ctys-common-addresssyntax(7)

Definition of the
Generic Address Superset

November 26, 2010

Contents

List of Figures

.

1 General

This document describes the common generic address syntax for single machines and groups of entities. This suffices all supported systems and may for some plugins applicable as a subset only.
The current version provides almost only the <**machine-address**> and the **GROUPS** objects, thus the remaining definitions were required for the design of an extendable overal concept.

2 Basic Elements

The addressing facility including the namebinding is splitted into a logical description as a general view and it's conrete adaptions which could be implemented by multiple presentations. The foreseen and implemented syntax scanners are designed to allow implementation in a straight-forward manner allowing an simple implementation of hierarchical structured syntax definitions by nested loops.

The following characters are reserved syntax elements, the full set and description is given in the chapter "Options Scanners - Reserved Characters".

'=' Seperator for option and it's suboptions.

',' Seperator for suboptions belonging to one set of suboptions.

':' Seperator for suboption keys and it's arguments.

The current syntax description may not yet formally be absolutely correct nor complete, but may cover the intended grade of open description and required understanding for it's application. Some modifications are still under development.

3 Syntax Elements

The following namebinding founds the superset of addressing attributes, which supports explicit addressing of targets as well as generic addressing of multiple targets by using search paths and content attributes in analogy to wildcards, a.k.a. keywords or attribute value assertions.

The given sub-options are defined not to be order dependent, the keywords are case-insensitive.

The contained paranthesis, angle, and square brackets are just syntactic helpers. When they are part of the syntax, they will be quoted with single quotation marks.

The top-level addressed entity is the APPLICATION, thus here the <**target-application-entity**>. This contains in analogy to the OSI model the machine as well as the access port.

```
<target-application-entity>:=<tae>
<tae>:=[<access-point>]<application>

<access-point>:={
   <physical-access-point>
   |<virtual-access-point>
   |<physical-access-point>[<virtual-access-point>]
   |<group-access-points>
}

<physical-access-point>:=<pm>
<pm>:=<machine-address>[:<access-port>]

<virtual-access-point>:='['<vm>']'
<vm>:=<machine-address>[:<access-port>]

<group-access-points>:=<group>[:<access-port>]

<application>:=<host-execution-frame><application-entity>
```

Figure 1: TAE - Target Application Entity address

The machine is addressed by the <**machine-address**>, which represents physical and virtual machines as well as login-sessions provided by the HOSTs plugin. The specific plugins may suppport a subset of the full scope, but the attributes **ID** and **LABEL** are mandatory in any case. The **ID** attribuet is here either a persistent identifier, in case of a VM a configuration file, or a dynamic identifier in case of the HOSTs plugin, e.g. for VNC the DISPLAY number excluding the port-offset. Whereas it is defined for X11 as the PID.

```
<machine-address>:=
  (
     [(ID|I|PATHNAME|PNAME|P):<mconf-filename-path>][,]
     |
     [(ID|I):<id>][,]
  )
  [(BASEPATH|BASE|B):<base-path>[\%<basepath>]{0,n}
  [(LABEL|L):<label>][,]
  [(FILENAME|FNAME|F):<mconf-filename>][,]
  [(UUID|U):<uuid>][,]
  [(MAC|M):<MAC-address>][,]
  [(TCP|T):<TCP/IP-address>][,]
```

Figure 2: Machine-Address

The GROUPS objects are a concatination of <machine-addresses> and nested GROUPS including specific context options.

```
<group-address>:= (
           [ <machine-addresses>['(' <machine-options> ')']{0,n}]
           [ <group-address>['('     <group-options>   ')']{0,n}]
)['('<group-options>')']
```

Figure 3: Group-Address

The <**DISPLAYext**> addresses a network display, where the full bath includes the <**target-application-**

entity>, thus providing for various addressing schemas including application gateways.

```
<DISPLAYext>:=<target-display-entity>

<target-display-entity>:=<tde>
<tde>:=<tae>:<local-display-entity>
<local-display-entity>:=<lde>
<lde>:=(<display>|<label>)[.<screen>]
```

Figure 4: TDE - Target Display Entity address

The given general syntaxes lead to the following applied syntaxes with the slightly variation of assigned keywords.

```
<target-application-entity>:=<tae>
<tae>:=[<access-point>]<application>

<access-point>:=<physical-access-point>[<virtual-access-point>]

<physical-access-point>:=<pm>
<pm>:=<machine-address>[:<access-port>]

<virtual-access-point>:='['<vm>']'
<vm>:=<machine-address>[:<access-port>]

<application>:=<host-execution-frame><application-entity>
```

Figure 5: TAE - Target Application Entity address

The above minor variations take into account some common implementation aspects.

<**access-point**>:=<**physical-access-point**>[<**virtual-access-point**>] The complete path to the execution environment.

<**access-port**> The port to be used on the access-point.

<**application**>:=<**host-execution-frame**><**application-entity**> The application itself, which has to be frequently used in combination with a given service as runtime environment.

<**application-entity**> The executable target entity of the addresses application, which could be an ordinary shell script to be executed by a starter instance, or an selfcontained executable, which operates standalone within the containing entity. E.g. this could be a shared object or an executable.

The following extends the DISPLAY for seamless usage within ctys. So redirections of entities to any PM, VM of VNC session supporting an active Xserver will be supported. The only restrictions apply, are the hard-coded rejection of unencrypted connections crossing machine-borders.

```
TDE - Target Display Entity address
===================================

<DISPLAYext>:=<target-display-entity>

<target-display-entity>:=<tde>
<tde>:=<tae>:<lde>
```

(basepath|base|b):<**base-path**>**1,n** Basepath could be a list of prefix-paths for usage by UNIX "find" command. When omitted, the current working directory of execution is used by default.

(filename|fname|f):<mconf-filename> A relative pathname, with a relative path-prefix to be used for down-tree-searches within the given list of <base-path>.

So far the theory. The actual behaviour is slightly different, as though as a simple pattern match against a full absolute pathname is performed. Thus also parts of the fullpathname may match, which could be an "inner part". This is perfectly all right, as far as the match leads to unique results.

More to say, it is a feature. Though a common standardname, where the containing directory of a VM has the same name as the file of the contained VM could be written less redundant, when just dropping the repetitive trailing part of the name.

<host-execution-frame> The starter entity of addressed container, which frequently supports a sub-command-call or the interactive dialog-access of users to the target system.

(id|i):<mconf-filename-path> The <id> is used for a variety of tasks just as a neutral matching-pattern of bytes, an in some cases as a uniqe VM identifier within the scope of single machine. The semantics of the data is handled holomporphic due to the variety of utilized subsystems, representing various identifiers with different semantics. Thus the ID is defined to be an abstract sequence of bytes to be passed to a specific application a.k.a. plugin, which is aware of it's actual nature.

The advantage of this is the possibility of a unified handling of IDs for subsystems such as VNC, Xen, QEMU and VMware. Where it spans semantics from beeing a DISPLAY number and offset of a base-port, to a configuration file-path for a DomU-IDs, or a PID of a "master process".

This eases the implementation of cross-over function like LIST, because otherwise e.g. appropriate access-rights to the file are required, which is normally located in a protected subdirectory. These has to be permitted, even though it might not be required by the actual performed function.

(LABEL|L):<label> `<label>={[a-zA-Z-_0-9]{1,n} (n<30, if possible)}`
User defined alias, which should be unique. Could be used for any addressing means.
.

(MAC|M):<MAC-address>
The MAC address, which has basically similar semantically meaning due to uniqueness as the UUID.

Within the scope of ctys, it is widely assumed - even though not really prerequired - that the UUIDs and MAC-Addresses are manual assigned statically, this could be algorithmic too. The dynamic assignment by VMs would lead to partial difficulties when static caches are used.

<mconf-filename> The filename of the configuration file without the path-prefix.

<mconf-filename-path> The complete filepathname of the configuration file.

<mconf-path> The pathname prefix of the configuration file.

(PATHNAME|PNAME|P):<mconf-path> When a VM has to be started, the <pathname> to it's configuration file has to be known. Therefore the <pathname> is defined. The pathname is the full qualified name within the callers namescope. SO in case of UNIX it requires a leading '/'.

<physical-access-point>:=<machine-address>[:<access-port>] The physical termination point as the lowest element of the execution stack. This is the first entity to be contacted from the caller's site, normally by simple network access.

<target-application-entity> The full path of the stacked execution stack, addressing the execution path from the caller's machine to the terminating entity to be executed. This particularly includes any involved PM, and VM, as well as the final executable. Thus the full scope of actions to be performed in order to start the "On-The-Top" executable is contained.

(TCP|T):<tcp/ip-address> The TCP/IP address is assumed by ctys to assigned in fixed relation to a unique MAC-Address.

(UUID|U):<uuid> The well known UUID, which should be unique. But might not, at least due to inline backups, sharing same UUID as the original. Therefore the parameter FIRST, LAST, ALL is supported, due to the fact, that backup files frequently will be assigned a name extension, which places them in alphabetical search-order behind the original. So, when using UUID as unique identifier, a backup will be ignored when FIRST is used.

Anyhow, cross-over ambiguity for different VMs has to be managed by the user.

<virtual-access-point>:=<machine-address>[:<access-port>] The virtual termination point as an element of the execution stack. The stack-level is at least one above the bottom This stack element could be accessed either by it's operating hypervisor, or by native access to the hosted OS.

4 Stack Addresses

The stack address is a logical collection of VMs, including an eventually basic founding PM, which are in a vertical dependency. The dependency results from the inherent nested physical execution dependency of each upper-peer from it's close underlying peer. Therefore the stack addresses are syntactically close to **GROUPS** with additional specific constraints, controlling execution dependency and locality. Particularly the addressing of a VM within an upper layer of a stack could be smartly described by several means of existing path addresses schemas. Within the UnifiedSessionsManager a canonical form is defined for internal processing(SECTIONs:StacksAsVerticalSubgroups), which is available at the user interface too. Additional specific syntactical views are implemented in order to ease an intuitive usage for daily business. The following section depicts a formal meta-syntax as a preview of the final ASN.1 based definition. A stack address has the sytax as depicted in Figure˜6.

```
<stack-address>:=<access-point-list>

<access-point-list>:=[
       <physical-access-point>
       |<virtual-access-point-list>
       ]

<virtual-access-point-list>:=
       '['<virtual-access-point>']'['('<context-opts>')']
       [<virtual-access-point-list>]
```

Figure 6: Stack-Address

A stack can basically contain wildcards and simple regexpr for the various levels, groups of entities within one level could be provided basically to. And of course any MACRO based string-replacement is applicable. But for the following reasons the following features are shifted to a later version:

Wildcards: An erroneous user-provided wildcard could easily expnad to several hundred VMs, which might be not the original intention. Even more worst, due to the detached background operation on remote machines, this can not easily be stopped, almost just ba reboot of the execution target. Which, yes, might take some time, due to the booting VMs.

Level-Groups/Sets: Due to several highe priorities this version supports explicitly addressed entries only.

5 Groups Resolution

Groups are valid replacements of any addressed object, such as a HOST. Groups can contain in addition to a simple set of hostnames a list of entities with context specific parameters and include other groups in a nested manner. Each set of superposed options is permutated with the new included set.

The resolution of group names is processed by a search path algorithm based on the variable
,

CTYS_GROUPS_PATH , which has the same syntax as the PATH variable. The search algorithm is a first-wins filename match of a preconfigured set. Nested includes are resolved with a first-win algorithm beginning at the current position.

In addition to simple names a relative pathname for a group file could be used. This allows for example the definition of arbitrary categories, such as server, client, desktop, db, and scan. Here are some examples for free definitions of categories based on simple subdirectories to search paths. The level of structuring into subdirectories is not limited.

server/* A list of single servers with stored specific call parameters. Server is used here as a synonym for a backend process. Which could be either a PM or a VM, the characteristics is the inclusion of the backend process only.

client/* A list of single clients with stored specific call parameters. This is meant as the user front end only, which could be a CONNECTIONFORWARDING. The user can define this category also as a complete client machine including the backend and frontend, which is a complete client for a service.

desktop/* A composition of combined clients and servers for specific tasks. This could be specific desktops for office-applications, systems administration, software-development, industrial applications, test environments. Either new entries could be created, or existing groups could be combined by inclusion.

db/* Multiple sets of lists of targets to be scanned into specific caching databases. This could be used for a working set as well as for different views of sets of machines.

scan/* A list of items to be scanned by tools for access validation and check of operational mode. Therefore this entities should contain basic parameters onyl, such as machine specific remote access permissions type.

REMARK: The group feature requires a configured SSO, either by SSH-Keys of Kerberos when the parallel or async mode is choosen, which is the default mode. This is required due to the parallel-intermixed password request, which fails frequently.

For additional information on groups refer to "GroupTargets" and "ctys-groups" .

6 Groups of Machines

The GROUPS objects are a concatination of <machine-addresses> and nested GROUPS including specific context options. The end of the command with it's specific option should be marked by the common standard with a double column '–'.

```
ctys -a <action> -- '(<glob-opts>)' <group>'('<group-opts>')'

=> The expansion of contained hosts results to:

   ...
   <host0>'(<host-opts> <glob-opts> <group-opts>')'
   <host1>'(<host-opts> <glob-opts> <group-opts>')'
   ...

=> The expansion of contained nested groups results to:

   ...
   <group-member0>'(<glob-opts>)'('<group-opts>')'
   <group-member1>'(<glob-opts>)'('<group-opts>')'
   ...
```

Figure 7: Groups of Stack-Addresses

The context options are applied succesively, thus are 'no-real-context' options, much more a successive superposition. More worst, the GROUP is a set, thus the members of a group are reordered for display and execution purposes frequently. So the context options are - in most practical cases - a required minimum for the attached entity.

7 Groups of Stack Addresses

The usage of stacked addresses is supported by the GROUPS objects for any entry, where an address is required, except for cases only applicable to PMs, e.g. WoL. The usage of stacked addresses within groups is supported too.

Therefore the behaviour for global remote options on ctys-CLI is to chain the option with any entity within the group, such as for the single PM case in Figure~??.

```
ctys -a <action> -- '(<glob-opts>)' <group>'('<group-opts>')'

=> group expansion results to:

   ...
   <group-member0>'(<glob-opts>)'('<group-opts>')'
   <group-member1>'(<glob-opts>)'('<group-opts>')'
   ...

=> host expansion result to:

   ...
   <group-member0>'(<glob-opts> <group-opts>')'
   <group-member1>'(<glob-opts> <group-opts>')'
   ...
```

Figure 8: Groups of Stack-Addresses

This behaviour of "chaining options" results due it's intended mapping to the internal canonical form before expanding it's options, to the permutation of the <group-options> to each member of the group. The same is true for the special group VMSTACK

that the global and context options are in case of groups just set for the last - topmost - stack element Figure~**??**.

```
  <group-member0>'(<glob-opts>)(<group-opts>)'

=> group expansion results to:

  '[<vm0>][vm1][vm2](<glob-opts>)(<group-opts>)'

=> host + stack expansion result to:

   level-0: <vm0>
   level-1: <vm0>'['<vm1>']'
   level-2: <vm0>'['<vm1>']''['vm2']''('<glob-opts>)'('<group-opts>')'
```

Figure 9: Groups member option expansion

When entries within the stack require specific context-options, these has to be set explicitly within the group definition, or the stack has to be operated step-by-step. This behaviour is planned to be expanded within one of the next versions.

8 SEE ALSO

UserManual , *HowTo*

ctys(1) , *ctys-vhost(1)*

9 AUTHOR

Maintenance:	<acue_sf1@sourceforge.net>
Homepage:	<http://www.UnifiedSessionsManager.org>
Sourceforge.net:	<http://sourceforge.net/projects/ctys>
Berlios.de:	<http://ctys.berlios.de>
Commercial:	<http://www.i4p.com>

10 COPYRIGHT