EXTERNAL

How to use the SAPControl Web Service Interface

Table of contents

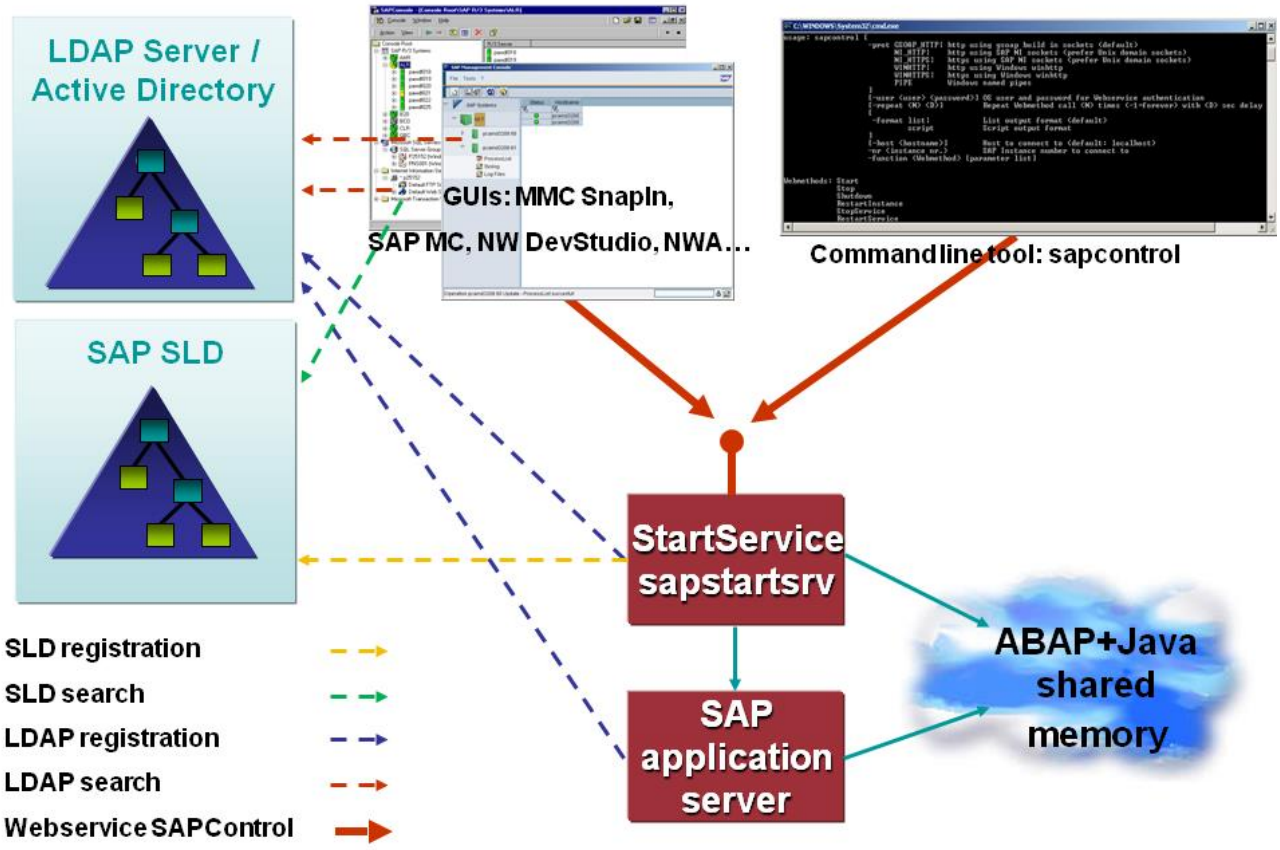
Introduction

The SAP Start Service (sapstartsrv) provides basic management services for systems and instances and single server processes. Services include starting and stopping, monitoring the current run-time state, reading logs, traces and configuration files, executing commands and retrieving other technology-specific information, like network access points, active sessions, thread list etc. They are exposed by a SOAP Web service interface named “SAPControl”. This paper describes how to use this Web service interface.

Since release 7.00 sapstartsrv is available for all SAP supported platforms and used to start and stop the SAP instance. On Windows the ISAPControl DCOM interface is still supported up to release 7.48 but will be deprecated. As of release 7.49 the ISAPControl DCOM interface is removed from all components. The new SAPControl Web service interface should be used instead. It offers significantly improved functionality with respect to AS Java monitoring as well as platform independent monitoring. The Web service interface can be used from any Web service enabled client that can handle the doc/literal communication style (e.g. Java, ABAP, .NET, gSOAP ...).

On Windows each SAP instance is started by a specific NT service named SAP<SID>_<NR>. Up to release 4.5A the service was implemented in sapntstartb.exe, which offers a simple proprietary interface via named pipe communication. SAP tools like sservmgr.exe, sapstart.exe, sapsrvkill.exe or sapntwaitforhalt.exe used this named pipe interface to start or stop the SAP system. Since release 4.5B the service has been replaced by sapstartsrv.exe.

The following figure shows the architecture of the components involved.



Concerning the GUI components, the overview is not complete. In release 7.1 there is also the NetWeaver Administrator and the Developer Studio acting as frontend communicating with the SAP Start Service.

Web Service Interface

The SAP Start Service offers its Web service interface on port sapctrl<NR> (HTTP) and sapctrls<NR> (HTTPS) where "<NR>" corresponds to the SAP instance number (00...98). If the ports are not defined in etc/services, the default values 5<NR>13 (HTTP) and 5<NR>14 (HTTPS) are used. HTTPS is only available if SAP standard SSL Software "Secude" and the required certificates are installed. Sapstartsrv uses the same certificates (SAPSSLS.pse, SAPSSLC.pse) as the other parts of the instance (icman, message server, ...). Starting with release 7.42 sapstartsrv automatically setups a system PKI and generates a certificate (sap_system_pki_instance.pse) for each instance. This way HTTPS is enabled by default. The system PKI is intended to be used for secure system internal communication. A HTTPS client can switch from SAPSSLS.pse to sap_system_pki_instance.pse usage via Server Name Indication (SNI) using special SNI hostname "sap_internal_communication". Instance number 99 is reserved for SAPHostControl, which is installed once per host to perform host instead of instance specific tasks, e.g. adaptive computing or saposcol. It uses IANA registered ports 1128 / saphostctrl and 1129 / saphostctrls. SAPHostControl is not described in this paper.

On Unix a trusted local connect via Unix domain sockets (SAP NI standard naming /tmp/.sapstream<port-nr>) is also possible. On Windows a trusted connect via named pipe \\<hostname>\pipe\sapcontrol_<NR> is possible. There is no authentication check (see below) for trusted connects. This enables a client to use the protected methods (see below) without any additional authentication in a secure way.

If LDAP and/or SLD registration are configured (profile parameter ldap/autoregister=1 / slddest.cfg present in DIR_GLOBAL directory), the service registers itself during service startup in an LDAP directory or SAP System Landscape Directory (SLD). Especially it will register the necessary information to bind to the old DCOM and new HTTP/HTTPS interface. The LDAP registration will use the SAP-R3-ServiceConnectionPoint class with CN=ControlService, CN=ControlService_HTTP and CN=ControlService_HTTPS for registration. The SLD registration will use the SAP_BCControlInstance class with name=<SID>.HostName.<Host>.InstanceNumber.<NR>. Please refer to the "SAP System Information in Directory Services" document on the SAP Service Marketplace and the SLD documentation for further details. A trace of the registration process will be written in the working directory of the SAP instance (dev_ldaps, dev_sldregs).

In releases >= 738 the access to almost all methods of the Web service is protected by default (service/protectedwebmethods=SDEFAULT). In early releases only access to critical methods altering the instance state was protected by default (service/protectedwebmethods=DEFAULT). The list of protected methods can be changed by using the profile parameter "service/protectedwebmethods", either a blank separated list of webmethods or a one of the 4 default sets optionally followed by webmethods to be added (+) or removed from the given default set: [ALL|SDEFAULT|DEFAULT|NONE] +|-<method1> +|-<method2>... +|-<methodN>. To use these protected methods one

- has to provide a valid OS user and password via HTTP basic authentication encoded as UTF8, authorized by service/admin_users, service/admin_groups profile parameters or sapstartsrv executable file permissions (Windows: execute permission, Unix: write permission) or
- connect via https with a valid client certificate authorized by service/sso_admin_user_<N> profile parameter or

- connect via https with SNI hostname “sap_internal_communication” using a system PKI client certificate (release >= 7.42) or
- request a temporary local logon ticket using RequestLogonFile webmethod and use user “{2D4A6FB8-37F1-43d7-88BE-AD279C89DCD7}” with provided ticket as password (only available for local connections), authorized by service/admin_users, service/admin_groups profile parameters or sapstartsrv executable file permissions (Windows: execute permission, Unix: write permission) or
- use a trusted connect via Windows named pipe or Unix domain socket (only available for local connections)

If authentication or authorization check fails, the request will fail with “Invalid Credentials" or "Permission denied" fault string. Missing credentials when accessing a critical method will result in HTTP error 401. Windows users may be given in format <domain>\<user> or <user>@<domain>. On Unix sapstartsrv will ignore the domain user part. On Windows sapstartsrv will try any trusted domain if no domain is given.

The Web service interface is implemented in C++ by using gSOAP 2.7. Doc/literal encoding style is used. The WSDL interface definition can be obtained directly from the Web service using http://<host>:<port>/?wsdl. It can be used to generate a client proxy in Web service enabled programming environments, like gSOAP, Axis, Microsoft .NET, SAP ABAP, SAP J2EE.

Most methods use similar in and out parameters. Some methods like “Shutdown” require no parameters at all. Some others like “SendSignal” require input parameters. Most of the methods return information in a table like data structure (e.g. “GetProcessList”). The interface is using SOAP exception and HTTP error code for error handling. Below you will find the currently implemented methods in the format gSOAP uses for Web service definition (without “SAPControl__” namespace prefix). The last parameter of each method defines the SOAP response (output parameter). All other parameters define input parameters for a method.

General Methods

```
Start(struct StartResponse{} *out)
Stop( int                    softtimeout=0,
      struct StopResponse{}  *out)
RestartInstance(  int                                softtimeout=0,
                  struct RestartInstanceResponse{}   *out);
Shutdown(struct ShutdownResponse{} *out)
```

Use these functions to start, stop or restart a SAP instance. **Start** triggers an instance start. **Stop** triggers an instance stop. **softtimeout** specifies a timeout in sec for a soft shutdown via SIGQUIT, if the timeout expires a hard shutdown is used. **Shutdown** triggers a soft shutdown via SIGQUIT. **RestartInstance** triggers an instance restart. All functions work asynchronously, which means they trigger the operation and return immediately.

```
StartBypassHA(struct StartResponse{} *out)
StopBypassHA(     int                    softtimeout=0,
                  struct StopResponse{}  *out)
```

These functions are indented to be used internally by third party HA solutions. They behave similar to **Start** and **Stop** but do not notify the HA solution in case the instance is controlled by a HA setup. HA products can use it to start / stop the instance from within the HA solution.

```
InstanceStart(    char *host 1:1,
                  int nr,
                  struct InstanceStartResponse{} *out)

InstanceStop(     char *host 1:1,
                  int nr,
                  int softtimeout=0,
                  struct InstanceStopResponse{} *out)

Bootstrap(  char *host,
            int nr=-1,
            struct BootstrapResponse{} *out)
```

Use this function to start, stop or bootstrap an instance given by hostname (**host)** and instance number (**nr)** and connecting to another sapstartsrv of the same system. When connecting to a local sapstartsrv of the same system via trusted connect this can be used start or stop remote instances without explicit user/password authentication. **Bootstrap** triggers sapcpe kernel replication, shared memory cleanup and sapstartsrv restart without actually starting the instance. If **host** and **nr** are not specified, the actual sapstartsrv triggers a bootstrap for its own instance.

```
RestartService(struct RestartServiceResponse{} *out)
StopService(struct StopServiceResponse{} *out)
```

Use these functions to restart of stop the sapstartsrv Web service. However, once the Web service is stopped you have to start sapstartsrv before using the Web service interface again.

```
ParameterValue(   char  *parameter,
                  char  **value);
```

Returns a SAP profile parameter **value** for a given profile **parameter**. If the given profile **parameter** is empty, it returns a string with all known parameter value pairs separated by newline.

```
GetProcessList(ArrayOfOSProcess *process);
```

```
      enum STATE_COLOR
      {
            SAPControl_GRAY   = 1,
            SAPControl_GREEN  = 2,
            SAPControl_YELLOW = 3,
            SAPControl_RED    = 4
      };

      class OSProcess
      {
            char *name;
            char *description;
```

```
        enum STATE_COLOR dispstatus;
        char *textstatus;
        char *starttime;
        char *elapsedtime;
        int pid;
};

class ArrayOfOSProcess
{
        OSProcess *__ptr;
        int __size;
};
```

Returns a list of all processes directly started by the sapstartsrv Web service according to the SAP start profile.

```
StartSystem(    enum StartStopOption          options,
                char                          *prioritylevel,
                int                           waittimeout,
                struct StartSystemResponse{}  *out);

StopSystem(     enum StartStopOption          options,
                char                          *prioritylevel,
                int                           softtimeout,
                int                           waittimeout,
                struct StopSystemResponse{}   *out);

RestartSystem(  enum StartStopOption                options,
                char                                *prioritylevel,
                int                                 softtimeout,
                int                                 waittimeout,
                struct RestartSystemResponse{}      *out)

enum StartStopOption
{
        SAPControl_ALL_INSTANCES    = 0,
        SAPControl_SCS_INSTANCES    = 1,
        SAPControl_DIALOG_INSTANCES = 2,
        SAPControl_ABAP_INSTANCES   = 3,
        SAPControl_J2EE_INSTANCES   = 4,
        SAPControl_PRIORITY_LEVEL   = 5,
        SAPControl_TREX_INSTANCES   = 6,
        SAPControl_ENQREP_INSTANCES = 7
};
```

Use these functions to start, stop or restart a complete SAP system or parts of it. **StartSystem** triggers a system start. **StopSystem** triggers a system stop. **RestartSystem** triggers a system restart. **options** defines which

instances to start/stop/restart. If **SAPControl_PRIORITY_LEVEL** is used, **prioritylevel** defines up/down to which instance priority level instances should be started/stopped. **waittimeout** specifies a timeout in sec to wait for an instance to start/stop. If the timeout expires during a start operation remaining instances with a higher instance priority are not started, since they rely on the other instances to be running. If the timeout expires during a stop operation, the operation will continue stopping the remaining instances. **softimeout** specifies a timeout in sec for a soft shutdown via SIGQUIT, if the timeout expires a hard shutdown is used for the remaining instances. All functions work asynchronously just triggering the operation and returning immediately.
By default, instance priority is calculated automatically from instance type, e.g. “0.3”: HDB, “0.5”: ENQREP, “1”: SCS or ASCS, “1.5”: TREX, “2”: ABAP with enqueue work process or messageserver, “3”: Other). To overwrite or define new priorities profile parameter service/startpriority can be used. Instances are started from lowest to highest priority (lexicographical sorted) and stopped vice versa. All instances with the same priority level are started/stopped in parallel. Once all instances of a level are fully started/stopped the system start/stop continues with the next level. **GetSystemInstanceList** provides a list of all instances of the system with its assigned priority level.

```
GetStartProfile(class GetStartProfileResponse
                  {char *name; ArrayOfString lines;} *file)

      class ArrayOfString
      {
            char **__ptr;
            int __size;
      };
```

Returns start profile name and its content.

```
GetTraceFile(class GetTraceFileResponse
                  {char *name; ArrayOfString lines;} *file)

      class ArrayOfString
      {
            char **__ptr;
            int __size;
      };
```

Returns the sapstartsrv Web service trace file name and its content.

```
ListDeveloperTraces(ArrayOfDirEntry *file)

      class DirEntry
      {
            char *filename;
            unsigned int size;
            char *modtime;
      };
```

```
class ArrayOfDirEntry
{
      DirEntry *__ptr;
      int __size;
};
```

Returns a list of all instance trace files in DIR_HOME (**superseded by ListLogFiles)**. A trace file can be read by using **ReadDeveloperTrace.**

```
ReadDeveloperTrace(     char                                    *filename,
                        int                                     size,
                        class ReadDeveloperTraceResponse
                              {char *name; ArrayOfString lines;}  *file)

class ArrayOfString
{
      char **__ptr;
      int __size;
};
```

Returns the content of a given trace file. Use **size=0** to read the entire file, **size>0** to read the first size bytes, **size<0** to read the last size bytes (**superseded by ReadLogFile)**. **filename** must match with one of the trace files returned by **ReadDeveloperTrace.**

```
GetEnvironment(ArrayOfString *env)

class ArrayOfString
{
      char **__ptr;
      int __size;
};
```

Returns the process environment as an array of parameter/value pair strings.

```
GetAlertTree(ArrayOfAlertNode *tree)

enum STATE_COLOR
{
      SAPControl_GRAY   = 1,
      SAPControl_GREEN  = 2,
      SAPControl_YELLOW = 3,
      SAPControl_RED    = 4
};

enum VISIBLE_LEVEL
```

```
{
        SAPControl_UNKNOWN      = 0,
        SAPControl_OPERATOR     = 1,
        SAPControl_EXPERT = 2,
        SAPControl_DEVELOPER    = 3
};

class AlertNode
{
        char *name;
        int parent;
        enum STATE_COLOR ActualValue;
        char *description;
        char *Time;
        char *AnalyseTool;
        enum VISIBLE_LEVEL VisibleLevel;
        enum STATE_COLOR HighAlertValue;
        char *AlDescription;
        char *AlTime;
        char *Tid;
};

class ArrayOfAlertNode
{
        AlertNode *__ptr;
        int __size;
};
```

Returns CCMS Alert tree as an array. The parent-child node relationship is encoded via the parent index of each node. (similar to rz20 transaction).

```
GetAlerts(  char              *RootTid,
            GetAlertsResponse *alertlist)

        enum STATE_COLOR
        {
                SAPControl_GRAY   = 1,
                SAPControl_GREEN  = 2,
                SAPControl_YELLOW = 3,
                SAPControl_RED    = 4
        };

        class Alert
        {
                char *Object;
                char *Attribute;
```

```
        enum STATE_COLOR Value;
        char *Description;
        char *Time;
        char *Tid;
        char *Aid;
};

class ArrayOfAlert
{
        Alert *__ptr;
        int __size;
};

class GetAlertsResponse
{
        char *RootTidName;
        ArrayOfAlert alert;
};
```

Returns a list of all CCMS alerts for a given node and its child nodes.

```
SendSignal( int                          pid,
            char                         *signal,
            struct SendSignalResponse{}  *out)
```

Sends a given OS **signal** to a process specified by its **pid**. The signal can be given by name (HUP, INT, QUIT, ILL, TRAP, ABRT, IOT, BUS, FPE, KILL, SIG, USR1, SEGV, USR2, SIG, PIPE, ALRM, TERM, STKFLT, CHLD, CONT, STOP, TSTP) or number. OS signals are platform dependet, some signals are not supported by all platforms.

```
GetVersionInfo(ArrayOfInstanceVersionInfo *version)

class InstanceVersionInfo
{
    char *Filename;
    char *VersionInfo;
    char *Time;
};

class ArrayOfInstanceVersionInfo
{
        InstanceVersionInfo *__ptr;
        int __size;
};
```

Returns a list of version information for the most important files of the instance.

```
GetQueueStatistic(ArrayOfTaskHandlerQueue *queue)

      class TaskHandlerQueue
      {
            char *Typ;
            int Now;
            int High;
            int Max;
            int Writes;
            int Reads;
      };

      class ArrayOfTaskHandlerQueue
      {
            TaskHandlerQueue *__ptr;
            int __size;
      };
```

Returns a list of queue information of ABAP work processes and icm.

```
OSExecute(  char                                    *command,
            int                                     async,
            int                                     timeout,
            char                                    *protocolfile,
            class OSExecuteResponse
                  {     int               exitcode;
                        int               pid;
                        ArrayOfString      lines;}    *result)

      class ArrayOfString
      {
            char **__ptr;
            int __size;
      };
```

Executes an external OS **command.** Use **async=0** to execute the command synchronously. The Web service method returns when the command has finished or the **timeout** (specified in sec, 0=infinite) is reached. If the timeout is reached the process will be terminated. Use **async=1** to execute the command asynchronous. The Web service method will return immediately. stdout/stderr of the command can be redirected to a **protocolfile.** Use **protocolfile=**"" for getting the result in the **lines** output parameter for synchronous commands or redirecting it to the OS NULL device for asynchronous commands. Protocol files will not be deleted automatically by sapstartsrv.

GetInstanceProperties(ArrayOfInstanceProperties *properties)

```
class InstanceProperty
{
      char  *property;
      char  *propertytype;
      char  *value;
};


class ArrayOfInstanceProperties
{
      InstanceProperty *__ptr;
      int __size;
};
```

Returns a list of available instance features and which Web service methods are supported to get the information. **GetInstanceProperties** provides some meta information about the instance, which allows a client to display only information relevant for the actual instance type and version. It also enables a client to work with multiple versions of the Web service interface. Currently 3 **propertytype** values are defined.
"**NodeWebmethod**" is used for nodes which provide information via Web service methods. A client should use **property** as node name for displaying the information and use any of the Web service methods defined by **value**. A client should use the leftmost method in the methods list it is supporting, e.g.:

```
property="J2EE Caches"
propertytype="NodeWebmethod"
value="J2EEGetCacheStatistic2,J2EEGetCacheStatistic"
```

The client should display the information as "J2EE Caches" and use webmethod **J2EEGetCacheStatistic2** to get the information. Older clients not aware of **J2EEGetCacheStatistic2** can still use **J2EEGetCacheStatistic** to get most of the information.

"**NodeURL**" is used for nodes which provide information via a generic URL, e.g.:

```
property="ICM"
propertytype="NodeURL"
value=HTTP://WDFD00155758A:56000/sap/admin
```

The client should display the information as "ICM" and use HTTP://WDFD00155758A:56000/sap/admin to display additional information about the node.

"**Attribute**" is used to provide additional information about the instance, e.g.:

```
property="StartPriority"
propertytype= "Attribute"
value= "3"
```

```
property="Protected Webmethods"
propertytype= "Attribute"
value="Start,Stop,Shutdown,StartSystem,StopSystem,StopService,J2EEControlProcess
,SendSignal,OSExecute"

ReadLogFile(char                                                    *filename,
            char                                                    *filter,
            char                                                    *language,
            int                                                     maxentries,
            char                                                    *statecookie,
            class ReadLogFileResponse
                  {     char              *format;
                        char              *startcookie;
                        char              *endcookie;
                        ArrayOfString     fields;}    *log)

      class ArrayOfString
      {
            char **__ptr;
            int __size;
      };
```

Returns the content of a given log file defined by **filename**. **filename** must match with one of the log files returned by **ListLogFiles. ReadLogFile** can read various file types like plain text, ABAP Syslog or J2EE log files.
filter can be used to limit the result to certain columns and only matching entries:

filter="": Read all entries and columns.
filter="<Column1>#<Column2>#...#<ColumnN>": Read all entries but only specified columns, e.g.:
filter="Time#Severity#Text"
filter="[CASEIGNORE:]<Column1><PatternSet1>#<Column2><PatternSet2>#...#<ColumnN><PatternSetN>"
Use "CASEIGNORE:" at filter beginning to define a case insensitive pattern matching
PatternSet syntax: <Pattern1>|...|<PatternN>
Pattern syntax:

- "=...." lexicographical equal, use "*,?" for wildcard matching
- "!...." lexicographical non equal, use "*,?" for wildcard matching
- "<...." lexicographical smaller
- "(...." semantical smaller
- ">...." lexicographical bigger
- ")...." semantical bigger
- "[]<Begin>,<End>" lexicographical between <Begin> and <End
- "][<Begin>,<End>" lexicographical outside <Begin> and <End> interval

e.g.: filter="CASEIGNORE:Time#Severity)Info#Text=*timeout*|=*null*"

language is reserved for future usage.

statecookie specifies the starting position to read from. Use **statecookie=””** to read from the beginning, **statecookie=”EOF”** to read from the end or **statecookie=<endcookie>** / **statecookie=<startcookie>** to continue reading from a previous call returning endcookie / startcookie.
Use **maxentries** to specify an upper limit of returned entries (0=all) and reading direction (>0: forward,<0: backward).

On return **format** contains a “#” separated string containing the column names, e.g.:

```
J2EE log file:
format="Version#Guid#Time#SourceName#Application#Location#User#Session#Transacti
on#DSRComponent#DSRUser#DSRTransaction#ThreadName#GroupId#GroupLevel#GroupIndent
#Severity#Relatives#MsgType#MsgCode#ResourceBundle#Text”

Plain text file:
format="Line”

ABAP Syslog:
format="Severity#Time#Typ#Client#User#Tcode#MNo#Pid#Terminal#Program#Session#Tex
t”
```

startcookie / **endcookie** identify file start and end position of the response and can be used additional calls of **ReadLogFile** to continue reading.
fields contains the log file entries. A log entry corresponds to a single string. The columns of an entry are separated by Tabs matching with the **format** string.
ReadLogFile supercedes ReadDeveloperTracse.

```
ListLogFiles(ArrayOfLogFile *file)

      class LogFile
      {
            char *filename;
            unsigned int size;
            char *modtime;
            char *format;
      };

      class ArrayOfLogFile
      {
            LogFile *__ptr;
            int __size;
      };
```

Returns a list of all instance log files (**supersedes ListDeveloperTraces**). **format** identifies the log file format (“Text”, “J2EE Fileset”, “J2EE Fileset Part”, “SAP Syslog”). A log file can be read by using **ReadLogFile.**

```
AnalyseLogFiles(  char *starttime,
                  char *endtime,
                  int severity_level=2,
                  int maxentries = 10000,
                  class AnalyseLogFilesResponse
                        {
                              char *format;
                              ArrayOfString fields;}   *log)

      class ArrayOfString
      {
            char **__ptr;
            int __size;
      };
```

Scans all log files for a given time period and returns a merged list of all matching log file entries. Time period of interest can be defined by **starttime** and **endtime** (Format: “YYYY MM DD HH:MM:SS”), if not defined the last 10 minutes of the last instance run are used. **severity_level** the log entry severity level to serach for (2=Only errors, 1=Errors and Warnings, 0=All). Use **maxentries** to limit the amount of log entries to return.

```
ConfigureLogFileList(    enum LogFileConfigOperation operation,
                         ArrayOfString *logfiles,
                         struct ConfigureLogFileListResponse{} *out)

      enum LogFileConfigOperation
      {
            SAPControl_SET_LOGFILES = 0,
            SAPControl_ADD_LOGFILES = 1,
            SAPControl_REMOVE_LOGFILES = 2
      }
      class ArrayOfString
      {
            char **__ptr;
            int __size;
      };
```

Configures log files accessible via **ReadLogFile** and **ListLogFiles** for sapstartsrv running in SAPHostControl mode. Log files given by the **logfiles** parameter can be set as the actual log file list, added, or removed from the list depending on the given **operation**. Configuration changes are persisted in service/logfile_XXX profile parameters. **logfiles** can contain filenames, directories, or filename patterns. When specifying a directory, the entire directory tree is accessible. A filename pattern is a directory followed by a filename pattern (using “?” and “*” wildcards). All files in the directory and matching with the filename pattern are accessible.

```
GetLogFileList(ArrayOfString *logfiles)
```

```
class ArrayOfString
{
      char **__ptr;
      int __size;
};
```

Returns a list of configured log files for sapstarstrv running in SAPHostControl mode. All files matching with any entry in the list are accessible via **ReadLogFile** and **ListLogFiles**. The log file list is configured with profile parameters service/logfile_XXX and can be modified using **ConfigureLogFileList**.

GetAccessPointList(ArrayOfAccessPoint *accesspoint)

```
class AccessPoint
{
      char *address;
      int port;
      char *protocol;
      char *processname;
      char *active;
};

class ArrayOfAccessPoint
{
      AccessPoint *__ptr;
      int __size;
};
```

Returns a list of all network access points of the instance.

GetSystemInstanceList(ArrayOfSAPInstance *instance)

```
      enum STATE_COLOR
      {
            SAPControl_GRAY   = 1,
            SAPControl_GREEN  = 2,
            SAPControl_YELLOW = 3,
            SAPControl_RED    = 4
      };

      class SAPInstance
      {
            char *hostname;
            int  instanceNr;
            int  httpPort;
            int  httpsPort;
            char *startPriority;
```

```
        char *features;
        enum STATE_COLOR dispstatus;
    };

    class ArrayOfSAPInstance
    {
        SAPInstance *__ptr;
        int __size;
    };
```

Returns a list of all instances of the SAP system. **features** identifies the instance type (ABAP, J2EE, GATEWAY, MESSAGESERVER, ENQUE, ICMAN, TREX, IGS, ENQREP), e.g.:

Dual-stack dialog instance: “ABAP|J2EE|GATEWAY|ICMAN”
SCS instance: “MESSAGESERVER|ENQUE”

```
AccessCheck(char                          *function,
        struct AccessCheckResponse{}      *out)
```

Check if execution of the specified webmethod is granted.

```
GetProcessParameter(    char                                  *processtype,
                        int                                   pid = -1,
                        ArrayOfProfileParameter           *parameter)

    enum RESTRICTION_TYPE
    {
        SAPControl_RESTRICT_NONE            = 0,
        SAPControl_RESTRICT_INT             = 1,
        SAPControl_RESTRICT_FLOAT           = 2,
        SAPControl_RESTRICT_INTRANGE        = 3,
        SAPControl_RESTRICT_FLOATRANGE      = 4,
        SAPControl_RESTRICT_ENUM            = 5,
        SAPControl_RESTRICT_BOOL            = 6
    };

    class ArrayOfString
    {
        char **__ptr;
        int __size;
    };

    class ParameterRestriction
    {
        enum RESTRICTION_TYPE    type;
        LONG64  *int_min;
```

```
        LONG64  *int_max;
        double  *float_min;
        double  *float_max;
        ArrayOfString *valuemap;
};

class SubProfileParameter
{
        char    *name;
        char    *description;
        char    *unit;
        bool    mandatory;
        ParameterRestriction   restriction;
};

class ArrayOfSubProfileParameter
{
        SubProfileParameter *__ptr;
        int __size;
};

class SAPControl__ProfileParameter
{
        char    *name;
        char    *group;
        char    *description;
        char    *unit;
        ParameterRestriction   restriction;
        ArrayOfSubProfileParameter  *sub_para;
        char    *value;
        char    *dynamic_value;
        ArrayOfString    *values;
        ArrayOfString    *dynamic_values;
};

class ArrayOfProfileParameter
{
        ProfileParameter *__ptr;
        int __size;
};
```

Returns a list of actual profile parameters for a given process. Known **processtype** values are “ICM”, “Web Dispatcher”, “MessageServer”, “Gateway”, “EnqueueServer”, “Dispatcher”. **pid** needs to be set if multiple processes of the same type exist within the instance.

```
SetProcessParameter(    char                                    *processtype,
```

```
                    int                                     pid = -1,
                    ArrayOfSetProfileParameter              parameter,
                    struct SetProcessParameterResponse{}    *out)

class ArrayOfString
{
      char **__ptr;
      int __size;
};

class SetProfileParameter
{
      char    *name;
      char    *value;
      ArrayOfString    *values;
};

class ArrayOfSetProfileParameter
{
      SetProfileParameter *__ptr;
      int __size;
};
```

Sets dynamic Profile Parameters for a given process. Known **processtype** values are “ICM”, “Web Dispatcher”, “MessageServer”, “Gateway”, “EnqueueServer”, “Dispatcher”. **pid** needs to be set if multiple processes of the same type exist within the instance.

```
ActivateSystemDBSuspendMode(  int                                 graceperiod,
                              int                                 downtime,
                    struct ActivateSystemDBSuspendModeResponse{}    *out);
```

Triggers the DB Suspend Mode for each instance. **graceperiod** defines the timeout for canceling active work processes. The unit is seconds. **downtime** provides the expected downtime, used by the ICM. If you want to pass the **downtime** but not the **graceperiod**, set **graceperiod** to -1.
Note: The instances will check $DIR_HOME for the quiesce file.

```
DeactivateSystemDBSuspendMode(
                    struct DeactivateSystemDBSuspendModeResponse{}   *out);
```

Deactivates the DB Suspend Mode for each instance, which was triggered by ActivateSystemDB-SuspendMode.

```
CheckParameter(    ArrayOfString *profile,
                   ArrayOfString *default_profile,
                  ArrayOfParameterCheck *result)
```

```
class ArrayOfString
{
      char **__ptr;
      int __size;
};

enum ParameterMessageState
{
    PARAMETER_INFO = 0,
    PARAMETER_WARNING = 1,
    PARAMETER_ERROR = 2
};

class ParameterMessage
{
    enum ParameterMessageState category;
    char *message;
};

class ArrayOfParameterMessage
{
    ParameterMessage *__ptr;
    int __size;
};

class ParameterCheck
{
    char *name;
    char *kernel_value;
    char *default_value;
    char *profile_value;
    char *default_definition;
    char *kernel_definition;
    char *profile_definition;
    int definition_mask;
    int default_redefinitions;
    int profile_redefinitions;
    ArrayOfParameterMessage messages;
};

class ArrayOfParameterCheck
{
    ParameterCheck *__ptr;
    int __size;
}
```

Returns a list of all profile parameters with definition (unsubstituted value) and value (substituted value) of all 3 parameter levels (kernel, default profile and profile). A list of parameter check findings is associated to each

parameter. By default (**profile** and **default_profile** not set or empty) the actual default profile and instance profile are used for the check. A caller can specify a different default or instance profile content using the **profile** and **default_profile** parameters. **definition_mask** is a bit mask which defines where parameter definitions have been found (1=Kernel, 2=Default Profile, 4=Profile). **default_redefinitions** and **profile_redefinitions** are set if a parameter is set multiple time in default profile or instance profile.

```
CreateSnapshot(    char              *description,
                   char              *datcol_param,
                   int               analyse_severity_level=2,
                   char              *analyse_starttime,
                   char              *analyse_endtime,
                   int               analyze_maxentries = 10000,
                   int               maxentries = -10000,
                   ArrayOfString     logfiles,
                   class CreateSnapshotResponse {char  *filename;} *snapshot)

      class ArrayOfString
      {
            char **__ptr;
            int __size;
      };
```

Creates an instance snapshot and stores it in the system DIR_GLOBAL directory. A snapshot is a ZIP archive containing several webservice responses of the actual instance state. It can later be opened by SAP MMC for offline problem analysis. **description** specifies some text describing the snapshot. If a non-empty string is given by **datcol_param** the J2EE data collector is started during snapshot creation with datacol_param value as commandline option. **analyse_severity_level**, **analyse_starttime**, **analyse_endtime** and **analyze_maxenetries** specify the log file analysis to be included in the snapshot (see **AnalyseLogFiles**). To disable logfile analysis use analyse_severity_level=-1. **maxentries** define the amount of logfile entries to be included (see **ReadlogFile**). **logfiles** defines the logfiles to be included in the snapshot, use “DEFAULT” to include a default set of logfiles. **filename** returns the filename of the created snapshot.

```
ReadSnapshot(      char              *filename,
                   SnapshotZip  *snapshot)

      class SnapshotZip
      {
         unsigned char *__ptr;
         int __size;
      };
```

Reads a snapshot specified by **filename** from the server and returns the binary ZIP archive content.

```
ListSnapshots(     ArrayOfSnapshotInfo      *snapshots)

      class SnapshotInfo
```

```
{
        char *filename;
        LONG64 size;
        char *modtime;
        char *description;
};

class ArrayOfSnapshotInfo
{
        SnapshotInfo *__ptr;
        int __size;
};
```

Returns a list of available snapshots.

```
DeleteSnapshots(  ArrayOfString     *snapshots,
                  class DeleteSnapshotsResponse{} *out)

class ArrayOfString
{
        char **__ptr;
        int __size;
};
```

Deletes a given list of snapshots in the DIR_GLOBAL filesystem.

```
RequestLogonFile( char *user 1:1,
                  char **filename)
```

A local webservice client can use **RequestLogonFile** to request a temporary password for a given **user** in a protected file. The file (returned by **filename**) is protected to be read by the given user only and contains a temporary password. The client can use the temporary password with username "{2D4A6FB8-37F1-43d7-88BE-AD279C89DCD7}" on the same socket connection to call additional webmethods. This enables a local client to authenticate itself with his OS user without actually having to know his own password.

```
GetNetworkId(     char  *service_ip,
                  int   service_port,
                  int   version,
                  class SAPControl__GetNetworkIdResponse{   char *key;} *id)
```

Returns a unique network ID for a network service given by **service_ip** and **service_port**. **version** specifies the algorithm version used to calculate the ID. Since this function doesn't provide any verification, it should only be used get the network ID value (e.g. for requesting a matching license in advance). To verify the network ID (e.g. during license verification **GetSecNetworkId** should be used instead).

```
GetSecNetworkId(  char  *service_ip,
```

```
                int    service_port,
                int    version,
                char   *challenge,
                class GetSecNetworkIdResponse{ char *key;
                                               char *proof;} *id)
```

Returns a unique network ID for a network service given by **service_ip** and **service_port** and a verification **proof** based on **service_ip**, **service_port** caller defined (typically random) **challenge** and **key**. **version** specifies the algorithm version used to calculate the ID. The caller can use the **proof** to verify authenticity of the response. If **challenge** is not given it is read from the Message Server. Since **GetSecNetworkId** additionally uses the client IP address from the actual socket communication the result may differ from GetNetworkId in case **service_ip** is not the real client IP address used to connect to SAPControl Web service.

```
UpdateSystemPKI(  bool force = false,
                  struct UpdateSystemPKIResponse{} *out)
```

Updates the system PKI if necessary. The system PKI consists of a system root PSE and PIN stored in the secure store (located in $(rsec/ssfs_datapath)) and an instance specific PIN protected PSE ($(DIR_INSTANCE)/sec/sap_system_pki_instance.pse) for each instance in the system. It enables secure (SSL) communication between system components. By default, only missing, outdated or bogus parts are updated, to enforce recreation of all components use **force**=true. Internally it uses **UpdateInstancePSE** to trigger creation of instance PSEs on all instances of the system.

```
UpdateInstancePSE(      bool force = false,
                  struct UpdateInstancePSEResponse{} *out)
```

Updates the instance PSE of the system PKI if necessary ($(DIR_INSTANCE)/sec/sap_system_pki_instance.pse). By default, only missing, outdated or bogus PSEs are updated, to enforce recreation of the instance PSE use **force**=true.

```
HACheckConfig(    ArrayOfHACheck *check)
```

```
enum HAVerificationState
{
    SAPControl_HA_SUCCESS = 0,
    SAPControl_HA_WARNING = 1,
    SAPControl_HA_ERROR   = 2
};

enum HACheckCategory
{
    SAPControl_SAP_CONFIGURTAION = 0,
    SAPControl_SAP_STATE         = 1,
    SAPControl_HA_CONFIGURATION  = 2,
    SAPControl_HA_STATE          = 3
```

```
};

class HACheck
{
    enum HAVerificationState state;
    enum HACheckCategory category;
    char *description;
    char *comment;
};

class ArrayOfHACheck
{
    HACheck *__ptr;
    int __size;
};
```

Performs various checks to verify the entire system is configured and operating compliant to the SAP high availability guidelines. The function returns a list of performed tests with check results. For instances hosting SPoFs (**S**ingle **P**oint **o**f **F**ailures) third party HA product specific test results are added (by calling **`HACheckFailoverConfig`** internally).

`HACheckFailoverConfig``(  ArrayOfHACheck *check)`

```
enum HAVerificationState
{
    SAPControl_HA_SUCCESS = 0,
    SAPControl_HA_WARNING = 1,
    SAPControl_HA_ERROR   = 2
};

enum HACheckCategory
{
    SAPControl_SAP_CONFIGURTAION = 0,
    SAPControl_SAP_STATE         = 1,
    SAPControl_HA_CONFIGURATION  = 2,
    SAPControl_HA_STATE          = 3
};

class HACheck
{
    enum HAVerificationState state;
    enum HACheckCategory category;
    char *description;
    char *comment;
};
```

```
class ArrayOfHACheck
{
    HACheck *__ptr;
    int __size;
};
```

Perform third party HA product specific checks of the instance.

```
HAGetFailoverConfig(    class HAGetFailoverConfigResponse{
                            bool HAActive;
                            char *HAProductVersion;
                            char *HASAPInterfaceVersion;
                            char *HADocumentation;
                            char *HAActiveNode;
                            ArrayOfString HANodes;
                        } *config)

class ArrayOfString
{
      char **__ptr;
      int __size;
};
```

Retrieve third party HA product specific information of the instance.

```
HAFailoverToNode( char *node 1:1,
                  struct HAFailoverToNodeResponse{} *out)
```

Trigger a failover of the instance to the given cluster node using the third party HA product interface.

```
HASetMaintenanceMode(   bool mode = true,
                        bool instance_only = false,
                        struct HASetMaintenanceModeResponse{} *out)
```

Enables (**mode=true**) or disables (**mode=false**) the HA product specific maintenance mode for a single instance (**instance_only=true**) or all instances of the system (**instance_only=false**). The webmethod may fail depending on the instance and HA state, it is only guaranteed to work when called on a fully started instance. The webmethods requires usage of a HA product supporting the maintenance mode via the SAP HA interface. The HA maintenance mode is implemented by the HA product and should guarantee that the HA product should not trigger any operation automatically in case of failures (e.g. a failover to another cluster node).

```
HACheckMaintenanceMode( bool instance_only = false,
```

```
                        struct HACheckMaintenanceModeResponse{} *out)
```

Check if the HA product specific maintenance mode for a single instance (**instance_only=true**) or all instances of the system (**instance_only=false**) is available.

```
GetCallstack(     int pid,
                  class GetCallstackResponse{
                        ArrayOfString lines;
                  } *callstack)

      class ArrayOfString
      {
            char **__ptr;
            int __size;
      };
```

Retrieve callstack of af threads of specified process.

```
StorePSE(   char *filename,
            PSEBlob *pseblob,
            int psemode = 0,
            bool overwrite = false,
            struct StorePSEResponse{} *out)

      class PSEBlob
      {
            unsigned char *__ptr;
            int __size;
      };
```

Store PSE given by **pselob** in instance SECUDIR directory with given **filename** using specifed pse mode (currently 0=plain text, 1=encrypted). Use **overwrite=true** to force overwriting an eventually already existing PSE file.

```
DeletePSE(  char *filename,
            struct DeletePSEResponse{} *out)
```

Delete specified PSE (**filename**) in instance SECUDIR directory.

```
CheckPSE(   char *filename,
            PSEBlob *pseblob,
            class CheckPSEResponse{
                  int psemode;
```

```
        } *out)

    class PSEBlob
    {
        unsigned char *__ptr;
        int __size;
    };
```

Check if specified PSE (**filename**) in instance SECUDIR directory is identical with PSE given by **pseblob**. If the two PSEs match the actual pse mode (currently 0=plain text, 1=encrypted) is returned.

```
CreatePSECredential(    char *filename,
                        char *pin,
                        struct CreatePSECredentialResponse{} *out)
```

Store given **pin** for given (pin protected) PSE (**filename**) in cred_v2 file in instance SECUDIR directory to allow usage of pin protected PSEs.

```
ListConfigFiles(  ArrayOfString *configfiles)

    class ArrayOfString
    {
        char **__ptr;
        int __size;
    };
```

Returns list of all configuration files of the instance.

```
ReadConfigFile(   char *filename 1:1,
                  ArrayOfString *lines)

    class ArrayOfString
    {
        char **__ptr;
        int __size;
    };
```

Reads configuration file specified by `filename`.

ABAP Specific Methods

```
ABAPReadSyslog(ArrayOfSyslogEntry *log)

    enum STATE_COLOR
    {
        SAPControl_GRAY    = 1,
```

```
        SAPControl_GREEN  = 2,
        SAPControl_YELLOW = 3,
        SAPControl_RED    = 4
};

class SyslogEntry
{
        char *Time;
        char *Typ;
        char *Client;
        char *User;
        char *Tcode;
        char *MNo;
        char *Text;
        enum STATE_COLOR Severity;
};

class ArrayOfSyslogEntry
{
        SyslogEntry *__ptr;
        int __size;
};
```

Reads the ABAP Syslog and returns it as an array of entries (similar to SM21 transaction).

```
ABAPReadRawSyslog(ArrayOfRawSyslogEntry *log)

class ArrayOfRawSyslogEntry
{
        char **__ptr;
        int __size;
};
```

Reads the SAP ABAP Syslog and returns the raw file content.

```
ABAPGetWPTable(ArrayOfWorkProcess *workprocess)

class WorkProcess
{
        int No;
        char *Typ;
        int Pid;
        char *Status;
        char *Reason;
        char *Start;
        char *Err;
```

```
        char *Sem;
        char *Cpu;
        char *Time;
        char *Program;
        char *Client;
        char *User;
        char *Action;
        char *Table;
};

class ArrayOfWorkProcess
{
        WorkProcess *__ptr;
        int __size;
};
```

Returns a list of the ABAP work processes (similar to SM50 transaction).

```
ABAPGetSystemWPTable(   bool activeonly = false,
                        ArrayOfSystemWorkProcess *workprocess)


class SystemWorkProcess
{
          char *Instance;
          int No;
          char *Typ;
          int Pid;
          char *Status;
          char *Reason;
          char *Start;
          char *Err;
          char *Sem;
          char *Cpu;
          char *Time;
          char *Program;
          char *Client;
          char *User;
          char *Action;
          char *Table;
};

class ArrayOfSystemWorkProcess
{
          SystemWorkProcess *__ptr;
          int __size;
};
```

Returns a list of all ABAP work processes in the system (similar to SM66 transaction). Use **activeonly=true** to only a list of currently active work processes.

```
ABAPAcknowledgeAlerts(  char              *R3Client,
                        char              *R3User,
                        char              *R3Password,
                        ArrayOfString     Aid,
                        ArrayOfInt        *alert)

      class ArrayOfString
      {
            char **__ptr;
            int __size;
      };

      class ArrayOfInt
      {
            int *__ptr;
            int __size;
      };
```

Acknowledge CCMS Alerts in the SAP ABAP system. Requires SAP user credentials and a list of alert ids to acknowledge. Returns a list of success code for each alert (1=success, 0=failure). As of release 7.40 specifying an ABAP user is optional. If not specified or empty sapstartsrv uses a MYSAPSSO2 ticket-based trust to access its ABAP instance via RFC.

```
UpdateSystem(     int softtimeout,
                  int waittimeout,
                  bool force,
                  struct UpdateSystemResponse{} *out)
```

Triggers a rolling kernel switch (RKS). **waittimeout** specifies a timeout in sec to wait for an instance to start. If the timeout expires, the RKS procedure continues with the next instance. **softtimeout** specifies a timeout in sec for a soft shutdown of an instance, if the timeout expires a hard shutdown is used for the stop the instance. **UpdateSystem** performs various checks before actually starting the RKS procedure in order to ensure the system fulfills the RKS requirements. Most of the checks are mandatory and cause the operation to abort if not fulfilled. However, some minor checks are optional and RKS execution can be enforced even if these checks are not fulfilled using the **force** flag. The function works asynchronously just triggering the operation and returning immediately.

```
CheckUpdateSystem (struct CheckUpdateSystemResponse{} *out)
```

Checks prerequisites for executing a rolling kernel switch (RKS) like **UpdateSystem** does without actually executing the RKS.

GetSystemUpdateList(ArrayOfUpdateInstance *instance)

```
enum STATE_COLOR
{
      SAPControl_GRAY   = 1,
      SAPControl_GREEN  = 2,
      SAPControl_YELLOW = 3,
      SAPControl_RED    = 4
};


class UpdateInstance
{
        char *hostname;
        int  instanceNr;
        char *status;
        char *starttime;
        char *endtime;
        enum STATE_COLOR dispstatus;
};

class ArrayOfUpdateInstance
{
        UpdateInstance *__ptr;
        int __size;
};
```

Returns a list with the actual state of an ongoing RKS procedure triggered by **UpdateSystem.**

UpdateSCSInstance(struct UpdateSCSInstanceResponse{} *out)

Restarts an ABAP SCS instance during the RKS procedure ensuring enqueue and message server operation are suspended and resumed, and the state is properly restored during the instance restart. **UpdateSCSInstance** is used internally by **UpdateSystem** during the RKS procedure.

ABAPGetComponentList(ArrayOfABAPComponent *component)

```
class ABAPComponent
{
    char *component;
    char *release;
    char *patchlevel;
    char *componenttype;
    char *description;
};
```

```
class ArrayOfABAPComponent
{
    ABAPComponent *__ptr;
    int __size;
};
```

Returns a list with installed ABAP components as defined in CVERS database table. The function is used internally by **`UpdateSystem`** during the RKS procedure.

`ABAPCheckRFCDestinations``(ArrayOfString *destination)`

```
class ArrayOfString
{
      char **__ptr;
      int __size;
};
```

Returns a list of system internal RFC destination which connect a dedicated instance of a system. These RFC destinations can become a single point of failure in case the related instance fails or must be restarted (e.g. by the RKS procedure). The function is used internally by **`UpdateSystem`** during the RKS procedure.

`ABAPSetServerInactive``(  bool inactive=true,`
`struct ABAPSetServerInactiveResponse{} *out)`

Sets the ABAP instance to inactive or active state like “Deactivate” / “Activate” in transaction SM51.

AS Java (J2EE) Specific Methods

`J2EEGetProcessList``(ArrayOfJ2EEProcess *process)`

```
enum J2EE_PSTATE
{
      SAPControl_J2EE_STOPPED = 1,
      SAPControl_J2EE_STARTING = 2,
      SAPControl_J2EE_CORE_RUNNING = 3,
      SAPControl_J2EE_RUNNING = 4,
      SAPControl_J2EE_STOPPING = 5,
      SAPControl_J2EE_MAINTENANCE = 6,
      SAPControl_J2EE_UNKNOWN = 7
};

class J2EEProcess
{
      int    telnetPort;
```

```
        char *name;
        int    pid;
        char *type;
        char *restart;
        char *exitCode;
        enum J2EE_PSTATE state;
        char *statetext;
        char *startTime;
        char *elapsedTime;
        int    restartCount;
        int    errorCount;
        char *cpu;
        char *debug;
};

class ArrayOfJ2EEProcess
{
        J2EEProcess *__ptr;
        int __size;
};
```

Returns a list of AS Java server processes (j2ee processes) controlled by jcontrol / jstart (**superseded by J2EEGetProcessList2**).

J2EEGetProcessList2(ArrayOfJ2EEProcess2 *process)

```
enum J2EE_PSTATE
{
        SAPControl_J2EE_STOPPED = 1,
        SAPControl_J2EE_STARTING = 2,
        SAPControl_J2EE_CORE_RUNNING = 3,
        SAPControl_J2EE_RUNNING = 4,
        SAPControl_J2EE_STOPPING = 5,
        SAPControl_J2EE_MAINTENANCE = 6,
        SAPControl_J2EE_UNKNOWN = 7
};

class J2EEProcess2
{
        int   telnetPort;
        char *name;
        int   pid;
        char *type;
        char *restart;
        char *exitCode;
```

```
        enum J2EE_PSTATE state;
        char *statetext;
        char *startTime;
        char *elapsedTime;
        int   restartCount;
        int   errorCount;
        char *cpu;
        char *debug;
        int  clusterId;
};

class ArrayOfJ2EEProcess2
{
        J2EEProcess *__ptr;
        int __size;
};
```

Returns a list of AS Java processes controlled by jcontrol / jstart (**supersedes J2EEGetProcessList**).

```
J2EEControlProcess(     char                                *processname,
                        char                                *function,
                        struct J2EEControlProcessResponse{} *out)
```

Performs a given control function (EnableProcess/StartProcess, DisableProcess/StopProcess, SoftStopProcess, ActivateProcess, DeactivateProcess, RestartProcess, SoftRestartProcess, DumpStackTrace, EnableDebugging, DisableDebugging, IncrementTrace, DecrementTrace) on a given AS Java process. **processname** must match with some process name in the AS Java process list returned by **J2EEGetProcessList**. To perform AS Java instance wide operations (StartInstance, StopInstance, RestartInstance, BootInstance, RebootInstance) use **processname** “all”.

```
J2EEControlCluster(     char  *processname 1:1,
                        char  *function 1:1,
                        char  *host,
                        int   nr = -1,
                        struct J2EEControlClusterResponse{} *out);
```

J2EEControlCluster is similar to J2EEControlPorcess but performs the given control function on another J2EE instance (given by **host** and **nr**) or all J2EE instances within the system (host=NULL).

```
J2EEGetThreadList(ArrayOfJ2EEThread *thread)
```

```
enum STATE_COLOR
{
      SAPControl_GRAY   = 1,
      SAPControl_GREEN  = 2,
      SAPControl_YELLOW = 3,
      SAPControl_RED    = 4
};

class J2EEThread
{
      char  *processname;
      char  *startTime;
      char  *updateTime;
      char  *taskupdateTime;
      char  *subtaskupdateTime;
      char  *task;
      char  *subtask;
      char  *name;
      char  *classname;
      char  *user;
      char  *pool;
      char  *state;
      enum STATE_COLOR dispstatus;
};

class ArrayOfJ2EEThread
{
      J2EEThread *__ptr;
      int __size;
};
```

Returns a list of threads in the AS Java instance (**superseded by J2EEGetThreadList2**).

J2EEGetThreadList2(ArrayOfJ2EEThread2 *thread)

```
enum STATE_COLOR
{
      SAPControl_GRAY   = 1,
      SAPControl_GREEN  = 2,
      SAPControl_YELLOW = 3,
      SAPControl_RED    = 4
};

class J2EEThread2
{
```

```
        char  *processname;
        char  *startTime;
        char  *updateTime;
        char  *taskupdateTime;
        char  *subtaskupdateTime;
        char  *task;
        char  *subtask;
        char  *name;
        char  *classname;
        char  *user;
        char  *pool;
        char  *state;
        enum STATE_COLOR dispstatus;
        int   index;
};

class ArrayOfJ2EEThread2
{
        J2EEThread2 *__ptr;
        int __size;
};
```

Returns a list of threads in the AS Java instance (**supersedes J2EEGetThreadList**).

J2EEGetSessionList(ArrayOfJ2EESession *session)

```
class J2EESession
{
        char  *processname;
        int   IdHash;
        int   size;
        int   timeout;
        int   activeRequests;
        char  *startTime;
        char  *updateTime;
        char  *sticky;
        char  *corrupt;
        char  *backingStore;
};

class ArrayOfJ2EESession
{
        J2EESession *__ptr;
        int __size;
};
```

Returns a list of (HTTP) sessions in the AS Java instance (**superseded by J2EEGetWebSessionList**).

J2EEGetWebSessionList(ArrayOfJ2EEWebSession *session)

```
class J2EEWebSession
{
      char  *processname;
      int   IdHash;
      int   size;
      int   timeout;
      int   activeRequests;
      char  *startTime;
      char  *updateTime;
      char  *state;
      char  *backingStore;
      char  *user;
};

class ArrayOfJ2EEWebSession
{
      J2EEWebSession *__ptr;
      int __size;
};
```

Returns a list of (HTTP) sessions in the AS Java instance (**supersedes J2EEGetSessionList**).

J2EEGetCacheStatistic(ArrayOfJ2EECache *cache)

```
enum STATE_COLOR
{
      SAPControl_GRAY   = 1,
      SAPControl_GREEN  = 2,
      SAPControl_YELLOW = 3,
      SAPControl_RED    = 4
};

class J2EECache
{
      char  *cachename;
      char  *processname;
      char  *type;
      LONG64      size;
      LONG64      attrSize;
      LONG64      keysSize;
      int   cachedObjects;
      int   usedObjects;
      int   puts;
```

```
        int   gets;
        int   hits;
        int   changes;
        int   removes;
        int   evictions;
        int   instanceInvalidations;
        int   clusterInvalidations;
        char  *updateTime;
        enum STATE_COLOR dispstatus;
};

class ArrayOfJ2EECache
{
        J2EECache *__ptr;
        int __size;
};
```

Returns a list of caches in the AS Java instance (**superseded by J2EEGetCacheStatistic2**).

J2EEGetCacheStatistic(ArrayOfJ2EECache *cache)

```
enum STATE_COLOR
{
        SAPControl_GRAY   = 1,
        SAPControl_GREEN  = 2,
        SAPControl_YELLOW = 3,
        SAPControl_RED    = 4
};

class J2EECache2
{
        char  *description;
        char  *owner;
        char  *processname;
        char  *type;
        LONG64      size;
        LONG64      attrSize;
        LONG64      keysSize;
        int   cachedObjects;
        int   usedObjects;
        int   puts;
        int   gets;
        int   hits;
        int   changes;
        int   removes;
        int   evictions;
```

```
        int   instanceInvalidations;
        int   clusterInvalidations;
        char  *updateTime;
        enum STATE_COLOR dispstatus;
};

class ArrayOfJ2EECache2
{
        J2EECache2 *__ptr;
        int __size;
};
```

Returns a list of caches in the AS Java instance (**supersedes J2EEGetCacheStatistic**).

J2EEGetApplicationAliasList(ArrayOfJ2EEApplicationAlias *alias)

```
class J2EEApplicationAlias
{
    char *AppName;
    char *Alias;
    int  TotalRequests;
    char *AppActive;
    char *IgnoreCookie;
};


class ArrayOfJ2EEApplicationAlias
{
        J2EEApplicationAlias *__ptr;
        int __size;
};
```

Returns a list of application aliases in the AS Java instance.

J2EEGetComponentList(ArrayOfJ2EEComponentInfo *component)

```
enum STATE_COLOR
{
        SAPControl_GRAY   = 1,
        SAPControl_GREEN  = 2,
        SAPControl_YELLOW = 3,
        SAPControl_RED    = 4
};

class J2EEComponentInfo
{
```

```
        char   *type;
        char   *name;
        char   *startupmode;
        char   *status;
        char   *expectedstatus;
        char   *details;
        enum   STATE_COLOR dispstatus;
};

class ArrayOfJ2EEComponentInfo
{
        J2EEComponentInfo *__ptr;
        int __size;
};
```

Returns a list of configured J2EE components (services and applications).

```
J2EEControlComponents(  char   *processName,
                        char   *operation,
                        char   *componentType,
                        char   *componentNames,
                        struct J2EEControlComponentsResponse{} *out);
```

Performs a given **operation** (“start”, “stop” or “restart”) on a given component. **componentType** and **componentNames** must match with **type** and **name** returned by **J2EEGetComponentList**. **processName** must match with a J2EE sever node **name** returned by **J2EEGetProcessList**. Use “all” to perform the operation on all J2EE server nodes. To perform the same operation on multiple components, use a “,” separated list of components in **componentNames**.

```
J2EEGetEJBSessionList(ArrayOfJ2EEEJBSession *ejbsession)

class J2EEEJBSession
{
        int    IdHash;
        char   *state;
        int    size;
        int    activeRequests;
        int    totalRequests;
        char   *backingStore;
        char   *processname;
        char   *startTime;
        char   *updateTime;
        int    responseTime;
        char   *user;
        char   *transaction;
        char   *ejb;
```

```
        char  *application;
        char  *reference;
};

class ArrayOfJ2EEEJBSession
{
        J2EEEJBSession *__ptr;
        int __size;
};
```

Returns a list of EJB sessions in the AS Java instance.

J2EEGetRemoteObjectList(ArrayOfJ2EERemoteObject *remoteobject)

```
class J2EERemoteObject
{
        int   IdHash;
        char  *address;
        int   port;
        char  *protocol;
        char  *direction;
        int   stubs;
        int   implementations;
        char  *creationTime;
        char  *updateTime;
        char  *processname;
};

class ArrayOfJ2EERemoteObject
{
        J2EERemoteObject *__ptr;
        int __size;
};
```

Returns a list of remote object connections in the AS Java instance.

J2EEGetClusterMsgList(ArrayOfJ2EEClusterMsg *msg)

```
class J2EEClusterMsg
{
        char  *service;
        char  *id;
        LONG64      count;
        LONG64      length;
        LONG64      avg_length;
        LONG64      max_length;
```

```
        LONG64      count_p2p_msg;
        LONG64      count_p2p_request;
        LONG64      count_p2p_reply;
        LONG64      count_broadcast_msg;
        LONG64      count_broadcast_reply;
};

class ArrayOfJ2EEClusterMsg
{
        J2EEClusterMsg *__ptr;
        int __size;
};
```

Returns a list of J2EE cluster communication statistic from the message server.

J2EEGetSharedTableInfo(ArrayOfJ2EESharedTableInfo *jsf)

```
enum STATE_COLOR
{
        SAPControl_GRAY   = 1,
        SAPControl_GREEN  = 2,
        SAPControl_YELLOW = 3,
        SAPControl_RED    = 4
};

class J2EESharedTableInfo
{
    Char    *table;
    int     used;
    int     peak;
    int     limit;
    enum    STATE_COLOR dispstatus;
};

class ArrayOfJ2EESharedTableInfo
{
        J2EESharedTableInfo *__ptr;
        int __size;
};
```

Returns a list of SAP startup framework shared memory table information.

J2EEEnableDbgSession(char *processname,
                           char *flags,
                           char *client,
                           class J2EEEnableDbgSessionResponse{
                                 char *key;

```
                        int port;} *debuginfo)
```

Creates a J2EE debug session on a specific AS Java server process given by “processname”. Use “” as processname for automatic node selection. “flags” defines a set of debug flags given as a blank seperated list of keywords (“SuspendAll”, “CodeIsolate”, “LoadIsolate”, “MigrateSessions”, “KeepSession”, “NoDebugger”), default value is “LoadIsolate MigrateSessions”. “client” identifies the calling client “<user>@<host>” for monitoring. On success debug key and network port are returned.

```
J2EEDisableDbgSession(char *key, struct J2EEDisableDbgSessionResponse{} *out)
```

Removes a J2EE debug session given by “key” paramter previously created by “J2EEEnableDbgSession”.

```
J2EEGetThreadCallStack( int index,
                        class J2EEGetThreadCallStackResponse{
                              char *name;
                              ArrayOfString lines;} *callstack)

      class ArrayOfString
      {
            char **__ptr;
            int __size;
      };
```

Returns the java callstack of a given java thread (“index” parameter returned by “J2EEGetThreadList2”) or all java threads (index=-1).

```
J2EEGetThreadTaskStack( int index,
                        class J2EEGetThreadTaskStackResponse{
                              char *name;
                              ArrayOfString lines;} *taskstack)

      class ArrayOfString
      {
            char **__ptr;
            int __size;
      };
```

Returns the J2EE taskstack of a given java thread (”index” parameter returned by J2EEGetThreadList2) or all java threads (index=-1).

```
J2EEGetVMGCHistory(ArrayOfGCInfo *gc)

      class GCInfo
      {
            char  *processname;
```

```
        char  *type;
        char  *reason;
        char  *startTime;
        int   duration;
        int   cpuTime;
        LONG64      objBytesBefore;
        LONG64      objBytesAfter;
        LONG64      objBytesFreed;
        LONG64      clsBytesBefore;
        LONG64      clsBytesAfter;
        LONG64      clsBytesFreed;
        LONG64      heapSize;
        int   unloadedClasses;
};

class ArrayOfGCInfo
{
        GCInfo *__ptr;
        int __size;
};
```

Returns a list of JAVA VM garbage collections in the AS Java instance (**superseded by J2EEGetVMGCHistory2**).

J2EEGetVMGCHistory2(ArrayOfGCInfo2 *gc)

```
enum STATE_COLOR
{
        SAPControl_GRAY   = 1,
        SAPControl_GREEN  = 2,
        SAPControl_YELLOW = 3,
        SAPControl_RED    = 4
};

class GCInfo2
{
        char        *processname;
        char        *type;
        char        *reason;
        char        *startTime;
        int         duration;
        int         cpuTime;
        LONG64      objBytesBefore;
        LONG64      objBytesAfter;
        LONG64      objBytesFreed;
        LONG64      clsBytesBefore;
```

```
        LONG64      clsBytesAfter;
        LONG64      clsBytesFreed;
        LONG64      heapSize;
        int         unloadedClasses;
        LONG64      pageFaults;
        enum        STATE_COLOR dispstatus;
};

class ArrayOfGCInfo2
{
        GCInfo2 *__ptr;
        int __size;
};
```

Returns a list of JAVA VM garbage collections in the AS Java instance (**supersedes J2EEGetVMGCHistory**).

J2EEGetVMHeapInfo(ArrayOfHeapInfo *heap)

```
enum STATE_COLOR
{
        SAPControl_GRAY   = 1,
        SAPControl_GREEN  = 2,
        SAPControl_YELLOW = 3,
        SAPControl_RED    = 4
};

class HeapInfo
{
    char    *processname;
    char    *type;
    LONG64  size;
    LONG64  commitSize;
    LONG64  maxUsedSize;
    LONG64  initialSize;
    LONG64  maxSize;
    enum    STATE_COLOR dispstatus;
};

class ArrayOfHeapInfo
{
        HeapInfo *__ptr;
        int __size;
};
```

Returns a list of JAVA VM heap information.

ICM Specific Methods

ICMGetThreadList(ArrayOfICMThread *thread)

```
class ICMThread
{
      char  *name;
      char  *id;
      LONG64      requests;
      char  *status;
      char  *requesttype;
};

class ArrayOfICMThread
{
      ICMThread *__ptr;
      int __size;
};
```

Returns a list of threads used by ICM.

ICMGetConnectionList(ArrayOfICMConnection *connection)

```
class ICMConnection
{
      char  *conid;
      char  *protocol;
      char  *role;
      char  *requesttype;
      char  *peer_address;
      int   peer_port;
      char  *local_address;
      int   local_port;
      int   proc_timeout;
      int   keepalive_timeout;
      char  *connection_time;
      int   nihdl;
};

class ArrayOfICMConnection
{
      ICMConnection *__ptr;
      int __size;
};
```

Returns a list of incoming network connections handled by ICM.

ICMGetCacheEntries(ArrayOfICMCacheEntry *entry)

```
class ICMCacheEntry
{
    char    *name;
    int     version;
    LONG64  size;
    bool    valid;
    char    *cache;
    char    *creation_time;
    char    *last_access_time;
    char    *expiration_time;
    char    *cacheurl;
};

class ArrayOfICMCacheEntry
{
      ICMCacheEntry *__ptr;
      int __size;
};
```

Returns a list of objects cached by ICM. This list contains entries of all ICM caches, “cache” identifies the cache name. “**cacheurl**” can be used to read the object directly from the cache.

ICMGetProxyConnectionList(ArrayOfICMProxyConnection *connection)

```
class ICMProxyConnection
{
      char  *conid;
      char  *role;
      char  *peer_address;
      int   peer_port;
      char  *local_address;
      int   local_port;
      char  *status;
      int   nihdl;
      int   socket;
      char  *partner;

      char  *internal_convid 0:1;
      char  *external_convid 0:1;
      int   *snc_context_length 0:1;
      char  *failover_status 0:1;
      char  *disconnect_time 0:1;
```

```
      char  *objectid 0:1;
      char  *tid_uid_mode 0:1;
};

class ArrayOfICMProxyConnection
{
      ICMProxyConnection *__ptr;
      int __size;
};
```

Returns a list of outgoing network proxy connections handled by ICM. The list contains JCo and VM container proxy connections. “partner” identifies the actual connection type. The remaining fields are only set for the matching connection type.

Web Dispatcher Specific Methods

WebDispGetServerList(ArrayOfWebDispServer *server)

```
class WebDispServer
{
      Char        *sid;
      char        *instance;
      char        *hostname;
      char        *protocol;
      char        *type;
      char        *status;
      int         capacity;
      int         load;
      int         port;
      int         cur_conn;
      int         peak_conn;
      int         max_conn;
      int         sec_port;
      int         sec_cur_conn;
      int         sec_peak_conn;
      int         sec_max_conn;
      LONG64req_cnt_stateless;
      LONG64req_cnt_stateful;
      LONG64req_cnt_group;
      LONG64resp_time_min;
      LONG64resp_time_avg;
      LONG64resp_time_last;
      LONG64ping_time_last;
};

class ArrayOfWebDispServer
```

```
{
      WebDispServer      *__ptr;
      int                __size;
};
```

Returns a list backend servers connected to the Web Dispatcher.

`WebDispGetUrlPrefixList`(ArrayOfWebDispUrlPrefix *urlPrefix)

```
class WebDispUrlPrefix
{
      char         *sid;
      int          virthostnr;
      char         *vhosts;
      char         *urlprefix;
      char         *group;
};

class ArrayOfWebDispUrlPrefix
{
      WebDispUrlPrefix  *__ptr;
      int               __size;
};
```

Returns a list of Web Dispatcher URL prefixes configured in the backend system.

`WebDispGetVirtHostList`(ArrayOfWebDispVirtHost *hosts)

```
class WebDispVirtHost
{
      char         *sid;
      char         *name;
      int          virthostnr;
      char         *runlevel;
};

class ArrayOfWebDispVirtHost
{
      WebDispVirtHost   *__ptr;
      int               __size;
};
```

Returns a list of Web Dispatcher virtual hosts configured in the backend system.

WebDispGetGroupList (ArrayOfWebDispGroup *groups)

```
class WebDispGroup
{
      char        *sid;
      char        *name;
      char        *instance;
};

class ArrayOfWebDispGroup
{
      WebDispGroup      *__ptr;
      int               __size;
};
```

Returns a list of Web Dispatcher logon groups.

Enqueue Specific Methods

EnqGetLockTable (ArrayOfEnqLock *lock)

```
class EnqLock
{
    char *lock_name;
    char *lock_arg;
    char *lock_mode;
    char *owner;
    char *owner_vb;
    int  use_count_owner;
    int  use_count_owner_vb;
    char *client;
    char *user;
    char *transaction;
    char *object;
    bool backup;
};

class ArrayOfEnqLock
{
    EnqLock *__ptr;
    int __size;
};
```

Returns a list of all enqueue locks.

```
EnqRemoveLocks(    ArrayOfEnqLock *lock,
                   struct EnqRemoveLocksResponse{} *out)

class EnqLock
{
    char *lock_name;
    char *lock_arg;
    char *lock_mode;
    char *owner;
    char *owner_vb;
    int  use_count_owner;
    int  use_count_owner_vb;
    char *client;
    char *user;
    char *transaction;
    char *object;
    bool backup;
};

class ArrayOfEnqLock
{
    EnqLock *__ptr;
    int __size;
};
```

Deletes given locks from enqueue lock table.

```
EnqRemoveUserLocks(      char *user 1:1,
                         struct EnqRemoveUserLocksResponse{} *out)
```

Deletes all enqueue locks of given user from enqueue lock table.

```
EnqGetStatistic(  SAPControl__EnqStatistic *statistic)

class EnqStatistic
{
    int owner_now;
    int owner_high;
    int owner_max;
    enum STATE_COLOR owner_state;
    int arguments_now;
    int arguments_high;
    int arguments_max;
```

```
    enum STATE_COLOR arguments_state;
    int locks_now;
    int locks_high;
    int locks_max;
    enum STATE_COLOR locks_state;
    LONG64 enqueue_requests;
    LONG64 enqueue_rejects;
    LONG64 enqueue_errors;
    LONG64 dequeue_requests;
    LONG64 dequeue_errors;
    LONG64 dequeue_all_requests;
    LONG64 cleanup_requests;
    LONG64 backup_requests;
    LONG64 reporting_requests;
    LONG64 compress_requests;
    LONG64 verify_requests;
    double lock_time;
    double lock_wait_time;
    double server_time;
    enum STATE_COLOR replication_state;
};

enum STATE_COLOR
{
        SAPControl_GRAY   = 1,
        SAPControl_GREEN  = 2,
        SAPControl_YELLOW = 3,
        SAPControl_RED    = 4
};
```

Returns enqueue status and statistic counters from ENSA 1.

```
EnqGetStatistic2( SAPControl__EnqStatistic2 *statistic)

class EnqStatistic2
{
    int locks_now;
    int locks_high;
    int locks_max;
    enum SAPControl__STATE_COLOR locks_state;
    int locks_total;
    int lockops_total;
    LONG64 enqueue_requests;
    LONG64 dequeue_requests;
    LONG64 dequeue_all_requests;
    LONG64 reporting_requests;
    LONG64 backup_requests;
```

```
    LONG64 remove_by_arg;
    LONG64 remove_by_owner_pattern;
    LONG64 remove_by_arg_pattern;
    LONG64 config_replica;
    LONG64 update_replica;
    LONG64 takeover_replica;
    LONG64 admin_requests;
    LONG64 enqueue_rejects;
    LONG64 backup_records;
    LONG64 locks_search_max;
    double locks_search_avg;
    LONG64 locks_update_max;
    LONG64 locks_update_avg;
    LONG64 locks_takeover_max;
    LONG64 locks_takeover_avg;
    double lock_time;
    double lock_wait_time;
    LONG64 session_overflows;
    LONG64 table_overflows;
    LONG64 server_error;
    LONG64 connection_error;
    LONG64 timeouts;
    LONG64 network_requests;
    LONG64 rollouts;
    LONG64 rollins;
    double data_received;
    double data_sent;
    double server_time;
    enum SAPControl__STATE_COLOR replication_state;
};

enum STATE_COLOR
{
        SAPControl_GRAY   = 1,
        SAPControl_GREEN  = 2,
        SAPControl_YELLOW = 3,
        SAPControl_RED    = 4
};
```

Returns enqueue status and statistic counters from ENSA 2.

Gateway Specific Methods

```
GWGetConnectionList(    ArrayOfGWConnection *connection)

class GWConnection
{
```

```
    int  index;
    char *local_luname;
    char *local_tpname;
    char *remote_luname;
    char *remote_tpname;
    char *user;
    char *state;
    char *destination;
    char *convid;
    char *protocol;
    char *lastrequest;
    int  errorcode;
    char *local_address;
    char *remote_address;
};

class ArrayOfGWConnection
{
    GWConnection *__ptr;
    int __size;
};
```

Returns a list of all gateway connections.

```
GWGetClientList(  ArrayOfGWClient *client)

class GWClient
{
    int index;
    char *luname;
    char *tpname;
    char *type;
    char *address;
    char *lastrequest;
    char *state;
};

class ArrayOfGWClient
{
    GWClient *__ptr;
    int __size;
};
```

Returns a list of all gateway clients.

```
GWDeleteClients(  ArrayOfGWIndex index,
```

```
                    struct GWDeleteClientsResponse{} *out)

class ArrayOfGWIndex
{
    int *__ptr;
    int __size;
};
```

Deletes gateway client(s) specified by `index`.

```
GWDeleteConnections(    ArrayOfGWIndex index,
                        struct GWDeleteConnectionsResponse{} *out)

class ArrayOfGWIndex
{
    int *__ptr;
    int __size;
};
```

Deletes gateway connection(s) specified by `index`.

```
GWCancelConnections(    ArrayOfGWIndex index,
                        struct GWCancelConnectionsResponse{} *out)

class ArrayOfGWIndex
{
    int *__ptr;
    int __size;
};
```

Cancels gateway connection(s) specified by `index`.

Error Handling

The interface is using SOAP Faults and HTTP error handling. General failures are reported via “HTTP/1.1 500 Internal Server Error” and SOAP Fault, <faultstring> contains the error details, e.g.:

```
<SOAP-ENV:Fault>
  <faultcode>SOAP-ENV:Server</faultcode>
  <faultstring>DpIPCInit failed</faultstring>
</SOAP-ENV:Fault>
```

Missing user credentials are reported via “HTTP/1.1 401 Unauthorized” error code and SOAP Fault:

```
<SOAP-ENV:Fault>
  <faultcode>SOAP-ENV:Client</faultcode>
  <faultstring>HTTP Error: 'Unauthorized'</faultstring>
</SOAP-ENV:Fault>
```

Invalid user credentials are reported via HTTP/1.1 500 Internal Server Error” and SOAP Fault:

```
<SOAP-ENV:Fault>
  <faultcode>SOAP-ENV:Server</faultcode>
  <faultstring>Invalid Credentials</faultstring>
</SOAP-ENV:Fault>
```

Insufficient user privileges are reported via HTTP/1.1 500 Internal Server Error” and SOAP Fault:

```
<SOAP-ENV:Fault>
  <faultcode>SOAP-ENV:Server</faultcode>
  <faultstring>Permission denied</faultstring>
</SOAP-ENV:Fault>
```

Web Service Clients

SAP offers 3 graphical user interfaces using the SAPControl Web service interface:

- SAP Microsoft Management Console SnapIn (SAP MMC, Microsoft platforms only)
- SAP Java Management Console (SAP MC)
- SAP NetWeaver Administrator

In addition, a command line client “sapcontrol” is available, offering easy access to all webmethods:

```
NAME
    sapcontrol (Version: 915, patch 0, commit xxx)
SYNOPSIS
    sapcontrol [-prot <protocol>]
               [-trace <filename>]
               [-debug]
               [-user <user> <password>]
               [-queryuser]
```

```
                [-repeat <N> <D>]
                [-format <format>]
                [-host <hostname>]
                [-systempki <profile>]
                [-tio <timeout>]
                [-tmax <timeout>]
                -nr <instance number>
                -function <webmethod> [parameter list]
```

DESCRIPTION

Control and monitor SAP instances via Webservice interface of SAP Start Service.

OPTIONS

```
    -prot <protocol>
        Specify the protocol for the communication with the SAP instance.
        Available protocols are:
          NI_HTTP     HTTP  using SAP NI sockets (default, prefer Unix domain sockets)
          NI_HTTPS    HTTPS using SAP NI sockets (prefer Unix domain sockets)
          GSOAP_HTTP  HTTP  using gsoap built-in sockets
          WINHTTP     HTTP  using Windows winhttp
          WINHTTPS    HTTPS using Windows winhttp
          PIPE        Windows named pipes (on Unix same as NI_HTTP)
    -trace <filename>
        Trace SOAP request/response
    -debug
        Write local trace to stderr
    -user <user> <password>
        OS user and password for Webservice authentication
    -queryuser
        Query interactively for user and password
    -repeat <N> <D>
        Repeat webmethod call <N> times (-1=forever) with <D> sec delay
-format <format>
        Specify the format for the output of the webmethod.
        Available formats are:
          list    List output format (default)
          script  Script output format
          custom  Webmethod specific human readable format (only available for some webmethods)
          csv     CSV output format
    -host <hostname>
        Host to connect to (default: localhost)
```

```
    -systempki <profile>

        Use system pki from profile configuration to connect using HTTPS and
        authenticate with instance PSE certificate defined by profile

    -tio <timeout>

        Specify network I/O timeout in sec (default: 0 (blocking/infinite))

    -tmax <timeout>

        Specify max processing timeout in sec (default: 0 (infinite))
```

WEBMETHODS

```
    Start [runlevel]

    InstanceStart <hostname> <instance number> [<runlevel>]

    Bootstrap [<hostname> <instance number>]

    Stop [softtimeout sec]

    InstanceStop <hostname> <instance number> [<softtimeout sec>]

    Shutdown

    RestartInstance [<softtimeout sec> [<runlevel>]]

    StopService

    StartService <SID>

    RestartService

    ParameterValue [<parameter>]

    GetStartProfile

    GetTraceFile

    ListConfigFiles

    ReadConfigFile <filename>

    GetAlertTree

    GetAlerts

    GetEnvironment

    GetVersionInfo

    GetQueueStatistic

    GetProcessList

    GetInstanceProperties

    ListDeveloperTraces

    ReadDeveloperTrace <filename> <filesize>

    ListLogFiles

    ReadLogFile <filename> [<filter> [<language> [<maxentries> [<cookie>]]]]

    AnalyseLogFiles [<severity 0..2> [<maxentries>

        [<starttime YYYY MM DD HH:MM:SS> <endtime YYYY MM DD HH:MM:SS>]]]

    ConfigureLogFileList set|add|remove [<filename1> <filename2>... <filenameN>]

    GetLogFileList

    CreateSnapshot [<description> [<datcol_param> [<analyse_severity -1..2>
```

```
    [<analyse_maxentries> [<analyse_starttime YYYY MM DD HH:MM:SS>
    <analyse_endtime YYYY MM DD HH:MM:SS> [<maxentries>
    [<filename1> ... <filenameN>]]]]]]]
ReadSnapshot <filename> [<local filename>]
ListSnapshots
DeleteSnapshots <filename1> [<filename2>... <filenameN>]
GetAccessPointList
GetProcessParameter <processtype> [pid]
SetProcessParameter <processtype> <pid> <parameter> <value1>
    [<value2> ... <valueN>]
SetProcessParameter2 <processtype> <pid> [DYNAMIC|PERSIST|DYNAMIC_PERSIST] <parameter> <value1>
    [<value2> ... <valueN>]
ActivateSystemDBSuspendMode [<graceperiod sec> [<downtime sec>]]
DeactivateSystemDBSuspendMode
CheckParameter [<profile> [<default profile>]]
OSExecute <command> <async> <timeout> <protocolfile>
SendSignal <pid> <signal>
GetCallstack <pid>
GetSystemInstanceList [<timeout sec>]
StartSystem [ALL|SCS|DIALOG|ABAP|J2EE|TREX|ENQREP|HDB|ALLNOHDB|LEVEL <level>
    [<waittimeout sec> [<runlevel>]]]
StopSystem [ALL|SCS|DIALOG|ABAP|J2EE|TREX|ENQREP|HDB|ALLNOHDB|LEVEL <level>
    [<waittimeout sec> [<softtimeout sec>]]]
RestartSystem [ALL|SCS|DIALOG|ABAP|J2EE|TREX|ENQREP|HDB|ALLNOHDB|LEVEL <level>
    [<waittimeout sec> [<softtimeout sec> [<runlevel>]]]]
GetSystemUpdateList
UpdateSystem [<waittimeout sec> [<softtimeout sec> [<force>]]]
UpdateSCSInstance
CheckUpdateSystem
AccessCheck <function>
GetSecNetworkId <service_ip> <service_port> [<version> [<challenge>]]
GetNetworkId <service_ip> <service_port> [<version>]
RequestLogonFile <user>
UpdateSystemPKI [<force>]
UpdateInstancePSE [<force>]
StorePSE <server filename> <local filename> [<psemode> [<overwrite>]]
DeletePSE <filename>
CheckPSE <server filename> <local filename>
CreatePSECredential <server filename> <pin>
```

```
HACheckConfig

HACheckFailoverConfig

HAGetFailoverConfig

HAFailoverToNode <node>

HASetMaintenanceMode [<mode> [<instance_only>]]

HACheckMaintenanceMode [<instance_only>]

ABAPReadSyslog

ABAPReadRawSyslog

ABAPGetWPTable

ABAPGetComponentList

ABAPCheckRFCDestinations

ABAPGetSystemWPTable [<activeonly>]

ABAPSetServerInactive [<inactive>]

J2EEControlProcess <processname> <function>

J2EEControlCluster <processname> <function> [<hostname> <instance number>]

J2EEEnableDbgSession <client> [<processname> <debugflags>]

J2EEDisableDbgSession <debugkey>

J2EEGetProcessList

J2EEGetProcessList2

J2EEGetThreadList

J2EEGetThreadList2

J2EEGetThreadCallStack [<threadindex>]

J2EEGetThreadTaskStack [<threadindex>]

J2EEGetSessionList

J2EEGetCacheStatistic

J2EEGetCacheStatistic2

J2EEGetApplicationAliasList

J2EEGetComponentList

J2EEControlComponents <process name> <operation> <componenttype>
    <componentname1>,...,<componentnameN>

J2EEGetWebSessionList

J2EEGetWebSessionList2

J2EEGetEJBSessionList

J2EEGetRemoteObjectList

J2EEGetVMGCHistory

J2EEGetVMGCHistory2

J2EEGetVMHeapInfo

J2EEGetClusterMsgList

J2EEGetSharedTableInfo
```

```
    ICMGetThreadList

    ICMGetConnectionList

    ICMGetProxyConnectionList

    ICMGetCacheEntries

    WebDispGetServerList

    WebDispGetGroupList

    WebDispGetVirtHostList

    WebDispGetUrlPrefixList

    EnqGetStatistic

    EnqGetStatistic2

    EnqGetLockTable

    EnqRemoveUserLocks <user>

    GWGetConnectionList

    GWGetClientList

    GWCancelConnections <connection1> ... <connectionN>

    GWDeleteConnections <connection1> ... <connectionN>

    GWDeleteClients <client1> ... <clientN>

    StartWait <timeout sec> <delay sec> [<runlevel>]

    StopWait <timeout sec> <delay sec>

    WaitforStarted <timeout sec> <delay sec>

    WaitforStopped <timeout sec> <delay sec>

    RestartServiceWait <timeout sec> <delay sec>

    WaitforServiceStarted <timeout sec> <delay sec>

    CheckHostAgent

    CheckSystemCertificates <verifcation pse>
```

EXITCODES

```
    0  Last webmethod call successful

    1  Last webmethod call failed, invalid parameter

    2  StartWait, StopWait, WaitforStarted, WaitforStopped, RestartServiceWait
       timed out
       CheckSystemCertificates detected warnings

    3  GetProcessList succeeded, all processes running correctly
       CheckSystemCertificates detected errors

    4  GetProcessList succeeded, all processes stopped
```

SECURITY

```
     Trusted connects without user and password check are possible through
     Unix domain socket or Windows named pipes. Protected webmethods like
     Start or Stop require a trusted connection or OS user and password
     authentication.
```

EXAMPLES

```
sapcontrol -nr 0 -function GetProcessList
    Gets the list of processes on instance 00 on localhost
```

NOTES

```
A detailed description of the SAPControl webservice interface is available on
https://www.sap.com/documents/2016/09/0a40e60d-8b7c-0010-82c7-eda71af511fa.html.
The actual interface definition can be queried from the SAP Start Service via
http://<host>:5XX13/?wsdl or https://<host>:5xx14/?wsdl. The WSDL contains a short
documentation of each webmethod (XML tags "<documentation>").
SAP MMC (https://help.sap.com/docs/SUPPORT_CONTENT/si/3362959223.html) provides a
graphical user interface as Snap-In for the Microsoft Management Console.
SAP MC provides a graphical user interface as Java Swing UI launched from a browser
http://<host>:5XX13 or https://<host>:5XX14).
```

Web Server Functionality

In addition to offering Web service functionality sapstartsrv acts as a very simple Web server on the same TCP/IP ports. A web browser can download files by using HTTP get from sapstartsrv. The root directory of the Web server is $(DIR_EXECUTBALE)/servicehttp.

If no path is given in the request, the client will be redirected to $(DIR_EXECUTABLE)/servicehttp/sapmc/sapmc.html which is intended to download a SAP Java management client.

As of release 7.40 servicehttp contains a HTML documentation of all profile parameters in English (sapparamEN.html) and German (sapparamDE.html). If available, the URL to the English document can be obtained by webmethod GetInstanceProperties looking for property “Parameter Documentation”. The HTML document is tagged with the profile parameter names. So e.g. http://<host>:5XX13/sapparamEN.html#<parameter> can be used to open the English documentation of the given parameter.

Logfiles

- Major problems are reported in the **Windows application eventlog** and **Unix Syslog**.
- SAP developer traces are written to **$(DIR_HOME)/sapstartsrv.log** (backup from previous run: **$(DIR_HOME)/sapstartsrv.old**). Default trace level is 0 (see service/trace profile parameter).
- SAP instance start/stop is traced in **$(DIR_HOME)/sapstart.log**. stdout/stderr of started programs is redirected to stderr0-N.
- SLD registration is traced in **$(DIR_HOME)/dev_sldregs**
- LDAP registration is traced in **$(DIR_HOME)/dev_ldaps**
- Major operations are protocolled in **$(DIR_HOME)/history.glf** (starting with release 741)

Profile Parameters

- Autostart: 0: no auto start (default)
  1: auto start of SAP instance during service start
- SETENV_<N>: Environment variables to be set
- Execute_<N>: Commands to be executed before instance startup (Unix only)

- Start_Program_<N>: Commands to be executed for instance startup, use prefix “immediate” to start process synchronously (similar to Unix only Execute_<N>) and “local” to start process asynchronously.
- Restart_Program_<N>: Commands to be executed for instance startup, commands failing unexpectedly are restarted automatically
- Stop_Program_<N>: Commands to be executed after shutdown (Unix only)
- ldap/autoregister: 0: no registration (default)
  1: register service in LDAP directory during service start
- service/trace: 0-3: trace level for sapstartsrv.log (default 0)
- service/protectedwebmethods: Protected webservice functions requiring user authorization. Either a blank separated list of webmethods or a one of the 4 default sets optionally followed by webmethods to be added (+) or removed from the given default set ([ALL|SDEFAULT|DEFAULT|NONE] +|-<method1> +|-<method2>... +|-<methodN>). ALL: all webmethods, SDEFAULT: almost all webmethods (recommended), DEFAULT: all webmethods altering the instance state (Start, Stop, ...), NONE: no webmethods
- service/admin_users: Additional OS users authorized for system administration (blank separated list of OS user names)
- service/admin_groups: Additional OS user groups authorized for system administration (Unix only, blank separated list of OS user groups)
- service/datcol_command: Data collector command line to be executed when creating snapshots
- service/datcol_mandatory: Data collector mandatory for snapshot creation (default 0)
- service/datacol_timeout: Maximum allowed runtime for data collector when creating snapshots (default 300 sec)
- service/halib: HA shared library to load for controlling clustered instances (default: Windows: sapNThalib.dll, Unix: None)
- service/hardkillonshutdown: Kill instance processes during SAP Start Service stop (Windows only, default 0)
- service/hostname: Hostname or IP address for binding the Web service interface
- service/http/hostname: Hostname or IP address for binding http Web service interface
- service/https/hostname: Hostname or IP address for binding https Web service interface
- service/http/acl_file: Network access control list for http Web service interface
- service/https/acl_file: Network access control list for https Web service interface
- service/j2eethreadtasktime/yellow: thread task time threshold (in sec) for yellow rating (default: 10)
- service/j2eethreadtasktime/red: thread task time threshold (in sec) for yellow rating (default: 20)
- service/j2eeccahehitratio/yellow: 0-100: cache hit ratio threshold (percentage) for yellow rating (default: 80)
- service/j2eeccahehitratio/red: 0-100: cache hit ratio threshold (percentage) for red rating (default: 90)
- service/j2eevmheapusageratio/yellow: 0-100: java vm heap usage ratio threshold (percentage) for red rating (default: 80)
- service/j2eevmheapusageratio/red: 0-100: java vm heap usage ratio threshold (percentage) for red rating (default: 90)
- service/logfile_<N>: Logfiles accessible via SAP Host Agent (SAP Host Agent Mode only), may contain “*” and “?” wildcards or specify a directory subtree.

- service/max_dia_queue_time: ABAP dialog queue time threshold (in sec) for yellow rating (default 5), obsolete for releases > 7.38
- service/max_snapshots: Maximum number of snapshots to be archived, oldest snapshot is overwritten automatically (default 10)
- service/norestart: Disable automatic service restart after executable update (Windows only, default 0)
- service/porttypes: Enabled webservice ports
- service/sso_admin_user_<N>: Additional users authenticated by client certificate authorized for system administration. Use subject DN value of client certificate, may contain "*" and "?" wildcards.
- service/startpriority: Instance start/stop priority during system start/stop (default: not set => instance priority is calculated automatically from instance type, e.g. "0.3": HDB, "0.5": ENQREP, "1": SCS, "1.5": TREX, "2": ABAP with enqueue work process or messageserver, "3": Other). Instances are started from lowest to highest priority (lexicographical sorted) and stopped vice versa.
- service/status_procs: Processes monitored via status file (default: disp+work hdbdaemon)
- service/umask: Set umask for all instance processes (Unix only)

C# Sample Client Using the SAPControl Interface

The following section describes how to create a small self-written sample application using the Web service interface. The description assumes Microsoft Visual Studio 2010 or 2013 and a local SAP system using instance number 61 to be installed.

- Launch Visual Studio 2010.
- Use "File->New->Project..." to create a new project. Select "Other Languages->Visual C#" project type and the "Console Application" template. Enter a project name and press "OK" to create the project.
- Open the solution explorer using "View->Solution Explorer". Select the "References" node in the solution explorer tree and choose context menu "Add Service References...". Press "Advanced..." button. Press "Add Web Reference..." button. Enter the SAPControl WSDL URL http://localhost:56113/?wsdl and press "Go" arrow button. The documentation of SAPControl will be displayed with all Web service methods. Select "Add Reference". Visual Studio now generates a Web service proxy using namespace "localhost".
- Modify the empty "Main" function of the template code like this:

```
static void Main(string[] args)
{
    // Create a proxy object for the SAPControl interface
    localhost.SAPControl myservice = new localhost.SAPControl();

    // Declare an array of processes returned by GetProcessList()
    localhost.OSProcess[] osprocs;

    // Set the Url to connect to sapstartsrv of the SAP instance
    myservice.Url = "http://localhost:56113";

    // Get the list of running processes
    osprocs = myservice.GetProcessList();
```

```
            // Break into debugger to inspect 'osprocs' value
            System.Diagnostics.Debugger.Break();
        }
```

- Start the SAP instance if not yet done. Use "Debug->Start Debugging" to start the sample client in the Visual Studio debugger.
- The client will get the process list from sapstartsrv and break into the debugger afterwards. "osprocs" contains the same information as you can see in the MMC in the "Process List" node.
- To call protected webmethods you need to provide valid credentials (e.g. <sid>adm user and password). To do so add the following code (user/password must be UTF8 encoded):

```
using System.Net;
myservice.Credentials = new NetworkCredential("<sid>adm","<sidadm password>");
```

Using the SAPControl Interface with PowerShell

Microsoft Windows PowerShell v2 provides easy access to webservices by using "New-WebServiceProxy". Here are some examples how to use it to access the SAPControl webservice interface:

Create a proxy object from the WSDL:

```
PS C:\> $proxy = New-WebServiceProxy -uri http://ldcsbke:52013?wsdl
```

Call a simple webmethods without authentication:

```
PS C:\> $proxy.GetProcessList() | Format-Table -auto *

name       description         dispstatus textstatus starttime           elapsedtime  pid
----       -----------         ---------- ---------- ---------           -----------  ---
msg_server MessageServer  SAPControlGREEN Running    2011 10 06 07:49:48 23:24:15    4346
enserver   EnqueueServer  SAPControlGREEN Running    2011 10 06 07:49:48 23:24:15    4347
sapwebdisp Web Dispatcher SAPControlGREEN Running    2011 10 06 07:49:48 23:24:15    4348
```

Connect to a different sapstartsrv using the same proxy by setting the URL:

```
PS C:\> $proxy.Url = "http://ldcibke:53613"
PS C:\> $proxy.GetProcessList() | Format-Table -auto *

name      description       dispstatus textstatus starttime           elapsedtime   pid
----      -----------       ---------- ---------- ---------           -----------   ---
disp+work Dispatcher   SAPControlGREEN Running    2011 10 06 07:35:03 23:38:24    13305
igswd_mt  IGS Watchdog SAPControlGREEN Running    2011 10 06 07:35:03 23:38:24    13306
gwrd      Gateway      SAPControlGREEN Running    2011 10 06 07:35:05 23:38:22    13339
icman     ICM          SAPControlGREEN Running    2011 10 06 07:35:05 23:38:22    13340
```

Authenticate with user and password to call protected webmethods and filter the result:

```
PS C:\> $proxy.Credentials = new-object System.Net.NetworkCredential("bkeadm","<password>")
PS C:\> $proxy.ABAPGetWPTable() | where {$_.Typ -eq "BTC"} | Format-Table -auto *

No Typ   Pid Status Reason Start Err Sem Cpu     Time Program Client User      Action Table
-- ---   --- ------ ------ ----- --- --- ---     ---- ------- ------ ----      ------ -----
31 BTC 13374 Wait          yes           0:04:00
32 BTC 13375 Wait          yes           0:09:58
33 BTC 13376 Wait          yes           0:02:56
34 BTC 13377 Wait          yes           0:03:48
35 BTC 13378 Wait          yes           0:03:35
36 BTC 13379 Wait          yes           0:21:19
37 BTC 13380 Stop   RFC    yes           0:01:01 3            100    SCHMITTAN
38 BTC 13381 Wait          yes           0:07:38
39 BTC 13382 Wait          yes           0:02:09
40 BTC 13383 Wait          yes           0:04:08
```

Access a single property of the returned data:

```
PS C:\> $wp = $proxy.ABAPGetWPTable() | where {$_.Typ -eq "BTC"}
PS C:\> $wp[0].Pid
13374
```

Call a webmethod with input parameter:

```
PS C:\> $proxy.ParameterValue("rdisp/myname")
ldcibke_BKE_36
```

Using the SAPControl Interface with Python

Suds is a nice package for accessing webservices in python. Here is an example of a simple python webservice client accessing the SAPControl webservice interface:

```
from suds.client import Client

# Create proxy from WSDL
url = 'http://ldcibke:53613?wsdl'
client = Client(url)

# Call unprotected webmethod with complex output
```

```
result = client.service.GetProcessList()
print result
# Access output data
print 'PID:', result[0][0].pid

# Call unprotected webmethod with complex output on another instance
client.set_options(location='http://ldcsbke:52013')
result = client.service.GetProcessList()
print result

# Provide user and password for protected webmethod
client2 = Client(url, username='bkeadm', password='<password>')
result = client2.service.ParameterValue('rdisp/myname')
print 'rdisp/myname:', result
```

which produces the following output:

```
C:\>python.exe client.py
(ArrayOfOSProcess){
   item[] =
      (OSProcess){
         name = "disp+work"
         description = "Dispatcher"
         dispstatus = "SAPControl-GREEN"
         textstatus = "Running"
         starttime = "2011 10 07 07:39:09"
         elapsedtime = "40:07:48"
         pid = 19195
      },
      (OSProcess){
         name = "igswd_mt"
         description = "IGS Watchdog"
         dispstatus = "SAPControl-GREEN"
         textstatus = "Running"
         starttime = "2011 10 07 07:39:09"
         elapsedtime = "40:07:48"
         pid = 19196
      },
```

```
      (OSProcess){
         name = "gwrd"
         description = "Gateway"
         dispstatus = "SAPControl-GREEN"
         textstatus = "Running"
         starttime = "2011 10 07 07:39:11"
         elapsedtime = "40:07:46"
         pid = 19234
      },
      (OSProcess){
         name = "icman"
         description = "ICM"
         dispstatus = "SAPControl-GREEN"
         textstatus = "Running"
         starttime = "2011 10 07 07:39:11"
         elapsedtime = "40:07:46"
         pid = 19235
      },
 }
PID: 19195
(ArrayOfOSProcess){
   item[] =
      (OSProcess){
         name = "msg_server"
         description = "MessageServer"
         dispstatus = "SAPControl-GREEN"
         textstatus = "Running"
         starttime = "2011 10 07 07:52:11"
         elapsedtime = "39:54:46"
         pid = 20558
      },
      (OSProcess){
         name = "enserver"
         description = "EnqueueServer"
         dispstatus = "SAPControl-GREEN"
         textstatus = "Running"
         starttime = "2011 10 07 07:52:11"
         elapsedtime = "39:54:46"
         pid = 20559
```

```
      },
      (OSProcess){
         name = "sapwebdisp"
         description = "Web Dispatcher"
         dispstatus = "SAPControl-GREEN"
         textstatus = "Running"
         starttime = "2011 10 07 07:52:11"
         elapsedtime = "39:54:46"
         pid = 20560
      },
 }
rdisp/myname: ldcibke_BKE_36
```

Using the SAPControl Interface with Perl

SOAP::Lite is a package for accessing webservices in Perl. Unfortunately, WSDL support currently has some limitations. Here is an example of a simple python webservice client accessing the SAPControl webservice interface:

```
#!perl -w
use SOAP::Lite;
use Data::Dumper;
#Useful for soap request / response debugging:
#SOAP::Lite->import(+trace => qw(debug));

# Provide User and Password for calling protected webmethods
sub SOAP::Transport::HTTP::Client::get_basic_credentials
{
    return 'bkeadm' => '<password>';
}

# Create proxy
my $service = SOAP::Lite->service('http://ldcibke:53613?wsdl');
$service->proxy('http://ldcsbke:52013');

# Call unprotected webmethod with complex result data
my $plist = $service->GetProcessList();
print Dumper $plist;

# Call unprotected webmethod with complex result data on another instance
```

```
# Unfortunately overriding endpoints doesn't work if WSDL contains address location
#$service->proxy('http://ldcibke:53613');
#my $plist2 = $service->GetProcessList();
#print Dumper $plist2;

# Call protected webmethod with simple result data
print "\nrdisp/myname=",$service->ParameterValue('rdisp/myname'), "\n";
```

which produces the following output:

```
C:\>perl.exe client.pl
$VAR1 = {
          'item' => [
                      {
                        'pid' => '19195',
                        'textstatus' => 'Running',
                        'starttime' => '2011 10 07 07:39:09',
                        'name' => 'disp+work',
                        'description' => 'Dispatcher',
                        'elapsedtime' => '40:34:30',
                        'dispstatus' => 'SAPControl-GREEN'
                      },
                      {
                        'pid' => '19196',
                        'textstatus' => 'Running',
                        'starttime' => '2011 10 07 07:39:09',
                        'name' => 'igswd_mt',
                        'description' => 'IGS Watchdog',
                        'elapsedtime' => '40:34:30',
                        'dispstatus' => 'SAPControl-GREEN'
                      },
                      {
                        'pid' => '19234',
                        'textstatus' => 'Running',
                        'starttime' => '2011 10 07 07:39:11',
                        'name' => 'gwrd',
                        'description' => 'Gateway',
                        'elapsedtime' => '40:34:28',
```

```
                'dispstatus' => 'SAPControl-GREEN'
              },
              {
                'pid' => '19235',
                'textstatus' => 'Running',
                'starttime' => '2011 10 07 07:39:11',
                'name' => 'icman',
                'description' => 'ICM',
                'elapsedtime' => '40:34:28',
                'dispstatus' => 'SAPControl-GREEN'
              }
            ]
        };
```

```
rdisp/myname=ldcibke_BKE_36
```

Using the SAPControl Interface with ABAP

The SAP Netweaver documentation describes how to consume webservices. In general, you first need to generate an ABAP consumer proxy from the WSDL [8] and later configure the consumer proxy by adding a logical port for each webservice instance you want to connect to [9].

To generate the consumer proxy, download the SAPControl WSDL from sapstartsrv (e.g. http://localhost:56113/?wsdl) and save it as SAPControl.wsdl. Open SAPControl.wsdl with a text editor, search for line "<import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>", remove it and save SAPControl.wsdl. Unfortunately, this extra step is necessary since otherwise the ABAP proxy generation from the original WSDL would fail.

Start transaction SPROXY in SAP GUI. Select "Enterprise Services Browser" right click und "Objects" and choose "Create new object" to start the wizard. Select "Service Consumer". Select "External WSDL/Schema". Select "Local File". Enter the previously saved modified SAPControl.wsdl filename. Enter package name or select local object and enter a prefix (e.g. SAPCTRL). Save and activate the generated proxy. Latest 740 ABAP release already contains a proxy "CO_SSISAPCONTROL_PORT_TYPE".

The ABAP proxy generator incorrectly maps data type xsd:long (64 Bit signed integer) to ABAP data type INT4 (32 Bit signed integer). Since several SAPControl webmethods use xsd:long this can cause overflow exceptions when transporting huge values. To avoid this use the "Internal View" tab of the ABAP proxy editor to map xsd:long type parameters manually to "DEC(19)" instead of INT4. Currently effected webmethods are ListDeveloperTraces, J2EEGetCacheStatistic, J2EEGetCacheStatistic2, J2EEGetVMGCHistory, J2EEGetVMGCHistory2, J2EEGetVMHeapInfo, ListLogFiles, J2EEGetClusterMsgList, ICMGetThreadList, ICMGetCacheEntries, WebDispGetServerList, EnqGetStatistic, EnqGetStatistic2, ListSnapshots, GetProcessParameter.

To actually use the generated proxy a logical port which defines to which sapstartsrv to connect needs to be created first. Start transaction "SOAMANAGER" which launches SOA Management in a browser. In order to be

able to add logical ports your ABAP user may need additional privileges (see [10], SAP_BC_WEBSERVICE_CONFIGURATOR role). Select “Web Service Configuration”. Search for the generated proxy by limiting the search to Object Type “Consumer Proxy”. Select the proxy (e.g. CO_SAPCTRLSAPCONTROL_PORT_TYPE) from the list. Select “Create” and “Manual Configuration”. Enter “Logical Port Name” and “Description”, press “Next”. In “Consumer Security” tab leave the default or choose “X.509 SSL Client certificate” and enter an SSL Client PSE (e.g. “DFAULT”), press “Next”. In “HTTP Settings” tab enter “/SAPConrol.cgi” as “URL Access Path”, “Computer Name of Access URL” (e.g. localhost), “Port Number of Access URL” (e.g. 56113 for http or 56114 for https), select “URL Protocol Information” (HTTP or HTTPS), press “Next”. In “SOAP protocol” set “Message ID Protocol” to “Suppress ID Transfer”, press “Next” several times until the wizard finishes.

Start transaction se80 to test the generated proxy. Select the generated proxy class (e.g. “CO_SAPCTRLSAPCONTROL_PORT_TYPE”). Start the test framework, enter the “LOGICAL_PORT_NAME” of the created logical port (e.g. “SAPCONTROL_LDCSZDT_10”) and press “Instance”. Execute a webmethod (e.g. “GetProcessList”) and press “Execute”.

Starting with 740 SP8 SAP ships a generated proxy CO_SSISAPCONTROL_PROT_TYPE and a modified proxy CO_SAPCONTROL_PROXY for SAP internal usage, this can utilize the newly introduced system PKI for authentication. CO_SAPCONTROL_FACTORY can be used to easily instantiate both via an in memory only logical port (internally utilizing CL_SRT_PUBLIC_FACTORY), e.g.:

```
data sapcontrol type ref to CO_SAPCONTROL_FACTORY.

CREATE OBJECT sapcontrol EXPORTING
  iv_host = 'ldai1zdt'
  iv_nr = '11'
  iv_sapinternal = ' '.

sapcontrol->mo_public_proxy->get_process_list( exporting input = input
                                               importing output = output ).
```

References

[1] SAP note 142100, Windows: Problems with new SAP service as of Rel. 4.5B

[2] SAP note 823941, SAP Start Service on Unix

[3] SAP note 877795, Problems with SAP Start Service sapstartsrv as of Release 7.00

[4] SAP note 927637, Web service authentication in sapstartsrv as of Release 7.00

[5] SAP note 936273, sapstartsrv for all platforms

[6] SAP note 995116, Backward porting of sapstartsrv for earlier releases

[7] https://help.sap.com/doc/si/1.0/en-US/attachments/3362959223/SAP_System_Information_in_Directory_Services.pdf, SAP System Information in Directory services

[8] https://help.sap.com/saphelp_nwpi71/helpdata/en/69/8a1e9553dc4baba6026a3db510cadb/frameset.htm , SAP Netweaver: Generating a Consumer Proxy

[9] http://help.sap.com/saphelp_nwpi71/helpdata/en/9e/c7a3591dc74a679bbc9716354e42af/content.htm , SAP Netweaver: Configuring a Consumer Proxy

[10] SAP note 1318883, Introduction of new authorizations in Web Service Framework

[11] SAP note 1439348, Extended security settings for sapstartsrv

[12] SAP note 1822055, Enhanced SAP HA library interface

[13] SAP note 1642340, Using SSL in sapcontrol

[14] SAP note 1717846, sapstartsrv SLD registration

[15] SAP note 2200230, Problems with use of system PKI

[16] SAP note 2269532, Web dispatcher registration in SLD is incomplete

[17] SAP note 2361323, SAPControl reports “The handle is invalid” error on Windows

[18] SAP note 2361335, SAP MMC: New certificate check

[19] SAP note 2672809, Missing monitoring data for ICM in SAP MMC/MC

[20] SAP note 2737554, sapstartsrv does not report all processes in the process list (GetProcessList)

[21] SAP note 2780969, sapstartsrv AnalyseLogFiles analyzes some dev traces incorrectly

[22] SAP note 2816546, Incorrect registration of standalone enqueue server 2 in SLD

[23] SAP note 2820075, sapstartsrv logging improvements

[24] https://help.sap.com/docs/SUPPORT_CONTENT/si/3362959223.html, SAP MMC Snap-In

Interface Version History

Function	722	753	754	777	789	793	912	913	914	915	804
Start	X	X	X	X	X	X	X	X	X	X	X
Stop	X	X	X	X	X	X	X	X	X	X	X
Shutdown	X	X	X	X	X	X	X	X	X	X	X
StartBypassHA	211	X	X	X	X	X	X	X	X	X	-
StopBypassHA	211	X	X	X	X	X	X	X	X	X	-
InstanceStart	X	X	X	X	X	X	X	X	X	X	X
InstanceStop	X	X	X	X	X	X	X	X	X	X	X
Bootstrap	X	X	X	X	X	X	X	X	X	X	X
ParameterValue	X	X	X	X	X	X	X	X	X	X	X
GetProcessList	X	X	X	X	X	X	X	X	X	X	X
GetProcessList2	X	X	X	X	X	X	X	X	X	X	X
GetStartProfile	X	X	X	X	X	X	X	X	X	X	X
GetTraceFile	X	X	X	X	X	X	X	X	X	X	X
GetAlertTree	X	X	X	X	X	X	X	X	X	X	X
GetAlerts	X	X	X	X	X	X	X	X	X	X	X
RestartService	X	X	X	X	X	X	X	X	X	X	X
StopService	X	X	X	X	X	X	X	X	X	X	X
GetEnvironment	X	X	X	X	X	X	X	X	X	X	X

Function	722	753	754	777	789	793	912	913	914	915	804
ListDeveloperTraces	X	X	X	X	X	X	X	X	X	X	X
ListLogFiles	X	X	X	X	X	X	X	X	X	X	X
ReadDeveloperTrace	X	X	X	X	X	X	X	X	X	X	X
ReadLogFile	X	X	X	X	X	X	X	X	X	X	X
AnalyseLogFile	X	X	X	X	X	X	X	X	X	X	X
ConfigureLogFileList	X	X	X	X	X	X	X	X	X	X	X
GetLogFileList	X	X	X	X	X	X	X	X	X	X	X
RestartInstance	X	X	X	X	X	X	X	X	X	X	X
SendSignal	X	X	X	X	X	X	X	X	X	X	X
GetVersionInfo	X	X	X	X	X	X	X	X	X	X	X
GetQueueStatistic	X	X	X	X	X	X	X	X	X	X	X
GetInstanceProperties	X	X	X	X	X	X	X	X	X	X	X
OSExecute	X	X	X	X	X	X	X	X	X	X	X
AnalyseLogFiles	X	X	X	X	X	X	X	X	X	X	X
GetAccessPointList	X	X	X	X	X	X	X	X	X	X	X
GetSystemInstanceList	X	X	X	X	X	X	X	X	X	X	X
StartSystem	X	X	X	X	X	X	X	X	X	X	X
StopSystem	X	X	X	X	X	X	X	X	X	X	X
RestartSystem	X	X	X	X	X	X	X	X	X	X	X
AccessCheck	X	X	X	X	X	X	X	X	X	X	X
GetProcessParameter	X	X	X	X	X	X	X	X	X	X	X
SetProcessParameter	X	X	X	X	X	X	X	X	X	X	X
SetProcessParameter2	-	X	X	X	X	X	X	X	X	X	X
ActivateSystemDB-SuspendMode	-	-	-	-	X	X	X	X	X	X	-
DeactivateSystemDB-SuspendMode	-	-	-	-	X	X	X	X	X	X	-
CheckParameter	-	X	X	X	X	X	X	X	X	X	-
ShmDetach	X	X	X	X	X	X	X	X	X	X	X
CreateSnapshot	X	X	X	X	X	X	X	X	X	X	X
ReadSnapshot	X	X	X	X	X	X	X	X	X	X	X
ListSnapshots	X	X	X	X	X	X	X	X	X	X	X
DeleteSnapshots	X	X	X	X	X	X	X	X	X	X	X
RequestLogonFile	X	X	X	X	X	X	X	X	X	X	X
GetNetworkId	X	X	X	X	X	X	X	X	X	X	X
GetSecNetworkId	X	X	X	X	X	X	X	X	X	X	X

Function	722	753	754	777	789	793	912	913	914	915	804
UpdateSystem	-	X	X	X	X	X	X	X	X	X	X
CheckUpdateSystem	-	X	X	X	X	X	X	X	X	X	-
GetSystemUpdateList	-	X	X	X	X	X	X	X	X	X	X
UpdateSCSInstance	-	X	X	X	X	X	X	X	X	X	X
ListConfigurationFiles	-	X	X	X	X	X	X	X	X	X	-
ReadConfigFile	-	X	X	X	X	X	X	X	X	X	-
ABAPReadSyslog	X	X	X	X	X	X	X	X	X	X	X
ABAPReadRawSyslog	X	X	X	X	X	X	X	X	X	X	X
ABAPGetWPTable	X	X	X	X	X	X	X	X	X	X	X
ABAPGetSystemWPTable	217	X	X	X	X	X	X	X	X	X	-
ABAPAcknoledgeAlerts	X	X	X	X	X	X	X	X	X	X	X
ABAPGetComponentList	-	X	X	X	X	X	X	X	X	X	-
ABAPCheckRFCDestinations	-	X	X	X	X	X	X	X	X	X	-
ABAPSetServerInactive	-	?	?	?	X	X	X	X	X	X	-
J2EEGetProcessList	X	X	X	-	-	-	-	-	-	-	-
J2EEGetProcessList2	X	X	X	-	-	-	-	-	-	-	-
J2EEControlProcess	X	X	X	-	-	-	-	-	-	-	-
J2EEControlCluster	X	X	X	-	-	-	-	-	-	-	-
J2EEGetThreadList	X	X	X	-	-	-	-	-	-	-	-
J2EEGetThreadList2	X	X	X	-	-	-	-	-	-	-	-
J2EEGetSessionList	X	X	X	-	-	-	-	-	-	-	-
J2EEGetWebSessionList	X	X	X	-	-	-	-	-	-	-	-
J2EEGetCacheStatistic	X	X	X	-	-	-	-	-	-	-	-
J2EEGetCacheStatistic2	X	X	X	-	-	-	-	-	-	-	-
J2EEGetApplicationAliasList	X	X	X	-	-	-	-	-	-	-	-
J2EEGetComponentList	X	X	X	-	-	-	-	-	-	-	-
J2EEControlComponents	X	X	X	-	-	-	-	-	-	-	-
J2EEGetVMCHistory	X	X	X	-	-	-	-	-	-	-	-
J2EEGetVMCHistory2	X	X	X	-	-	-	-	-	-	-	-
J2EEGetVMHeapInfo	X	X	X	-	-	-	-	-	-	-	-
J2EEGetEJBSessionList	X	X	X	-	-	-	-	-	-	-	-
J2EEGetRemoteObjectList	X	X	X	-	-	-	-	-	-	-	-
J2EEGetClusterMsgList	X	X	X	-	-	-	-	-	-	-	-
J2EEGetSharedTableInfo	X	X	X	-	-	-	-	-	-	-	-
J2EEEnableDbgSession	X	X	X	-	-	-	-	-	-	-	-
J2EEDisableDbgSession	X	X	X	-	-	-	-	-	-	-	-

Function	722	753	754	777	789	793	912	913	914	915	804
J2EEGetThreadCallStack	X	X	X	-	-	-	-	-	-	-	-
J2EEGetTaskStack	X	X	X	-	-	-	-	-	-	-	-
ICMGetThreadList	X	X	X	X	X	X	X	X	X	X	X
ICMGetConnectionList	X	X	X	X	X	X	X	X	X	X	X
ICMGetCacheEntries	X	X	X	X	X	X	X	X	X	X	X
ICMGetProxyConnectionList	X	X	X	X	X	X	X	X	X	X	X
WebDispGetServerList	X	X	X	X	X	X	X	X	X	X	X
WebDispGetGroupList	-	X	X	X	X	X	X	X	X	X	X
WebDispGetVirtHostList	-	X	X	X	X	X	X	X	X	X	X
WebDispGeUrlPrefixList	-	X	X	X	X	X	X	X	X	X	X
EnqGetLockTable	X	X	X	X	X	X	X	X	X	X	X
EnqRemoveLocks	X	X	X	X	X	X	X	X	X	X	X
EnqRemoveUserLocks	-	X	X	X	X	X	X	X	X	X	-
EnqGetStatistic	X	X	X	X	X	X	X	X	X	X	X
EnqGetStatistic2	-	-	-	-	-	-	-	X	X	X	-
GWGetConnectionList	-	X	X	X	X	X	X	X	X	X	-
GWGetClientList	-	X	X	X	X	X	X	X	X	X	-
GWDeleteClients	-	X	X	X	X	X	X	X	X	X	-
GWDeleteConnections	-	X	X	X	X	X	X	X	X	X	-
GWCancelConnections	-	X	X	X	X	X	X	X	X	X	-
UpdateSystemPKI	-	X	X	X	X	X	X	X	X	X	-
UpdateInstancePSE	-	X	X	X	X	X	X	X	X	X	-
HACheckConfig	211	X	X	X	X	X	X	X	X	X	-
HACheckFailoverConfig	211	X	X	X	X	X	X	X	X	X	-
HAGetFailoverConfig	211	X	X	X	X	X	X	X	X	X	-
HAFailoverToNode	211	X	X	X	X	X	X	X	X	X	-
HASetMaintenanceMode	-	X	X	X	X	X	X	X	X	X	-
HACheckMaintenanceMode	-	X	X	X	X	X	X	X	X	X	-
GetCallstack	217	X	X	X	X	X	X	X	X	X	-
StorePSE	-	X	X	X	X	X	X	X	X	X	-
DeletePSE	-	X	X	X	X	X	X	X	X	X	-
CheckPSE	-	X	X	X	X	X	X	X	X	X	-
CreatePSECredential	-	X	X	X	X	X	X	X	X	X	-

Index

www.sap.com.