

`FIXED:val`

is the amount to add/replace to the `<x>`st dataitem specified in `<domain>`. The math specified in `DOMAIN_INI` determines the used operator (add or replace).

DOMAIN_SET_STATE

This API overrides the system defined state Dataitem within ASAP. It should be used very carefully since you are taking care of tasks normally executed by ASAP (like the automatic invalidation of a measurement in case of an Abend). However this API is very useful for processes that monitor the state on behalf of objects.

```
{ CALL } DOMAIN_SET_STATE(    domain            ! i/o
                        ,     state             ! i
                        ,   [ text:textlen ]    ! i:i
                              );
```

domain *input/output*

`STRUCT:ref:EXT:*`

Domain structure that contains the description of the measurement to be modified.

state *input*

`INT:val`

Overrides the statemodel of ASAP. Valid states are Unknown (0), Exists (1), Up (2), Low (3), Medium (4), High (5), Odd (6), Critical (7), Down (8). Unfortunately these definitions are not available in headerfiles of ASAP release 2.0.

text:texlen *input:input*

`STRING:ref:EXT:*`

Defines the text that will replace the labels associated with the standard ASAP statemodel. If omitted, the standard ASAP labels will be used. Textlabel may not exceed more than 15 characters.

Considerations

`DOMAIN_SET` must be invoked to flush the state set by `DOMAIN_SET_STATE`.

FILES

The domain package should contain the following files:

Definitions/include

`mtricc`	→ C header file
`mtriccob`	→ Cobol85 copy members
`mtrictal`	→ (p)TAL function prototype file

Libraries

`mtriclib`	→ CISC library (lib run-option)
`mtricsro`	→ RISC linkable library (using NLD)
`mtricsrl`	→ RISC private shared runtime library (lib run-option)

Linkage (using NLD) for C-source code

3 simple steps:

1. make defines for your object that you want to bind with NLD
2. create NLD infile
3. link objects together

step 1: create defines
```
add define =mtricsro, class map, file $<vol>.<subvol>.mtricsro
add define =asapxsro, class map, file $system.zasapx.asapxsro
add define =CRTLMAIN, class map, file $system.system.crtlmain
add define =LIBCOBEY, class map, file $system.system.libcobey
```

step 2: create NLD-infile
```
=mtricsro                    /* domain sro
=asapxsro                    /* asap sro
=CRTLMAIN                    /* c runtime environment
-set HIGHPIN   OFF
-set SYSTYPE   GUARDIAN /* we're a guardian object
-set SAVEABEND ON
-set RUNNAMED  ON       /* we must be named
-obey =LIBCOBEY         /* C public SRL's
```

step 3: link the object
```
NLD -obey <nldinfile> -o <executable>
```

Example

The source code example on the next page gives an overview on instrumenting ASAP using the high level DOMAIN API's. This application is a fictitious application. It symbolizes a monitor process that monitors processes defined in the ZZKRN subsystem (persistence monitor). Every sample interval (lets say 2 minutes) it probes the state of all processes within ZZKRN and reports on behalf of these processes.

Steps:
1. Application must include the domain header file (`mtricc`)

2. Application must define 2 structures:
- ASAP environment structure (`asapenv_def`)
- domain structure (`domain_def`)

3. Application must initialize the ASAP environment structure using `ASAPENV_INI` API.

4. Application must initialize the DOMAIN structure to define the dataitems that will be used, using `DOMAIN_INI` API.

5. Application must update the dataitems (sensors) at the appropriate place in the application (in the middle of the production process) using `DOMAIN_SET_DATAITEM` API

6. Application must flush the data to the ASAP environment using `DOMAIN_SET` API when all data has been collected.

7. At controlled shutdown (on receiving last close message) the measurements should be disabled using `DOMAIN_STOP` API.

Example (source)

```
/*
 * include domain prototype defintions.
 * don't include any asap stuff.
 */
[1]
#include <mtriclib headerfile>

/*
 * asapenv_def and domain_def type definitions are
 * defined in mtricc headerfile
 */
[2]
asapenv_def asapenv;
domain_def  domain;

/*
 * no constants are defined within asap for mathematical
 * operators, so you must define them by yourselves ...
 */
typedef enum _asapx_math {
  ASAPX_ENM_MATH_ADD = 0,
  ASAPX_ENM_MATH_REPLACE = 1,
  ASAPX_ENM_MATH_REPLACE_TEXT = 2
} asapx_math_t;


#define MAX_B_LEN_ASAPID 4
#define MAX_B_LEN_DOMAINID 64

char

  asapid[MAX_B_LEN_ASAPID+1];
  domainid[MAX_B_LEN_DOMAINID+1];

/*
 * application defined dataitems
 */
typedef enum _appl_dataitem {
  APPL_ENM_DIT_CONFIGURED = 0,
  APPL_ENM_DIT_STARTING = 1,
  APPL_ENM_DIT_RUNNING = 2,
  APPL_ENM_DIT_STOPPING = 3,
  APPL_ENM_DIT_SUSPENDED = 4,
  APPL_ENM_DIT_FROZEN = 5,
  APPL_ENM_DIT_UNKNOWN = 6
} appl_dataitem_t;


fixed

  configured = 0ll,
  starting = 0ll,
  running = 0ll,
  stopping = 0ll,
  stopped = 0ll,
  suspended = 0ll,
  unknown = 0ll;
```

```
int main(int argc, char *argv[], char **envp) {
  strcpy(asapid,"$SLM");
[3]
  ASAPENV_INI(&asapenv,asapid,(short)strlen(asapid)); /* setup env */
  strcpy(domainid,"PCMON\\PROCESS");
[4]
  DOMAIN_INI(&domain, /* register all dataitems */
             domainid,(short)strlen(domainid),
             &asapenv,flags,
             APPL_ENM_DIT_CONFIGURED,ASAPX_ENM_MATH_REPLACE,
             APPL_ENM_DIT_STARTING,ASAPX_ENM_MATH_REPLACE,
             APPL_ENM_DIT_RUNNING,ASAPX_ENM_MATH_REPLACE,
             APPL_ENM_DIT_STOPPING,ASAPX_ENM_MATH_REPLACE,
             ...
             APPL_ENM_DIT_UNKNOWN,ASAPX_ENM_MATH_REPLACE);
  while (true) {
    ...
    ... business logic
    ...
    while (!processed all processes) {
      /*
       * retrieve process within ZZKRN subsystem
       */
      configured++
      conditional increment of other counters depending
      on returned state of process in ZZKRN subsystem
      /*
      /* increase process state counters depending on outcome
      /* of the subsequent queries
      /*
    }
[5]
    DOMAIN_SET_DATAITEM(&domain,APPL_ENM_DIT_CONFIGURED,configured);
    DOMAIN_SET_DATAITEM(&domain,APPL_ENM_DIT_STARTING,starting);
    DOMAIN_SET_DATAITEM(&domain,APPL_ENM_DIT_RUNNING,running);
    DOMAIN_SET_DATAITEM(&domain,APPL_ENM_DIT_STOPPING,stopping);
    ....
    DOMAIN_SET_DATAITEM(&domain,APPL_ENM_DIT_UNKNOWN,unknown);
[6]
    DOMAIN_SET(&domain); /* flush to ASAP */
    DELAY(wait for n-seconds before starting next sample interval);
  }
[7]
DOMAIN_STOP(&domain); /* turn off measurements */
return 0;
}
```