CHANGES IN MCS API VERSION 2.0

New Method for Initializing MCS

In version 2.0 of the MCS application programming interface (API) new functions have been added for initialization and release of MCS systems.

In previous API versions `SA_AddSystemToInitList` and `SA_InitSystems` had to be used to initialize all MCS at once and in the same communication mode – synchronous or asynchronous. In API version 2.0 the functions

```
SA_OpenSystem
SA_CloseSystem
SA_FindSystems
SA_GetSystemLocator
```

have been added to open and close one system at a time without affecting other systems. Every system can be configured for a different communication mode.

Locators for System Identification

One major difference to the older functions is how systems are identified. `SA_InitSystems` uses USB device IDs to identify an MCS and can only initialize systems connected over USB. `SA_OpenSystem` is more flexible and supports other communication interfaces. Systems are identified with a *locator* string, similar to URLs used to locate web pages. Typical locators are:

```
usb:id:3118167233
network:192.186.1.100:5000
```

The first locator identifies an MCS with the given system ID connected over USB. The second one identifies an MCS that is connected to the network.

The New Functions in Detail

```
SA_STATUS SA_OpenSystem(SA_INDEX *systemIndex,const char *systemLocator,
     const char *options)

SA_STATUS SA_CloseSystem(SA_INDEX systemIndex)

SA_STATUS SA_FindSystems(const char *options
     char *outBuffer,unsigned int *ioBufferSize)

SA_STATUS SA_GetSystemLocator(SA_INDEX systemIndex,
     char *outBuffer,unsigned int *ioBufferSize)
```

`SA_OpenSystem`
initializes one MCS specified in *systemLocator*. s*ystemIndex* is a handle to the opened system that is returned after a successful execution. It must be passed in the *systemIndex* parameter to the API functions. *options* is a string parameter for configuration options of the open command. The options must be separated by a comma or a newline. The following options are available:

- *reset* the MCS is reset on open.
- *async*, *sync* use the *async* option to set the communication mode to asynchronous, *sync* for synchronous communication.
- *open-timeout <t>* only available for network interfaces. *<t>* is the maximum time in milliseconds the PC tries to connect to the MCS. Default is 3000 milliseconds. The maximum timeout may be limited by operating system default parameters.

SmarAct GmbH
Schütte-Lanz-Straße 9
D-26135 Oldenburg
Germany

Tel.: +49 (0) 441 800 879-0
Fax: +49 (0) 441 800 879-21
E-mail info@smaract.de
Web: www.smaract.de

E.g: Initialize a network MCS for asynchronous communication. Wait not longer than 10 seconds for connect:

```
SA_OpenSystem(&index,"network:192.168.1.200:5000","async,open-timeout 10000");
```

SA_CloseSystem
closes a system initialized with SA_OpenSystem. It is important not to forget to close a system, because SA_ReleaseSystems cannot be used to "clean up" unclosed systems that have been initialized with SA_OpenSystem. Not closing a system will cause a *resource leak*. An attempt to open an unclosed MCS again will fail because the connection is still hold by the previous initialization.

SA_FindSystems
generates a list of MCS devices that are connected to the PC. Currently the function only lists MCS with a USB interface. o*ptions* is a string paramter for configuring the find procedure (currently unused). When calling this function the caller must pass a buffer with a sufficient size and write the buffer size in ioBufferSize. After the call the function has written a list of system locators into *outBuffer* and the number of bytes into *ioBufferSize*.

SA_GetSystemLocator
returns the locator of the initialized system *systemIndex* in *outBuffer*. See the remarks to *outBuffer* and *ioBufferSize* under SA_FindSystems above.

Automatically Generated System Indices

SA_OpenSystem generates a system index and returns it to the caller. It is not possible for the program (or the programmer) to know beforehand which index will be assigned to which system. The program must save the index returned by SA_OpenSystem in a variable and pass its value to the API functions. This is similar to other resource managing functions, e.g. file I/O, where the respective *open* or *create* functions typically return a special *handle* to the opened resource.

Backward Compatibility

SA_InitSystems and SA_ReleaseSystems are still supported by MCSControl v2.0 for backward compatibility and can be mixed with the new functions. This allows to keep existing application source code which uses SA_InitSystems and add new code that uses SA_OpenSystem. If you plan to mix old and new style functions please read section "Functions Compared".

We recommend to use open/close for managing MCS connections because of the many advantages listed above.

USB Device Locator Syntax

MCS devices with USB interface can be addressed with the following locator syntax:

```
usb:id:<id>
```

where <*id*> is the first part of a USB devices serial number which is usually printed on the MCS controller. This is the same ID you would use with the older function SA_AddSystemToInitSystemsList.

MCS with a USB interface can also be addressed with the alternative locator syntax

```
usb:ix:<n>
```

where the number <*n*> selects the *nth* device in the list of all currently connected MCS with a USB interface.

The drawback of identifying an MCS with this method is, that the number and order of the connected MCS can change between sessions, so the index *n* may not always refer to the same device. It is only safe to do this if you have exactly one MCS connected to the PC. We recommend to use the usb:id:... format for USB systems.

Support for Network Devices

Beginning with version 2.0 the API supports MCS with network interfaces. The network locator format is:

```
network:<ip>:<port>
```

<*ip*> is an IPv4 address which consists of four integer numbers between 0 and 255 separated by a dot. <*port*> is an integer number. For example, the locator `network:192.168.1.200:5000` addresses a device with the IP address 192.168.1.200 and TCP port 5000.

Some points should be remembered when writing or extending software for network communication:

- Network MCS systems can only be initialized with the new function `SA_OpenSystem`.
- Data transmission bandwidth and latencies over networks can vary much more than over USB. A program should not rely on the low transmission latencies typical for USB. Ensure that all timeout parameters in calls to functions like `SA_ReceiveNextPacket` are adequate for the network environment.
- The location (IP and port) of a MCS with network must be known to the program.

Functions Compared

The following table lists the capabilities of the old and new functions:

Old...	
`SA_InitSystems`	Initializes multiple devices connected over USB. All MCS are initialized in the same communication mode. If the *reset* flag is set in the *configuration* parameter, all MCS are reset.
`SA_ReleaseSystems`	Releases all systems that have been initialized with `SA_InitSystems`. Does not release systems opened with `SA_OpenSystem`. If a program mixes old and new functions it must call `SA_ReleaseSystems` to release all systems initialized with `SA_InitSystems` and `SA_CloseSystem` for each system that has been initialized with `SA_OpenSystem`.
`SA_AddSystemToInitSystemsList` `SA_ClearInitSystemsList`	Can only be used in conjunction with `SA_InitSystems`.
`SA_GetAvailableSystems`	Returns a list of system IDs of all MCS that are currently connected to the PC over USB.

`SA_GetInitState`
`SA_GetNumberOfSystems`
`SA_GetSystemID`
	These functions return information only about systems that have been initialized with `SA_InitSystems`. The value returned by `SA_GetNumberOfSystems` does not include the systems opened with `SA_OpenSystem`!

New...	
`SA_OpenSystem`	Opens any MCS that supports the MCS binary communication protocol.
`SA_CloseSystem`	Closes a system that has been opened with `SA_OpenSystem`. Does not close systems that have been initialized with `SA_InitSystems`.
`SA_FindSystems`	A more universal function to find MCS devices than `SA_GetAvailableSystems`. It returns a list of system locators for all MCS it can find. (Works only with USB devices currently)
`SA_GetSystemLocator`	The modern counterpart to `SA_GetSystemID`. The function returns the system locator for an initialized system. It can be used with systems that have been initialized with `SA_OpenSystem` or `SA_InitSystems`.

Elements Removed From the API

Some functions and constants that have been deprecated in earlier versions have been removed in v2.0. **Programs that use these functions or constants will not compile** until the function calls and constants are replaced by

alternatives (see table below). **Compiled programs that use some of the removed function will not run** if MCSControl (DLL or shared library) v1.x is replaced by v2.x on the computer. These programs must be recompiled for the MCSControl library v2.0+.

The following table lists the removed elements and alternatives.

Removed Elements	
Removed Function	**Alternative**
`SA_SetZeroPosition_S` `SA_SetZeroPosition_A`	Use `SA_SetPosition(systemIndex,channelIndex,0)` to set the scale to zero at the current actuator position.
`SA_SetReceiveNotification_A`	This function was available under Windows only. Call `SA_ReceiveNextPacket_A` to wait for the next packet. Due to the removal of this function, a client program can not use the Windows event mechanism to wait for a MCS packet anymore. Windows functions *WaitForSingleObject* or *WaitForMultipleObjects* must be replaced by calls to `SA_ReceiveNextPacket_A`.
`SA_ReceiveNextPacketIfChannel_A`	This function can be simulated by calling `SA_LookAtNextPacket_A` followed by `SA_DiscardPacket_A` if the packet has the expected channel index.
Removed Constant	**Replacement**
`SA_LIN20UMS_SENSOR_TYPE`	`SA_S_SENSOR_TYPE`
`SA_ROT3600S_SENSOR_TYPE`	`SA_SR_SENSOR_TYPE`
`SA_ROT50LS_SENSOR_TYPE`	`SA_ML_SENSOR_TYPE`
`SA_ROT50RS_SENSOR_TYPE`	`SA_MR_SENSOR_TYPE`
`SA_LINEAR_SENSOR_TYPE`	`SA_S_SENSOR_TYPE`
`SA_ROTARY_SENSOR_TYPE`	`SA_SR_SENSOR_TYPE`

Programming Example

The following *C++* code example opens two MCS in different communications modes. System 1 is opened for synchronous communication and is reset on open, system 2 is configured for asynchronous communication.

```
#include <MCSControl.h>
#include <iostream>

using namespace std;

void ExitOnError(SA_STATUS st) {
    if(st) {
        cout << "Error " << st << endl;
        exit(1);
    }
}

const char loc1[] = "usb:id:3118167233";
const char loc2[] = "network:192.168.1.200:5000";

void main(void) {
    SA_PACKET pk;
    int position;
    unsigned int status;
    unsigned int sys1,sys2;
    ExitOnError( SA_OpenSystem(&sys1,loc1,"sync,reset") );
    ExitOnError( SA_GetStatus_S(sys1,0,&status) );
    cout << "Status of channel 0 of system 1 is " << status << endl;

    ExitOnError( SA_OpenSystem(&sys2,loc2,"async,open-timeout 1500") );
    ExitOnError( SA_GetStatus_A(sys2,0) );
    ExitOnError( SA_ReceiveNextPacket_A(sys2,2000,&packet) );
    cout << "Status of channel 0 of system 2 is " << pk.data1 << endl;
    ExitOnError( SA_CloseSystem(sys2) );

    ExitOnError( SA_GetPosition_S(sys1,0,&position) );
    cout << "Position of channel 0 of system 1 is " << position << endl;

    ExitOnError( SA_CloseSystem(sys1) );
}
```