Servicebroker

Version 0.1

Harman/Becker Automotive Systems GmbH

Draft

Servicebroker: Version 0.1

Harman/Becker Automotive Systems GmbH

Becker-Göring-Straße 16, 76307 Karlsbad, Germany

Author:
Florentin Picioroaga Florentin.Picioroaga@harman.com

Person in charge:
Florentin Picioroaga Florentin.Picioroaga@harman.com
Publication date 19.03.2012

Draft

Revision History

Revision 0.1	February, 2012	FPicioroaga
Initial version		

Table of Contents

List of Figures

Document overview

This document is part of a documentation package comprising *DSI* and focuses on the service discovery part of the DSI system.

The document is organized in the following chapters:

1. Introduction

   In this chapter an overview of the DSI system is presented and the basic principles of the service discovery are explained.

2. Servicebroker behaviour

   The chapter explaining the Servicebroker dynamic behaviour and its API.

3. Doxygen Servicebroker API

   This is the extract from the source code documentation.

1 Introduction

This chapter provides an introduction to the DSI system explaining how the applications find themselves at run-time.

1.1 Service discovery

Currently, the DSI applications use a service discovery mechanism at run-time , the Servicebroker, in order to build-up communication connections.

The Servicebroker has the role to provide:

- a notification mechanism to inform applications about services that sign-in and sign-out.
- address information used by applications to establish communication over the DSI IPC.

The Servicebroker can be used in a distributed system and can scale gracefully its infrastructure over many nodes. The naming service organization structure is similar with the DNS (Domain Naming Service) structure where each node in the network is serviced by a Servicebroker. The Servicebrokers are organized in a tree structure, see below image. A Servicebroker node in the tree, the slave Servicebroker, will forward its unresolved requests and its information about local existing services to its parent, its master Servicebroker.

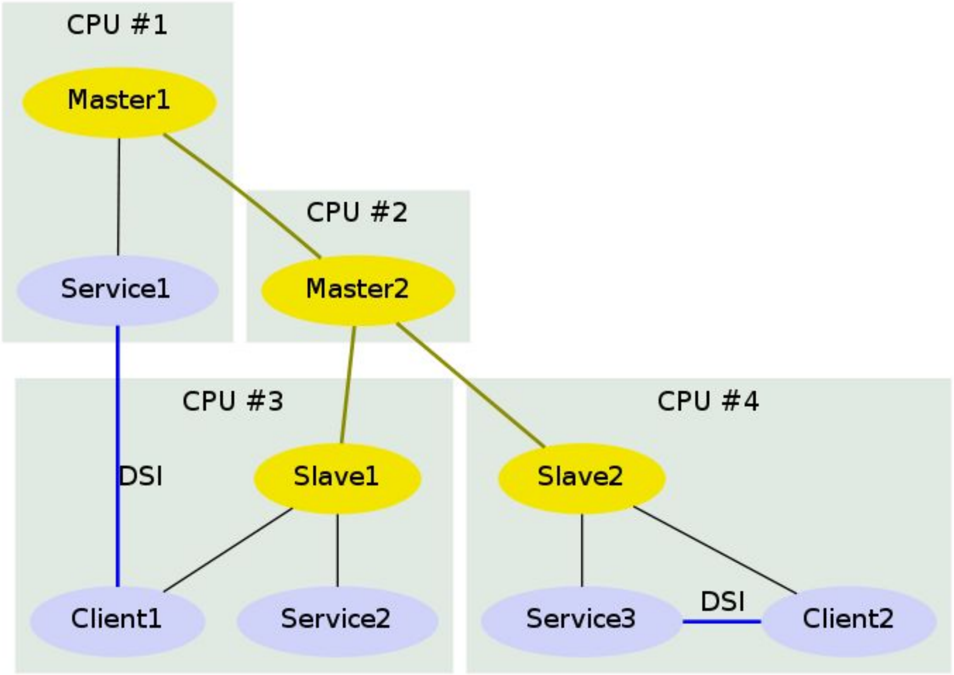


Figure 1.1. DSI run-time environment

In the image above, the yellow part represents the Servicebroker's tree structure, the blue lines are the DSI IPC communication connections and the black lines represent the communication connections between the DSI applications and the Servicebrokers. In this document only the two communication protocols where the Servicebroker is involved will be described. An improvement to the current architecture is to have these protocols run over DSI IPC.

A simplified description from the point of view of a DSI application on how the service discovery works is done here, a deeper view is realized in the next chapter.

The DSI applications connect on start-up to the Servicebroker running on the node and register the service interfaces they implemented. In case they require to connect to a service interface implementation that is external to the application they can:

- query the Servicebroker for the address information of the requested service. On success the application can already start communicating over DSI IPC with the requested service implementation.

- if the service is not present in the system a notification will be set to be sent by the Servicebroker when the respective service implementation is registered in the system. The application will be able to repeat the address information query on receiving of the notification.

2 Servicebroker behaviour

This chapter will describe the dynamic behaviour of the Servicebroker on base of sequence diagrams.

2.1 Dynamic behaviour

These user cases are intended to bring a better understanding of the interactions between the DSI applications and the Servicebroker.

Some general considerations about the diagrams:

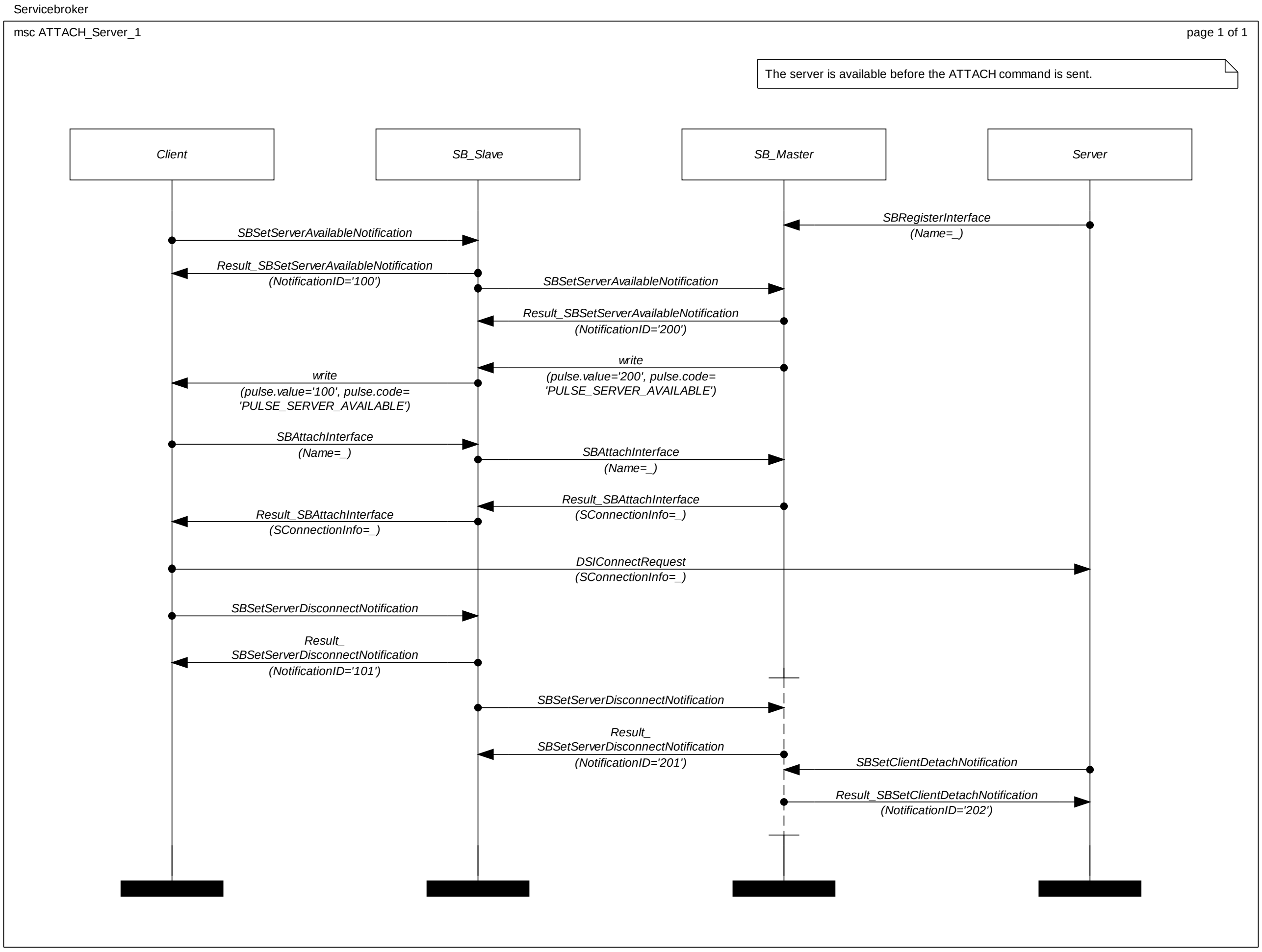
- In order to not overload the diagrams only the most relevant information for each action will be provided.
- The servicebrokers generate notification ids that are unique only for themselves, the ids are not unique among the complete servicebroker tree structure.
- Naming conventions for the actions:
  - API names are used whenever possible.
  - Suggestive names are used to describe a logical step.
  - libc names are used for low level operations, e.g. write will send data to a file descriptor.

2.1.1 UC1: attach to an available server

A client attaches to a server that is already known in the system. First the client creates a notification on the servicebroker for the event the server will appear in the system.

The servicebrokers inter-communicate and inform the client that the server is already present in the DSI system.

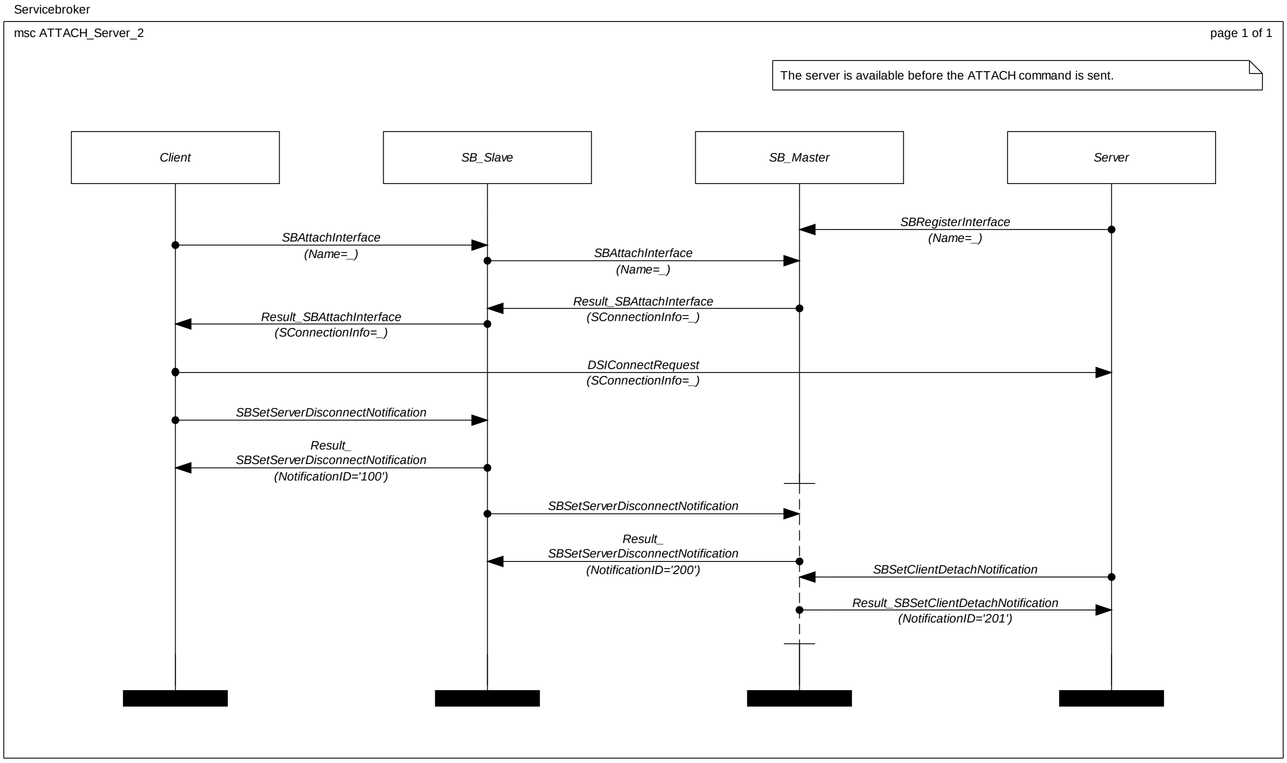
After the DSI connection is established both communication partners set notifications on the servicebroker in the event that the communication partner leaves the DSI system.


Servicebroker
msc ATTACH_Server_1
page 1 of 1
The server is available before the ATTACH command is sent.
Client
SB_Slave
SB_Master
Server
SBRegisterInterface
(Name=_)
SBSetServerAvailableNotification
Result_SBSetServerAvailableNotification
(NotificationID='100')
SBSetServerAvailableNotification
Result_SBSetServerAvailableNotification
(NotificationID='200')
write
(pulse.value='200', pulse.code=
'PULSE_SERVER_AVAILABLE')
write
(pulse.value='100', pulse.code=
'PULSE_SERVER_AVAILABLE')
SBAttachInterface
(Name=_)
SBAttachInterface
(Name=_)
Result_SBAttachInterface
(SConnectionInfo=_)
Result_SBAttachInterface
(SConnectionInfo=_)
DSIConnectRequest
(SConnectionInfo=_)
SBSetServerDisconnectNotification
Result_
SBSetServerDisconnectNotification
(NotificationID='101')
SBSetServerDisconnectNotification
Result_
SBSetServerDisconnectNotification
(NotificationID='201')
SBSetClientDetachNotification
Result_SBSetClientDetachNotification
(NotificationID='202')

2.1.2 UC2: attach to an available server

A client attaches to a server that is already known in the DSI system.

The difference to the UC1 is that the client will request immediately the connection information to the servicebroker. In this case the client bets on the presence of the server in the system and saves some communication time.



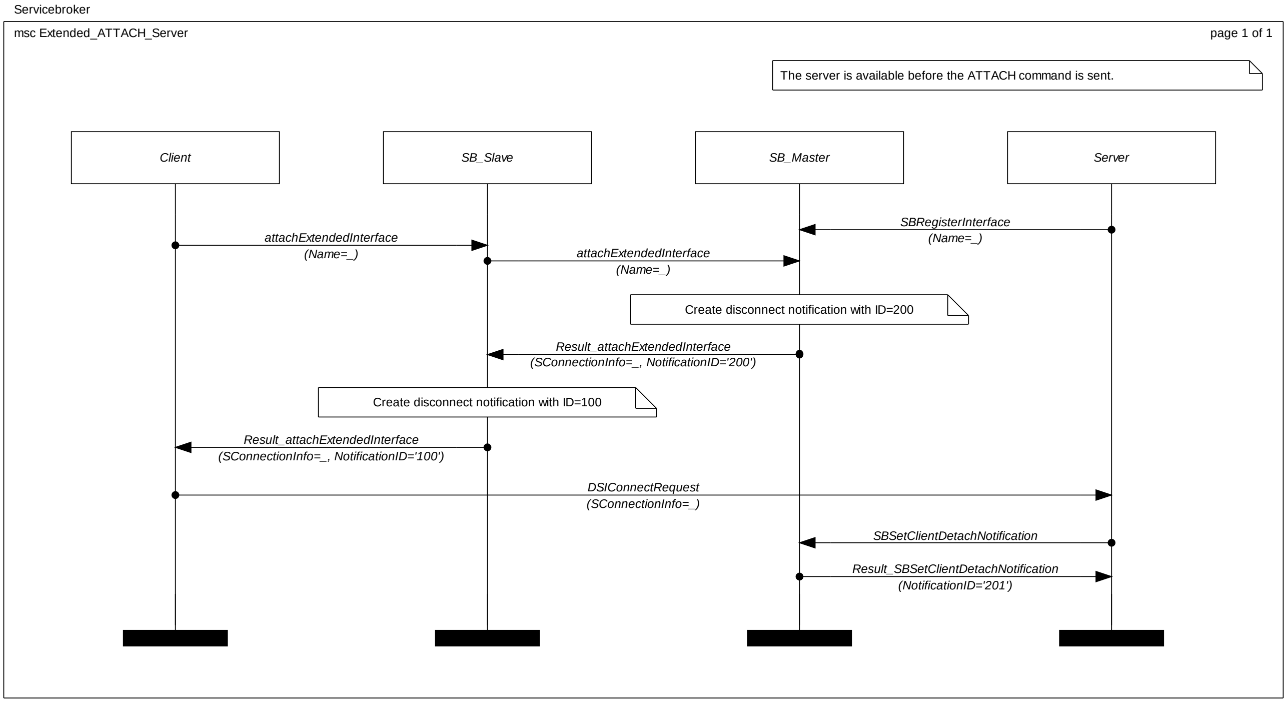
2.1.3 UC3: extended attach to an available server

The client uses an improved and newer version of the attach command. The attach extended command was introduced to save even more communication time. This can bring a consistent especially when the DSI applications run on different CPUs.

The command will combine the attach, setting server available and server disconnect notifications.

In case the server is not available in the DSI system a connect notification will be automatically set on servicebroker without the need of an extra step.

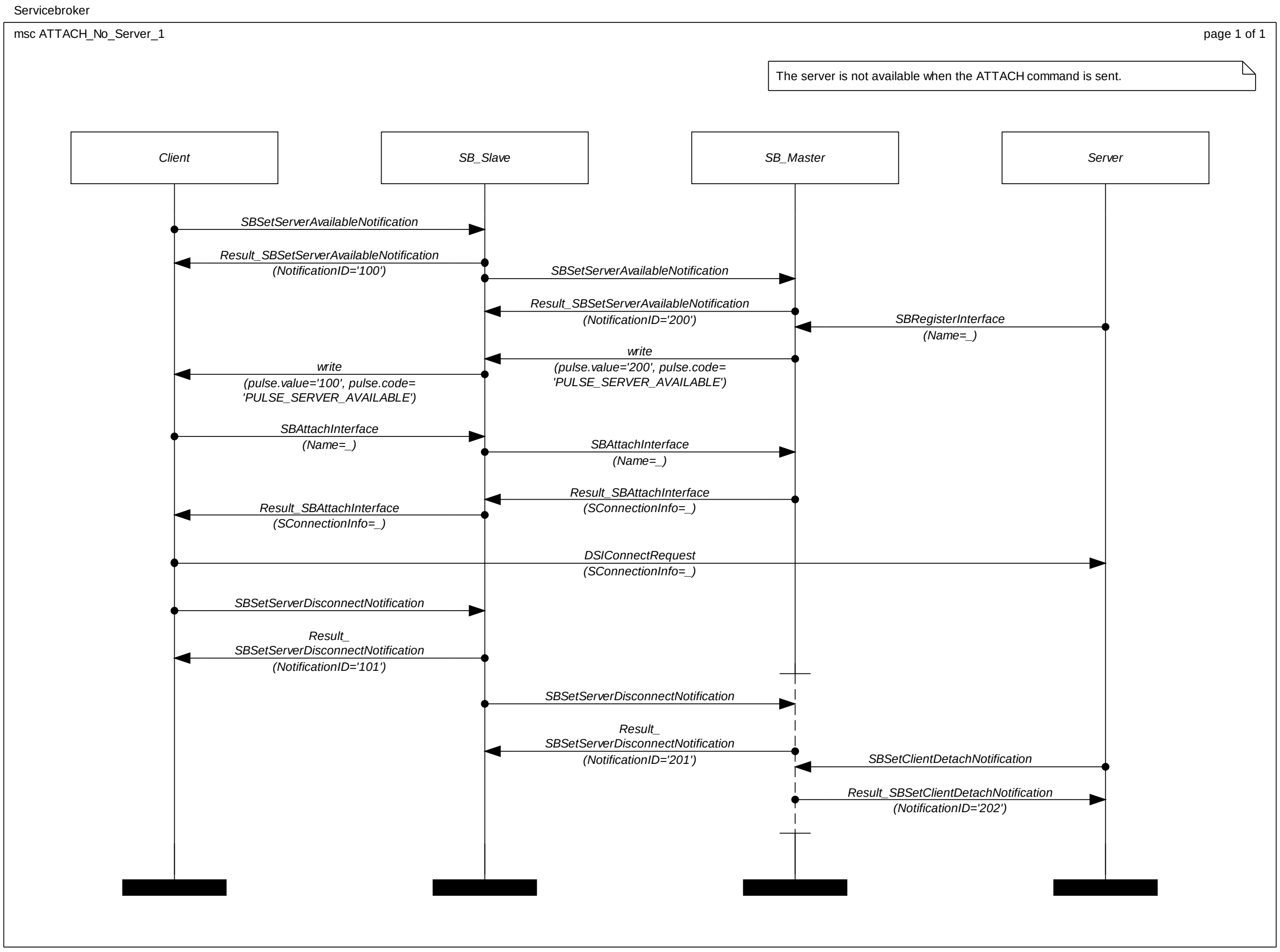
If the server is available in the DSI system the connection information will be delivered and a disconnect notification will be automatically set on servicebroker without the need of an extra step.


Servicebroker
msc Extended_ATTACH_Server
page 1 of 1
The server is available before the ATTACH command is sent.
Client
SB_Slave
SB_Master
Server
SBRegisterInterface
(Name=_)
attachExtendedInterface
(Name=_)
attachExtendedInterface
(Name=_)
Create disconnect notification with ID=200
Result_attachExtendedInterface
(SConnectionInfo=_, NotificationID='200')
Create disconnect notification with ID=100
Result_attachExtendedInterface
(SConnectionInfo=_, NotificationID='100')
DSIConnectRequest
(SConnectionInfo=_)
SBSetClientDetachNotification
Result_SBSetClientDetachNotification
(NotificationID='201')

2.1.4 UC4: attach to a not available server

A client attaches to a server that is not known in the DSI system.

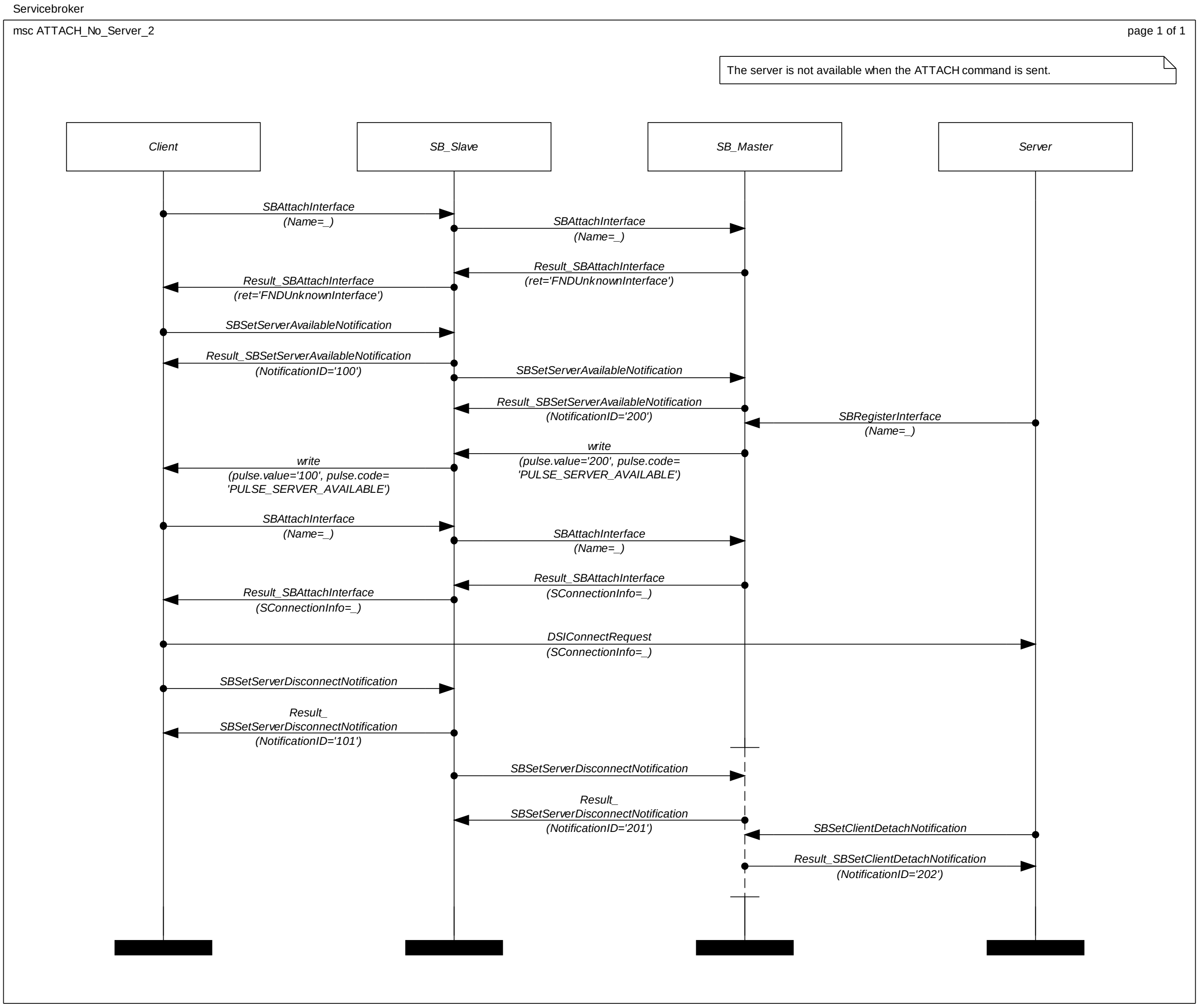
This use case will generate the same amount of communication in the system as in UC1.


Servicebroker
msc ATTACH_No_Server_1
page 1 of 1
The server is not available when the ATTACH command is sent.
Client
SB_Slave
SB_Master
Server
SBSetServerAvailableNotification
Result_SBSetServerAvailableNotification (NotificationID='100')
SBSetServerAvailableNotification
Result_SBSetServerAvailableNotification (NotificationID='200')
SBRegisterInterface (Name=_)
write (pulse.value='200', pulse.code= 'PULSE_SERVER_AVAILABLE')
write (pulse.value='100', pulse.code= 'PULSE_SERVER_AVAILABLE')
SBAttachInterface (Name=_)
SBAttachInterface (Name=_)
Result_SBAttachInterface (SConnectionInfo=_)
Result_SBAttachInterface (SConnectionInfo=_)
DSIConnectRequest (SConnectionInfo=_)
SBSetServerDisconnectNotification
Result_ SBSetServerDisconnectNotification (NotificationID='101')
SBSetServerDisconnectNotification
Result_ SBSetServerDisconnectNotification (NotificationID='201')
SBSetClientDetachNotification
Result_SBSetClientDetachNotification (NotificationID='202')

2.1.5 UC5: attach to a not available server

A client attaches to a server that is not known in the DSI system.

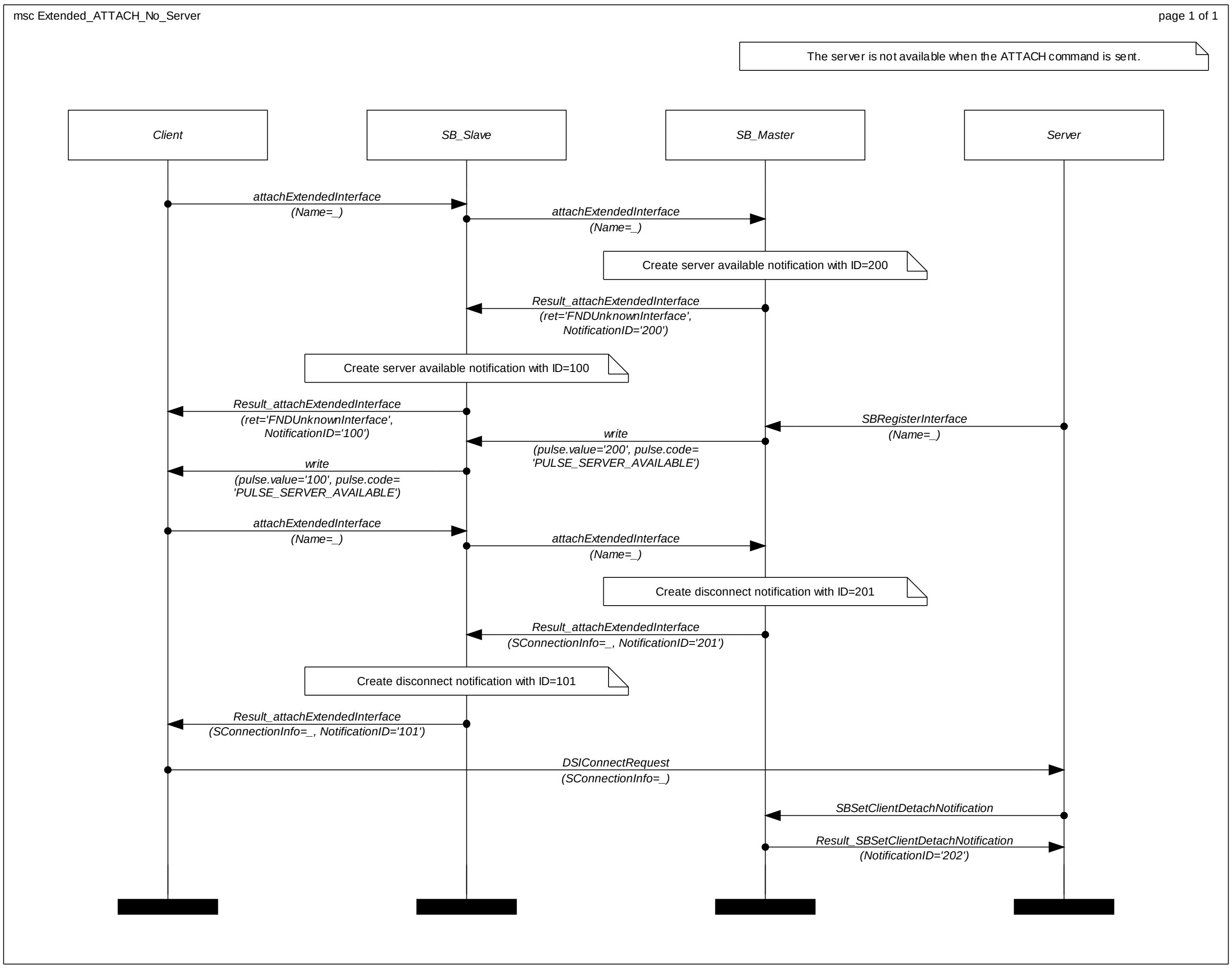
In this use case the bet of the client that the server is in the DSI system will generate more communication than in the previous use case.


Servicebroker
msc ATTACH_No_Server_2
page 1 of 1
The server is not available when the ATTACH command is sent.
Client
SB_Slave
SB_Master
Server
SBAttachInterface (Name=_)
SBAttachInterface (Name=_)
Result_SBAttachInterface (ret='FNDUnknownInterface')
Result_SBAttachInterface (ret='FNDUnknownInterface')
SBSetServerAvailableNotification
Result_SBSetServerAvailableNotification (NotificationID='100')
SBSetServerAvailableNotification
Result_SBSetServerAvailableNotification (NotificationID='200')
SBRegisterInterface (Name=_)
write (pulse.value='200', pulse.code= 'PULSE_SERVER_AVAILABLE')
write (pulse.value='100', pulse.code= 'PULSE_SERVER_AVAILABLE')
SBAttachInterface (Name=_)
SBAttachInterface (Name=_)
Result_SBAttachInterface (SConnectionInfo=_)
Result_SBAttachInterface (SConnectionInfo=_)
DSIConnectRequest (SConnectionInfo=_)
SBSetServerDisconnectNotification
Result_ SBSetServerDisconnectNotification (NotificationID='101')
SBSetServerDisconnectNotification
Result_ SBSetServerDisconnectNotification (NotificationID='201')
SBSetClientDetachNotification
Result_SBSetClientDetachNotification (NotificationID='202')

2.1.6 UC6: extended attach to a not available server

The improved attach command delivers in this case less communication traffic as in the previous use cases where the server is not already in the DSI system.

Servicebroker


msc Extended_ATTACH_No_Server
page 1 of 1
The server is not available when the ATTACH command is sent.
Client
SB_Slave
SB_Master
Server
attachExtendedInterface (Name=_)
attachExtendedInterface (Name=_)
Create server available notification with ID=200
Result_attachExtendedInterface (ret='FNDUnknownInterface', NotificationID='200')
Create server available notification with ID=100
Result_attachExtendedInterface (ret='FNDUnknownInterface', NotificationID='100')
SBRegisterInterface (Name=_)
write (pulse.value='200', pulse.code= 'PULSE_SERVER_AVAILABLE')
write (pulse.value='100', pulse.code= 'PULSE_SERVER_AVAILABLE')
attachExtendedInterface (Name=_)
attachExtendedInterface (Name=_)
Create disconnect notification with ID=201
Result_attachExtendedInterface (SConnectionInfo=_, NotificationID='201')
Create disconnect notification with ID=101
Result_attachExtendedInterface (SConnectionInfo=_, NotificationID='101')
DSIConnectRequest (SConnectionInfo=_)
SBSetClientDetachNotification
Result_SBSetClientDetachNotification (NotificationID='202')

A.1 Reference

A.1.1 Header <dsi/clientlib.h>

```
int SBOpen(const char *);
int SBOpenNotificationHandle();
void SBClose(int);
int SBRegisterInterface(int, const char *, int, int, int, SPartyID *);
int SBRegisterInterfaceTCP(int, const char *, int, int, uint32_t, uint16_t,
                           SPartyID *);
int SBRegisterGroupInterface(int, const char *, int, int, int, const char *,
                             SPartyID *);
int SBRegisterInterfaceEx(int, const struct SFNDInterfaceDescription *, int,
                          int, SPartyID *);
int SBRegisterInterfaceExTCP(int, const struct SFNDInterfaceDescription *,
                             int, uint32_t, uint16_t, SPartyID *);
int SBUnregisterInterface(int, SPartyID);
int SBAttachInterface(int, const char *, int, int, struct SConnectionInfo *);
int SBAttachInterfaceExtended(int, const char *, int, int,
                              struct SConnectionInfo *, int, int, int,
                              notificationid_t *);
int SBGetServerInformation(int, const char *, int, int, struct SServerInfo *);
int SBAttachInterfaceTCP(int, const char *, int, int,
                         struct STCPConnectionInfo *);
int SBDetachInterface(int, SPartyID);
int SBSetServerAvailableNotification(int, const char *, int, int, int, int,
                                     int, notificationid_t *);
int SBSetServerDisconnectNotification(int, SPartyID, int, int, int,
                                      notificationid_t *);
int SBSetClientDetachNotification(int, SPartyID, int, int, int,
                                  notificationid_t *);
int SBClearNotification(int, notificationid_t);
int SBGetInterfaceList(int, struct SFNDInterfaceDescription *, int, int *);
int SBSetInterfaceListChangeNotification(int, int, int, int,
                                         notificationid_t *);
int SBMatchInterfaceList(int, const char *, struct SFNDInterfaceDescription *,
                         int, int *);
int SBSetInterfaceMatchChangeNotification(int, const char *, int, int, int,
                                          notificationid_t *);
void SBGetDSIVersion(struct SFNDInterfaceVersion *);
```

Name

SBOpen

Synopsis

```
// In header: <dsi/clientlib.h>


int SBOpen(const char * filename);
```

Description

Servicebroker Clientlib

The servicebroker clientlib provides a convenient interface to the servicebroker service. Opens the servicebroker and returns the handle to it.

Parameters:	**filename**	the absolute path to the servicebroker (usually '/srv/servicebroker')
Returns:	the handle to the servicebroker or -1 if an error occurred. see errno for details.	

Name

SBOpenNotificationHandle

Synopsis

```
// In header: <dsi/clientlib.h>


int SBOpenNotificationHandle();
```

Description

Returns a Unix socket acceptor to be used for accepting new notification clients. You must call listen on the returned handle.

Name

SBClose

Synopsis

```
// In header: <dsi/clientlib.h>

void SBClose(int handle);
```

Description

Closes the servicebroker handle.

Parameters: **handle** the servicebroker handle

Name

SBRegisterInterface

Synopsis

```
// In header: <dsi/clientlib.h>


int SBRegisterInterface(int handle, const char * ifName, int majorVersion,
                        int minorVersion, int chid, SPartyID * serverId);
```

Description

Registers a new interface at the servicebroker

Parameters:	**chid**	the channel id through which a client can connect to the interface
	handle	the servicebroker handle
	ifName	the interface name
	majorVersion	the major version of the interface
	minorVersion	the minor version of the interface
Returns:	EOK on success	

Name

SBRegisterInterfaceTCP

Synopsis

```
// In header: <dsi/clientlib.h>


int SBRegisterInterfaceTCP(int handle, const char * ifName, int majorVersion,
                           int minorVersion, uint32_t ipaddress,
                           uint16_t port, SPartyID * serverId);
```

Description

Experimental.

Parameters:	**ifName**	The normal interface name.
	ipaddress	in host byte order. If set to 0, localhost is assumed.
	port	in host byte order.

Name

SBRegisterGroupInterface

Synopsis

```
// In header: <dsi/clientlib.h>


int SBRegisterGroupInterface(int handle, const char * ifName,
                             int majorVersion, int minorVersion, int chid,
                             const char * groupName, SPartyID * serverId);
```

Description

Registers a new interface at the servicebroker that will only be accessible for members of a certain user group.

Parameters:	**chid**	the channel id through which a client can connect to the interface
	groupName	name of the user group that is allowed to attach to the interface (see /etc/group for user groups)
	handle	the servicebroker handle
	ifName	the interface name
	majorVersion	the major version of the interface
	minorVersion	the minor version of the interface
	serverId	A pointer to a SPartyID variable where to store the server handle in
Returns:	EOK on success	

Name

SBRegisterInterfaceEx

Synopsis

```
// In header: <dsi/clientlib.h>


int SBRegisterInterfaceEx(int handle,
                          const struct SFNDInterfaceDescription * ifDescr,
                          int descrCount, int chid, SPartyID * serverId);
```

Description

Registers multiple interface at the servicebroker

Parameters:

chid the channel id through which a client can connect to the interface

descrCount number of elements in ifDescr

handle the servicebroker handle

ifDescr array of **SFNDInterfaceDescription** structures

serverId A pointer to an array of SPartyID that must be at least descrCount in size. Each entry in this array receives the server id of the corresponding entry in the ifDescr array. If an error occurred while registering an interface the globalId of the server id is set to -1

Returns: EOK on success

Name

SBRegisterInterfaceExTCP

Synopsis

```
// In header: <dsi/clientlib.h>


int SBRegisterInterfaceExTCP(int handle,
                             const struct SFNDInterfaceDescription * ifDescr,
                             int descrCount, uint32_t ipaddress,
                             uint16_t port, SPartyID * serverId);
```

Description

Registers multiple interface at the servicebroker

Parameters:

descrCount number of elements in ifDescr

handle the servicebroker handle

ifDescr Array of **SFNDInterfaceDescription** structures (interface names stay as-is though the servicebroker will add a _tcp at the end of the names).

ipaddress In host byte order.

port In host byte order.

serverId A pointer to an array of SPartyID that must be at least descrCount in size. Each entry in this array receives the server id of the corresponding entry in the ifDescr array. If an error occurred while registering an interface the globalId of the server id is set to -1

Returns: EOK on success

Name

SBUnregisterInterface

Synopsis

```
// In header: <dsi/clientlib.h>


int SBUnregisterInterface(int handle, SPartyID serverId);
```

Description

Remove an existing interface from the servicebroker registry

Parameters:	**handle**	the servicebroker handle
	serverId	the id of the server
Returns:	EOK on success	

Name

SBAttachInterface

Synopsis

```
// In header: <dsi/clientlib.h>


int SBAttachInterface(int handle, const char * ifName, int majorVersion,
                      int minorVersion, struct SConnectionInfo * connInfo);
```

Description

Attach to an existing interface

Parameters:	**connInfo**	pointer to a **SConnectionInfo** structure that receives the information needed to connect to the interface
	handle	the servicebroker handle
	ifName	the interface name
	majorVersion	the major version of the interface
	minorVersion	the minor version of the interface
Returns:	EOK on success	

Name

SBAttachInterfaceExtended

Synopsis

```
// In header: <dsi/clientlib.h>


int SBAttachInterfaceExtended(int handle, const char * ifName,
                              int majorVersion, int minorVersion,
                              struct SConnectionInfo * connInfo, int chid,
                              int code, int value,
                              notificationid_t * notificationID);
```

Description

Attach to an existing interface. The function reacts distinctly depending on the availability of the interface:

- if interface available: returns connection information back and sets server disconnect notification
- if interface not available: sets a server available notification

Parameters:	**chid**	the channel id through which a client can connect to the interface
	code	the pulsecode of the notification
	connInfo	pointer to a **SConnectionInfo** structure that receives the information needed to connect to the interface
	handle	the servicebroker handle
	ifName	the interface name
	majorVersion	the major version of the interface
	minorVersion	the minor version of the interface
	value	the value of the notification
Returns:	EOK on success	

Name

SBGetServerInformation

Synopsis

```
// In header: <dsi/clientlib.h>


int SBGetServerInformation(int handle, const char * ifName, int majorVersion,
                           int minorVersion, struct SServerInfo * serverInfo);
```

Description

Starts a service broker query for the server providing the service interface.

Parameters:	**handle**	the servicebroker handle
	ifName	the interface name
	majorVersion	the major version of the interface
	minorVersion	the minor version of the interface
	serverInfo	pointer to a **SServerInfo** structure that receives the information needed to connect to the interface
Returns:	EOK on success	

Name

SBAttachInterfaceTCP

Synopsis

```
// In header: <dsi/clientlib.h>


int SBAttachInterfaceTCP(int handle, const char * ifName, int majorVersion,
                         int minorVersion,
                         struct STCPConnectionInfo * connInfo);
```

Description

Experimental.

Parameters:	**ifName**	The normal interface name.

Name

SBDetachInterface

Synopsis

```
// In header: <dsi/clientlib.h>


int SBDetachInterface(int handle, SPartyID clientId);
```

Description

Detaches a connected interface

Parameters:	**clientId**	the client id of interface (returned by SBAttachInterface)
	handle	the servicebroker handle
Returns:	EOK on success	

Name

SBSetServerAvailableNotification

Synopsis

```
// In header: <dsi/clientlib.h>


int SBSetServerAvailableNotification(int handle, const char * ifName,
                                     int majorVersion, int minorVersion,
                                     int chid, int code, int value,
                                     notificationid_t * notificationID);
```

Description

Set a notification that will be sent as soon as the specified server is available.

Parameters:	**chid**	the channel id of the channel that will receive the notification pulse. The pid is the pid of the calling process.
	code	the pulsecode of the notification
	handle	the servicebroker handle
	ifName	the interface name
	majorVersion	the major version of the interface
	minorVersion	the minor version of the interface
	notificationID	a pointer to an integer that receives the notification id. This id can be used to clear the notification.
	value	the value of the notification
Returns:	EOK on success	

Name

SBSetServerDisconnectNotification

Synopsis

```
// In header: <dsi/clientlib.h>


int SBSetServerDisconnectNotification(int handle, SPartyID serverID, int chid,
                                      int code, int value,
                                      notificationid_t * notificationID);
```

Description

Set a notification that will be sent when the specified server disconnects.

Parameters:	**chid**	the channel id of the channel that will receive the notification pulse. The pid is the pid of the calling process.
	code	the pulsecode of the notification
	handle	the servicebroker handle
	notificationID	a pointer to an integer that receives the notification id. This id can be used to clear the notification.
	serverID	the id of the server
	value	the value of the notification
Returns:	EOK on success	

Name

SBSetClientDetachNotification

Synopsis

```
// In header: <dsi/clientlib.h>


int SBSetClientDetachNotification(int handle, SPartyID clientID, int chid,
                                  int code, int value,
                                  notificationid_t * notificationID);
```

Description

Set a notification that will be sent when the specified client disconnects.

Parameters:	**chid**	the channel id of the channel that will receive the notification pulse. The pid is the pid of the calling process.
	clientID	the id of the client
	code	the pulsecode of the notification
	handle	the servicebroker handle
	notificationID	a pointer to an integer that receives the notification id. This id can be used to clear the notification.
	value	the value of the notification
Returns:	EOK on success	

Name

SBClearNotification

Synopsis

```
// In header: <dsi/clientlib.h>


int SBClearNotification(int handle, notificationid_t notificationID);
```

Description

Clear a notification.

Parameters:	**handle**	the servicebroker handle
	notificationID	the id of the notification to clear
Returns:	EOK on success	

Name

SBGetInterfaceList

Synopsis

```
// In header: <dsi/clientlib.h>


int SBGetInterfaceList(int handle, struct SFNDInterfaceDescription * ifs,
                       int inCount, int * outCount);
```

Description

Returns the list of registered interfaces from the servicebroker.

Parameters:	**handle**	the servicebroker handle
	ifs	pointer to an array of **SFNDInterfaceDescription** structs
	inCount	number of elements in the array ifs points to
	outCount	pointer to a variable that will receive the number of interfaces
Returns:	EOK on success or an error if the handle is invalid or the buffer is not big enough.	

Name

SBSetInterfaceListChangeNotification

Synopsis

```
// In header: <dsi/clientlib.h>


int SBSetInterfaceListChangeNotification(int handle, int chid, int code,
                                         int value,
                                         notificationid_t * notificationID);
```

Description

Set a notification that will be sent when a new service is available or if a service was removed. This notification is not a one shot notification. It will remain armed until it is cleared.

Parameters:	**chid**	the channel id of the channel that will receive the notification pulse. The pid is the pid of the calling process.
	code	the pulsecode of the notification
	handle	the servicebroker handle
	notificationID	a pointer to an integer that receives the notification id. This id can be used to clear the notification.
	value	the value of the notification
Returns:	EOK on success	

Name

SBMatchInterfaceList

Synopsis

```
// In header: <dsi/clientlib.h>


int SBMatchInterfaceList(int handle, const char * regExpr,
                         struct SFNDInterfaceDescription * ifs, int inCount,
                         int * outCount);
```

Description

Returns a list of registered interfaces from the servicebroker that matches a regulat expression.

Parameters:	**handle**	the servicebroker handle
	ifs	pointer to an array of **SFNDInterfaceDescription** structs
	inCount	number of elements in the array ifs points to
	outCount	pointer to a variable that will receive the number of interfaces that match the regular expression. If the regular expression could not be compiled this argument returns a negative value. See the servicebroker log for details in this case.
	regExpr	the regular expression (man regcomp).
Returns:	EOK on success or an error if the handle is invalid or the buffer is not big enough.	

Name

SBSetInterfaceMatchChangeNotification

Synopsis

```
// In header: <dsi/clientlib.h>


int SBSetInterfaceMatchChangeNotification(int handle, const char * regExpr,
                                          int chid, int code, int value,
                                          notificationid_t * notificationID);
```

Description

Set a notification that will be sent a newly registered interface matches the provides regular expression. This notification is not a one shot notification. It will remain armed until it is cleared.

Parameters:	**chid**	the channel id of the channel that will receive the notification pulse. The pid is the pid of the calling process.
	code	the pulsecode of the notification
	handle	the servicebroker handle
	notificationID	a pointer to an integer that receives the notification id. This id can be used to clear the notification.
	regExpr	the regular expression (man regcomp).
	value	the value of the notification
Returns:	EOK on success	

Name

SBGetDSIVersion

Synopsis

```
// In header: <dsi/clientlib.h>


void SBGetDSIVersion(struct SFNDInterfaceVersion * sbVersion);
```

Description

Get the dsi version of the clientlib

Parameters:	**sbVersion**	a pointer to the DSI version of the servicebroker clientlib
Returns:	EOK on success	

A.1.2 Header <dsi/private/platform.h>

```

_DCMD_MISC
__DIOF(kind, cmd, data)
__DIOT(kind, cmd, data)
__DIOTF(kind, cmd, data)
__DION(kind, cmd)
NAME_MAX
```

Name

_DCMD_MISC

Synopsis

```
// In header: <dsi/private/platform.h>

_DCMD_MISC
```

Name

__DIOF

Synopsis

```
// In header: <dsi/private/platform.h>

__DIOF(kind, cmd, data)
```

Name

__DIOT

Synopsis

```
// In header: <dsi/private/platform.h>

__DIOT(kind, cmd, data)
```

Name

__DIOTF

Synopsis

```
// In header: <dsi/private/platform.h>

__DIOTF(kind, cmd, data)
```

Name

__DION

Synopsis

```
// In header: <dsi/private/platform.h>

__DION(kind, cmd)
```

Name

NAME_MAX

Synopsis

```
// In header: <dsi/private/platform.h>

NAME_MAX
```

A.1.3 Header <dsi/private/servicebroker.h>

```

FND_SERVICEBROKER_ROOT
FND_SERVICEBROKER_PATHNAME
FOUNDATION_INVALID_PARTY_ID
DCMD_FND_REGISTER_INTERFACE
DCMD_FND_UNREGISTER_INTERFACE
DCMD_FND_ATTACH_INTERFACE
DCMD_FND_DETACH_INTERFACE
DCMD_FND_NOTIFY_SERVER_DISCONNECT
DCMD_FND_NOTIFY_SERVER_AVAILABLE
DCMD_FND_NOTIFY_SERVER_AVAILABLE_EX
DCMD_FND_NOTIFY_CLIENT_DETACH
DCMD_FND_CLEAR_NOTIFICATION
DCMD_FND_GET_INTERFACELIST
DCMD_FND_NOTIFY_INTERFACELIST_CHANGE
DCMD_FND_MATCH_INTERFACELIST
DCMD_FND_NOTIFY_INTERFACELIST_MATCH
DCMD_FND_REGISTER_INTERFACE_EX
DCMD_FND_REGISTER_INTERFACE_GROUPID
DCMD_FND_REGISTER_MASTER_INTERFACE_EX
DCMD_FND_MASTER_PING
DCMD_FND_MASTER_PING_ID
DCMD_FND_ATTACH_INTERFACE_EXTENDED
DCMD_FND_GET_SERVER_INFORMATION
```

```
union _SPartyID;

struct sb_pulse;
struct SFNDImplementationVersion;
struct SFNDInterfaceDescription;
struct SFNDInterfaceDescriptionEx;
struct SFNDChannelInfo;
struct SConnectionInfo;
struct SServerInfo;
struct STCPSocketInfo;
struct STCPConnectionInfo;
struct SFNDPulseInfo;

union SFNDInterfaceRegisterArg;
union SFNDInterfaceRegisterGroupIDArg;
union SFNDInterfaceRegisterExArg;
union SFNDInterfaceRegisterMasterExArg;
union SFNDInterfaceUnregisterArg;
union SFNDInterfaceAttachArg;
union SFNDInterfaceAttachExtendedArg;
union SFNDGetServerInformation;
union SFNDNotifyServerDisconnectArg;
union SFNDNotifyServerAvailableArg;

struct SFNDNotificationDescription;
struct SFNDNotificationID;

union SFNDNotifyServerAvailableExArg;
union SFNDInterfaceDetachArg;
union SFNDMasterPingIdArg;

struct SFNDMasterPingIdArgExt;

union SFNDNotifyClientDetachArg;
union SFNDClearNotificationArg;
union SFNDGetInterfaceListArg;
union SFNDNotifyInterfaceListChangeArg;
union SFNDMatchInterfaceListArg;
union SFNDNotifyInterfaceListMatchArg;

typedef uint32_t notificationid_t;
typedef int32_t dcmd_t;
typedef struct sb_pulse sb_pulse_t;
typedef enum _SBStatus SBStatus;  // Contains symbolc values for the status
returned by a devctl().
union _SPartyID __attribute__((aligned(8)));
```

Name

_SPartyID

Synopsis

```
// In header: <dsi/private/servicebroker.h>


union _SPartyID {

  // public data members
  uint32_t localID;
  uint32_t extendedID;
  struct _SPartyID::@0 s;
  uint64_t globalID;
};
```

Name

sb_pulse

Synopsis

```
// In header: <dsi/private/servicebroker.h>


struct sb_pulse {

  // public data members
  int32_t code;
  int32_t value;
};
```

Description

A pulse representation in socket mode.

Name

SFNDImplementationVersion — Describes an implementation version.

Synopsis

```
// In header: <dsi/private/servicebroker.h>


struct SFNDImplementationVersion {

  // public data members
  uint16_t majorVersion;  // The major version number of the implementation.
  uint16_t minorVersion;  // The minor version number of the implementation.
  uint16_t patchlevel;  // The patch level number of the implementation.
};
```

Name

SFNDInterfaceDescription — Describes an interface.

Synopsis

```
// In header: <dsi/private/servicebroker.h>


struct SFNDInterfaceDescription {

  // public data members
  struct SFNDInterfaceVersion version;  // The interface version.
  char name;  // The nul terminated interface name.
};
```

Name

SFNDInterfaceDescriptionEx

Synopsis

```
// In header: <dsi/private/servicebroker.h>


struct SFNDInterfaceDescriptionEx {

  // public data members
  struct SFNDImplementationVersion implVersion;  // The version of the
implementation to register.
  int32_t chid;  // The server channel.
  int32_t pid;  // The server process id.
  int32_t nd;  // The node id.
  SPartyID serverID;  // the local id on the slave.
  struct SFNDInterfaceDescription ifDescription;  // The null terminated
interface name.
};
```

Description

SFNDInterfaceDescriptionEx public public data members

1. `SPartyID serverID;`

   explicit padding here since serverID should be aligned on 8 bytes

Name

SFNDChannelInfo — Describes channel information.

Synopsis

```
// In header: <dsi/private/servicebroker.h>


struct SFNDChannelInfo {

  // public data members
  uint32_t nid;
  pid_t pid;
  int32_t chid;
};
```

Description

SFNDChannelInfo public public data members

1. `uint32_t nid;`

   The id of the node the process is running on

2. `pid_t pid;`

   The ID of the process.

3. `int32_t chid;`

   The channel ID.

Name

SConnectionInfo — Arguments returned from the service broker back to the client.

Synopsis

```
// In header: <dsi/private/servicebroker.h>


struct SConnectionInfo {

  // public data members
  struct SFNDInterfaceVersion ifVersion;
  struct SFNDChannelInfo channel;
  SPartyID serverID;
  SPartyID clientID;
};
```

Description

SConnectionInfo public public data members

1. `struct SFNDInterfaceVersion ifVersion;`

   The actual interface version.

2. `struct SFNDChannelInfo channel;`

   The channel information of the server.

3. `SPartyID serverID;`

   The ID of the server.

4. `SPartyID clientID;`

   Unique ID for the attached client.

Name

SServerInfo — Connection infomration for a server.

Synopsis

```
// In header: <dsi/private/servicebroker.h>


struct SServerInfo {

  // public data members
  struct SFNDChannelInfo channel;
  SPartyID serverID;
};
```

Description

SServerInfo public public data members

1. `struct SFNDChannelInfo channel;`

   The channel information of the server.

2. `SPartyID serverID;`

   The ID of the server.

Name

STCPSocketInfo

Synopsis

```
// In header: <dsi/private/servicebroker.h>


struct STCPSocketInfo {

  // public data members
  uint32_t ipaddress;  // in host byte order
  uint16_t port;  // in host byte order
};
```

Name

STCPConnectionInfo

Synopsis

```
// In header: <dsi/private/servicebroker.h>


struct STCPConnectionInfo {

  // public data members
  struct SFNDInterfaceVersion ifVersion;
  struct STCPSocketInfo socket;
  SPartyID serverID;
  SPartyID clientID;
};
```

Description

STCPConnectionInfo public public data members

1. `struct SFNDInterfaceVersion ifVersion;`

   The actual interface version.

2. `struct STCPSocketInfo socket;`

   The socket information of the server.

3. `SPartyID serverID;`

   The ID of the server.

4. `SPartyID clientID;`

   Unique ID for the attached client.

Name

SFNDPulseInfo — Describes pulse information passed for notifications.

Synopsis

```
// In header: <dsi/private/servicebroker.h>


struct SFNDPulseInfo {

  // public data members
  int32_t code;
  int32_t value;
  int32_t chid;
  int32_t pid;
};
```

Description

SFNDPulseInfo public public data members

1. `int32_t code;`

   The pulse code.

2. `int32_t value;`

   The pulse value.

3. `int32_t chid;`

   The id of the channel to send the pulse to.

4. `int32_t pid;`

   The id of process that owns the channel.

Name

SFNDInterfaceRegisterArg — Argument passed with the DCMD_FND_REGISTER_INTERFACE command.

Synopsis

```
// In header: <dsi/private/servicebroker.h>


union SFNDInterfaceRegisterArg {

  // public data members
  struct SFNDInterfaceVersion sbVersion;  // The service broker version.
  struct SFNDInterfaceDescription ifDescription;  // The description of the 
interface to register.
  struct SFNDImplementationVersion implVersion;  // The version of the 
implementation to register.
  int32_t chid;  // The server channel.
  int32_t pid;  // The server process id.
  struct SFNDInterfaceRegisterArg::@2 i;  // Arguments passed from the client 
to the service broker.
  SPartyID serverID;  // The unique server ID.
  struct SFNDInterfaceRegisterArg::@3 o;  // Arguments passed from the service 
broker to client.
};
```

Description

SFNDInterfaceRegisterArg public public data members

1. `struct SFNDInterfaceVersion sbVersion;`

   Clients have to provide the version of the actually used service broker in this element.

Name

SFNDInterfaceRegisterGroupIDArg — Argument passed with the DCMD_FND_REGISTER_INTERFACE_GROUPID command.

Synopsis

```
// In header: <dsi/private/servicebroker.h>


union SFNDInterfaceRegisterGroupIDArg {

  // public data members
  struct SFNDInterfaceVersion sbVersion;  // The service broker version.
  struct SFNDInterfaceDescription ifDescription;  // The description of the
interface to register.
  struct SFNDImplementationVersion implVersion;  // The version of the
implementation to register.
  int32_t chid;  // The server channel.
  int32_t pid;  // The server process id.
  int32_t grpid;  // The os user group id of this interface.
  struct SFNDInterfaceRegisterGroupIDArg::@4 i;  // Arguments passed from the
client to the service broker.
  SPartyID serverID;  // The unique server ID.
  struct SFNDInterfaceRegisterGroupIDArg::@5 o;  // Arguments passed from the
service broker to client.
};
```

Description

SFNDInterfaceRegisterGroupIDArg public public data members

1. `struct SFNDInterfaceVersion sbVersion;`

   Clients have to provide the version of the actually used service broker in this element.

Name

SFNDInterfaceRegisterExArg — Argument passed with the DCMD_FND_REGISTER_INTERFACE_EX command.

Synopsis

```
// In header: <dsi/private/servicebroker.h>


union SFNDInterfaceRegisterExArg {

  // public data members
  struct SFNDInterfaceVersion sbVersion;  // The service broker version.
  struct SFNDImplementationVersion implVersion;  // The version of the 
implementation to register.
  int32_t chid;  // The server channel.
  int32_t pid;  // The server process id.
  int32_t ifCount;  // number of interfaces.
  struct SFNDInterfaceRegisterExArg::@6 i;  // Arguments passed from the client 
to the service broker.
  SPartyID serverID;  // list of unique server IDs.
  struct SFNDInterfaceRegisterExArg::@7 o;  // Arguments passed from the 
service broker to client.
};
```

Description

SFNDInterfaceRegisterExArg public public data members

1. `struct SFNDInterfaceVersion sbVersion;`

   Clients have to provide the version of the actually used service broker in this element.

Name

SFNDInterfaceRegisterMasterExArg — Argument passed with the DCMD_FND_REGISTER_MASTER_INTERFACE_EX command.

Synopsis

```
// In header: <dsi/private/servicebroker.h>


union SFNDInterfaceRegisterMasterExArg {

  // public data members
  struct SFNDInterfaceVersion sbVersion;  // The service broker version.
  int32_t ifCount;  // number of interfaces.
  struct SFNDInterfaceRegisterMasterExArg::@8 i;  // Arguments passed from the
client to the service broker.
  SPartyID serverID;  // list of unique server IDs.
  struct SFNDInterfaceRegisterMasterExArg::@9 o;  // Arguments passed from the
service broker to client.
};
```

Name

SFNDInterfaceUnregisterArg — Argument passed with the DCMD_FND_UNREGISTER_INTERFACE command.

Synopsis

```
// In header: <dsi/private/servicebroker.h>


union SFNDInterfaceUnregisterArg {

  // public data members
  struct SFNDInterfaceVersion sbVersion;
  uint32_t reserved;
  SPartyID serverID;
  struct SFNDInterfaceUnregisterArg::@10 i;  // Arguments passed to the service
broker.
};
```

Description

SFNDInterfaceUnregisterArg public public data members

1. `struct SFNDInterfaceVersion sbVersion;`

   The service broker version.

2. `uint32_t reserved;`

   Reserved for future use.

3. `SPartyID serverID;`

   The ID of the server to unregister.

Name

SFNDInterfaceAttachArg — Argument passed with the DCMD_FND_INTERFACE_ATTACH command.

Synopsis

```
// In header: <dsi/private/servicebroker.h>


union SFNDInterfaceAttachArg {

  // public data members
  struct SFNDInterfaceVersion sbVersion;
  struct SFNDInterfaceDescription ifDescription;  // The interface description.
  struct SFNDInterfaceAttachArg::@11 i;  // Argument passed to the service
broker.
  struct SConnectionInfo o;  // Arguments returned from the service broker back
to the client.
};
```

Description

SFNDInterfaceAttachArg public public data members

1. `struct SFNDInterfaceVersion sbVersion;`

   The service broker version.

Name

SFNDInterfaceAttachExtendedArg — Argument passed with the DCMD_FND_ATTACH_INTERFACE_EXTENDED command.

Synopsis

```
// In header: <dsi/private/servicebroker.h>


union SFNDInterfaceAttachExtendedArg {

  // public data members
  struct SFNDInterfaceVersion sbVersion;
  struct SFNDPulseInfo pulse;
  struct SFNDInterfaceDescription ifDescription;  // The interface description.
  struct SFNDInterfaceAttachExtendedArg::@12 i;  // Argument passed to the
service broker.
  struct SConnectionInfo connInfo;
  notificationid_t notificationID;
  struct SFNDInterfaceAttachExtendedArg::@13 o;  // Arguments returned from the
service broker back to the client.
};
```

Description

SFNDInterfaceAttachExtendedArg public public data members

1. `struct SFNDInterfaceVersion sbVersion;`

   The service broker version.

2. `struct SFNDPulseInfo pulse;`

   Information about the pulse to send when the interface is available.

3. `struct SConnectionInfo connInfo;`

   The connection information for the server.

4. `notificationid_t notificationID;`

   A unique notification ID for the server available or disconnect notifications.

Name

SFNDGetServerInformation — Argument passed with the DCMD_FND_GET_SERVER_INFORMATION command.

Synopsis

```
// In header: <dsi/private/servicebroker.h>


union SFNDGetServerInformation {

  // public data members
  struct SFNDInterfaceVersion sbVersion;
  struct SFNDInterfaceDescription ifDescription;  // The interface description.
  struct SFNDGetServerInformation::@14 i;  // Argument passed to the service
broker.
  struct SServerInfo o;  // Arguments returned from the service broker back to
the client.
};
```

Description

SFNDGetServerInformation public public data members

1. `struct SFNDInterfaceVersion sbVersion;`

   The service broker version.

Name

SFNDNotifyServerDisconnectArg — Argument passed with the DCMD_FND_NOTIFY_SERVER_DISCONNECT command.

Synopsis

```
// In header: <dsi/private/servicebroker.h>


union SFNDNotifyServerDisconnectArg {

  // public data members
  struct SFNDInterfaceVersion sbVersion;
  uint32_t reserved;
  SPartyID serverID;
  struct SFNDPulseInfo pulse;
  struct SFNDNotifyServerDisconnectArg::@15 i;  // Arguments passed to the
service broker.
  notificationid_t notificationID;
  struct SFNDNotifyServerDisconnectArg::@16 o;  // Arguments passed from the
service broker to the client.
};
```

Description

SFNDNotifyServerDisconnectArg public public data members

1. `struct SFNDInterfaceVersion sbVersion;`

   The service broker version.

2. `uint32_t reserved;`

   Reserved for future use.

3. `SPartyID serverID;`

   The server ID of the server.

4. `struct SFNDPulseInfo pulse;`

   Information about the pulse to send when the server has disconnected.

5. `notificationid_t notificationID;`

   A unique notification ID.

Name

SFNDNotifyServerAvailableArg — Argument passed with the DCMD_FND_NOTIFY_SERVER_AVAILABLE command.

Synopsis

```
// In header: <dsi/private/servicebroker.h>


union SFNDNotifyServerAvailableArg {

  // public data members
  struct SFNDInterfaceVersion sbVersion;  // The service broker version.
  struct SFNDPulseInfo pulse;
  struct SFNDInterfaceDescription ifDescription;
  struct SFNDNotifyServerAvailableArg::@17 i;  // Arguments passed to the
service broker.
  notificationid_t notificationID;
  struct SFNDNotifyServerAvailableArg::@18 o;  // Arguments passed from the
service broker to the client.
};
```

Description

SFNDNotifyServerAvailableArg public public data members

1. `struct SFNDPulseInfo pulse;`

   Information about the pulse to send when the interface is available.

2. `struct SFNDInterfaceDescription ifDescription;`

   The interface description.

3. `notificationid_t notificationID;`

   A unique notification ID.

Name

SFNDNotificationDescription — Argument passed with the DCMD_FND_NOTIFY_SERVER_AVAILABLE_EX command.

Synopsis

```
// In header: <dsi/private/servicebroker.h>


struct SFNDNotificationDescription {

  // public data members
  uint32_t cookie;
  struct SFNDPulseInfo pulse;
  struct SFNDInterfaceDescription ifDescription;
};
```

Description

SFNDNotificationDescription public public data members

1. `struct SFNDPulseInfo pulse;`

   Information about the pulse to send when the interface is available.

2. `struct SFNDInterfaceDescription ifDescription;`

   The interface description.

Name

SFNDNotificationID

Synopsis

```
// In header: <dsi/private/servicebroker.h>


struct SFNDNotificationID {

  // public data members
  uint32_t cookie;
  notificationid_t notificationID;
};
```

Name

SFNDNotifyServerAvailableExArg

Synopsis

```
// In header: <dsi/private/servicebroker.h>


union SFNDNotifyServerAvailableExArg {

  // public data members
  struct SFNDInterfaceVersion sbVersion;  // The service broker version.
  int32_t count;  // size of the notificaion list
  struct SFNDNotifyServerAvailableExArg::@19 i;  // Arguments passed to the
service broker.
  struct SFNDNotificationID notifications;  // list of notification IDs.
  struct SFNDNotifyServerAvailableExArg::@20 o;  // Arguments passed from the
service broker to the client.
};
```

Name

SFNDInterfaceDetachArg — Argument passed with the DCMD_FND_DETACH_INTERFACE command.

Synopsis

```
// In header: <dsi/private/servicebroker.h>


union SFNDInterfaceDetachArg {

  // public data members
  struct SFNDInterfaceVersion sbVersion;
  uint32_t reserved;
  SPartyID clientID;
  struct SFNDInterfaceDetachArg::@21 i;
};
```

Description

SFNDInterfaceDetachArg public public data members

1. `struct SFNDInterfaceVersion sbVersion;`

   The service broker version.

2. `uint32_t reserved;`

   Reserved for future use.

3. `SPartyID clientID;`

   The ID of the client to detach.

Name

SFNDMasterPingIdArg

Synopsis

```
// In header: <dsi/private/servicebroker.h>


union SFNDMasterPingIdArg {

  // public data members
  struct SFNDInterfaceVersion sbVersion;  // The service broker version.
  uint32_t id;  // The extended id of the connecting slave.
  uint32_t reserved;  // Reserved for any purpose.
  struct SFNDMasterPingIdArg::@22 i;
};
```

Description

Authentication ping from slave to master servicebroker. When connecting a slave to the master the slave sends this ping at the beginning in order to authenticate itself with the id.

<xrefsect>
<xreftitle>Todo</xreftitle>
<xrefdescription>

This packet may probably moved into the 'AUTH' header which then must send its size so older server implementations may interact with newer clients and vice versa.
</xrefdescription>
</xrefsect>

Name

SFNDMasterPingIdArgExt

Synopsis

```
// In header: <dsi/private/servicebroker.h>


struct SFNDMasterPingIdArgExt {

  // public data members
  union SFNDMasterPingIdArg arg;
  uint32_t slaveIP;  // The peer IP of the slave servicebroker as seen from the
master.
  uint32_t masterIP;  // The masters local interface IP over which the slave
servicebroker is reachable.
};
```

Name

SFNDNotifyClientDetachArg — Argument passed with the DCMD_FND_NOTIFY_CLIENT_DETACH command.

Synopsis

```
// In header: <dsi/private/servicebroker.h>


union SFNDNotifyClientDetachArg {

  // public data members
  struct SFNDInterfaceVersion sbVersion;
  uint32_t reserved;
  SPartyID clientID;
  struct SFNDPulseInfo pulse;
  struct SFNDNotifyClientDetachArg::@23 i;  // Arguments passed to the service
broker.
  notificationid_t notificationID;
  struct SFNDNotifyClientDetachArg::@24 o;  // Arguments passed from the
service broker to the client.
};
```

Description

SFNDNotifyClientDetachArg public public data members

1. `struct SFNDInterfaceVersion sbVersion;`

   The service broker version.

2. `uint32_t reserved;`

   Reserved for future use.

3. `SPartyID clientID;`

   The ID of the client.

4. `struct SFNDPulseInfo pulse;`

   Information about the pulse to send when the client has been detached.

5. `notificationid_t notificationID;`

   A unique notfication ID.

Name

SFNDClearNotificationArg — Argument passed with the DCMD_FND_CLEAR_NOTIFICATION command.

Synopsis

```
// In header: <dsi/private/servicebroker.h>


union SFNDClearNotificationArg {

  // public data members
  struct SFNDInterfaceVersion sbVersion;
  notificationid_t notificationID;
  struct SFNDClearNotificationArg::@25 i;  // Arguments passed to the service
broker.
};
```

Description

SFNDClearNotificationArg public public data members

1. `struct SFNDInterfaceVersion sbVersion;`

   The service broker version.

2. `notificationid_t notificationID;`

   The ID of the notification to remove.

Name

SFNDGetInterfaceListArg — Argument passed with the DCMD_FND_GET_INTERFACELIST command.

Synopsis

```
// In header: <dsi/private/servicebroker.h>


union SFNDGetInterfaceListArg {

  // public data members
  struct SFNDInterfaceVersion sbVersion;
  uint32_t inElementCount;
  struct SFNDGetInterfaceListArg::@26 i;
  struct SFNDInterfaceDescription interfaceDescriptions;
  struct SFNDGetInterfaceListArg::@27 o;  // Arguments passed to the service
broker.
};
```

Description

SFNDGetInterfaceListArg public public data members

1. `struct SFNDInterfaceVersion sbVersion;`

   The service broker version.

2. `uint32_t inElementCount;`

   buffer element count

3. `struct SFNDInterfaceDescription interfaceDescriptions;`

   the element count within output, could be smaller than inElementCount

Name

SFNDNotifyInterfaceListChangeArg

Synopsis

```
// In header: <dsi/private/servicebroker.h>


union SFNDNotifyInterfaceListChangeArg {

  // public data members
  struct SFNDInterfaceVersion sbVersion;
  struct SFNDPulseInfo pulse;
  struct SFNDNotifyInterfaceListChangeArg::@28 i;  // Arguments passed to the
service broker.
  notificationid_t notificationID;
  struct SFNDNotifyInterfaceListChangeArg::@29 o;  // Arguments passed from the
service broker to the client.
};
```

Description

SFNDNotifyInterfaceListChangeArg public public data members

1. `struct SFNDInterfaceVersion sbVersion;`

   The service broker version.

2. `struct SFNDPulseInfo pulse;`

   Information about the pulse to send when the client has been detached.

3. `notificationid_t notificationID;`

   A unique notfication ID.

Name

SFNDMatchInterfaceListArg — Argument passed with the DCMD_FND_MATCH_INTERFACELIST command.

Synopsis

```
// In header: <dsi/private/servicebroker.h>


union SFNDMatchInterfaceListArg {

  // public data members
  struct SFNDInterfaceVersion sbVersion;
  uint32_t inElementCount;
  char regExpr;
  struct SFNDMatchInterfaceListArg::@30 i;
  struct SFNDInterfaceDescription interfaceDescriptions;
  struct SFNDMatchInterfaceListArg::@31 o;  // Arguments passed to the service
broker.
};
```

Description

SFNDMatchInterfaceListArg public public data members

1. `struct SFNDInterfaceVersion sbVersion;`

   The service broker version.

2. `uint32_t inElementCount;`

   buffer element count

3. `char regExpr;`

   the regular expression

4. `struct SFNDInterfaceDescription interfaceDescriptions;`

   the element count within output, could be smaller than inElementCount

Name

SFNDNotifyInterfaceListMatchArg

Synopsis

```
// In header: <dsi/private/servicebroker.h>


union SFNDNotifyInterfaceListMatchArg {

  // public data members
  struct SFNDInterfaceVersion sbVersion;
  struct SFNDPulseInfo pulse;
  char regExpr;
  struct SFNDNotifyInterfaceListMatchArg::@32 i;  // Arguments passed to the
service broker.
  notificationid_t notificationID;
  struct SFNDNotifyInterfaceListMatchArg::@33 o;  // Arguments passed from the
service broker to the client.
};
```

Description

SFNDNotifyInterfaceListMatchArg public public data members

1. `struct SFNDInterfaceVersion sbVersion;`

   The service broker version.

2. `struct SFNDPulseInfo pulse;`

   Information about the pulse to send when the client has been detached.

3. `char regExpr;`

   the regular expression

4. `notificationid_t notificationID;`

   A unique notfication ID.

Name

FND_SERVICEBROKER_ROOT

Synopsis

```
// In header: <dsi/private/servicebroker.h>

FND_SERVICEBROKER_ROOT
```

Name

FND_SERVICEBROKER_PATHNAME — The service broker pathname.

Synopsis

```
// In header: <dsi/private/servicebroker.h>

FND_SERVICEBROKER_PATHNAME
```

Name

FOUNDATION_INVALID_PARTY_ID

Synopsis

```
// In header: <dsi/private/servicebroker.h>

FOUNDATION_INVALID_PARTY_ID
```

Name

DCMD_FND_REGISTER_INTERFACE — Command to register an interface.

Synopsis

```
// In header: <dsi/private/servicebroker.h>

DCMD_FND_REGISTER_INTERFACE
```

Name

DCMD_FND_UNREGISTER_INTERFACE — Command to unregister an interface.

Synopsis

```
// In header: <dsi/private/servicebroker.h>

DCMD_FND_UNREGISTER_INTERFACE
```

Name

DCMD_FND_ATTACH_INTERFACE — Command to attach to an interface.

Synopsis

```
// In header: <dsi/private/servicebroker.h>

DCMD_FND_ATTACH_INTERFACE
```

Name

DCMD_FND_DETACH_INTERFACE — Command to detach an attached interface.

Synopsis

```
// In header: <dsi/private/servicebroker.h>

DCMD_FND_DETACH_INTERFACE
```

Name

DCMD_FND_NOTIFY_SERVER_DISCONNECT — Command to set a 'server disconnect' notification.

Synopsis

```
// In header: <dsi/private/servicebroker.h>

DCMD_FND_NOTIFY_SERVER_DISCONNECT
```

Name

DCMD_FND_NOTIFY_SERVER_AVAILABLE — Command to set a 'server available' notification.

Synopsis

```
// In header: <dsi/private/servicebroker.h>

DCMD_FND_NOTIFY_SERVER_AVAILABLE
```

Name

DCMD_FND_NOTIFY_SERVER_AVAILABLE_EX

Synopsis

```
// In header: <dsi/private/servicebroker.h>

DCMD_FND_NOTIFY_SERVER_AVAILABLE_EX
```

Name

DCMD_FND_NOTIFY_CLIENT_DETACH — Command to set a 'client detached' notification.

Synopsis

```
// In header: <dsi/private/servicebroker.h>

DCMD_FND_NOTIFY_CLIENT_DETACH
```

Name

DCMD_FND_CLEAR_NOTIFICATION — Command to clear a notification.

Synopsis

```
// In header: <dsi/private/servicebroker.h>

DCMD_FND_CLEAR_NOTIFICATION
```

Name

DCMD_FND_GET_INTERFACELIST — Command receive a serverlist.

Synopsis

```
// In header: <dsi/private/servicebroker.h>

DCMD_FND_GET_INTERFACELIST
```

Name

DCMD_FND_NOTIFY_INTERFACELIST_CHANGE — Command set a 'serverlist change' notification.

Synopsis

```
// In header: <dsi/private/servicebroker.h>

DCMD_FND_NOTIFY_INTERFACELIST_CHANGE
```

Name

DCMD_FND_MATCH_INTERFACELIST — Command receive a serverlist.

Synopsis

```
// In header: <dsi/private/servicebroker.h>

DCMD_FND_MATCH_INTERFACELIST
```

Name

DCMD_FND_NOTIFY_INTERFACELIST_MATCH — Command set a 'serverlist change' notification.

Synopsis

```
// In header: <dsi/private/servicebroker.h>

DCMD_FND_NOTIFY_INTERFACELIST_MATCH
```

Name

DCMD_FND_REGISTER_INTERFACE_EX — Command to register multiple interfaces at a time.

Synopsis

```
// In header: <dsi/private/servicebroker.h>

DCMD_FND_REGISTER_INTERFACE_EX
```

Name

DCMD_FND_REGISTER_INTERFACE_GROUPID — Command to register an interfaces with restricted access rights.

Synopsis

```
// In header: <dsi/private/servicebroker.h>

DCMD_FND_REGISTER_INTERFACE_GROUPID
```

Name

DCMD_FND_REGISTER_MASTER_INTERFACE_EX — Command to register an interface.

Synopsis

```
// In header: <dsi/private/servicebroker.h>

DCMD_FND_REGISTER_MASTER_INTERFACE_EX
```

Name

DCMD_FND_MASTER_PING — Slave SBs ping their master so connections would be kept alive.

Synopsis

```
// In header: <dsi/private/servicebroker.h>

DCMD_FND_MASTER_PING
```

Name

DCMD_FND_MASTER_PING_ID — Slave SBs ping their master in TCP mode with this package.

Synopsis

```
// In header: <dsi/private/servicebroker.h>

DCMD_FND_MASTER_PING_ID
```

Name

DCMD_FND_ATTACH_INTERFACE_EXTENDED — Command to set a 'server disconnect' notification.

Synopsis

```
// In header: <dsi/private/servicebroker.h>

DCMD_FND_ATTACH_INTERFACE_EXTENDED
```

Name

DCMD_FND_GET_SERVER_INFORMATION — Command to attach to an interface.

Synopsis

```
// In header: <dsi/private/servicebroker.h>

DCMD_FND_GET_SERVER_INFORMATION
```

Name

sb_pulse_t

Synopsis

```
// In header: <dsi/private/servicebroker.h>


typedef struct sb_pulse sb_pulse_t;
```

Description

A pulse representation in socket mode.

A.1.4 Header <dsi/version.h>

```


DSI_VERSION_MAJOR
DSI_VERSION_MINOR
DSI_SERVICEBROKER_VERSION_MAJOR
DSI_SERVICEBROKER_VERSION_MINOR
DSI_PROTOCOL_VERSION_MAJOR
DSI_PROTOCOL_VERSION_MINOR
```

```

struct SFNDInterfaceVersion;
```

Name

SFNDInterfaceVersion — Describes an interface version.

Synopsis

```
// In header: <dsi/version.h>


struct SFNDInterfaceVersion {

  // public data members
  uint16_t majorVersion;  // The major version number of the interface.
  uint16_t minorVersion;  // The minor version number of the interface.
};
```

Name

DSI_VERSION_MAJOR — The DSI major version number.

Synopsis

```
// In header: <dsi/version.h>

DSI_VERSION_MAJOR
```

Name

DSI_VERSION_MINOR — The DSI minor version number.

Synopsis

```
// In header: <dsi/version.h>

DSI_VERSION_MINOR
```

Name

DSI_SERVICEBROKER_VERSION_MAJOR — The DSI servicebroker major version number.

Synopsis

```
// In header: <dsi/version.h>

DSI_SERVICEBROKER_VERSION_MAJOR
```

Name

DSI_SERVICEBROKER_VERSION_MINOR — The DSI servicebroker minor version number.

Synopsis

```
// In header: <dsi/version.h>

DSI_SERVICEBROKER_VERSION_MINOR
```

Name

DSI_PROTOCOL_VERSION_MAJOR — The DSI protocol major version number.

Synopsis

```
// In header: <dsi/version.h>

DSI_PROTOCOL_VERSION_MAJOR
```

Name

DSI_PROTOCOL_VERSION_MINOR — The DSI protocol minor version number.

Synopsis

```
// In header: <dsi/version.h>

DSI_PROTOCOL_VERSION_MINOR
```

Glossary

D

DSI (SI)

The distributed service interface DSI in version 2 or greater is an inter-process communication protocol between several applications running on multiple platforms. DSI implementations exist for different operating systems like QNX, Linux or Windows, having different programming language bindings like C++, C and Java. As transport protocols DSI can use TCP, Unix domain sockets and QNX message passing.
See Also distributed service interface, DSI2, SI.