1.1 Purpose of this document 4

1.2 Scope 4

2 Overview 5

2.1 The DSI model 7

2.1.1 Hierarchy within the DSI model 7

2.1.2 Relations of the elements of the DSI DSI model 8

2.1.3 Illustration of DSI DSI elements of the model to Java language elements 9

3 DSI specification 10

3.1 instances on DSI 10

3.1.1 DSI Service instance 10

3.1.2 runtime instances 10

3.1.3 Special DSI Definitions 11

3.2 Communication mechanisms 12

3.2.1 Request-Response mechanism 12

3.2.2 Indications 15

3.2.3 Notification mechanism 16

3.2.4 Identification of methods 17

3.3 DSI service provider 19

3.3.1 Administrative Functions 20

3.3.2 Registration / Deregistriegung a DSI service listener 21

3.4 DSI service listener 23

3.5 DSI Service Directory 24

3.5.1 General Conventions 24

3.5.2 Registration of a DSI service provider 25

3.5.3 Registration of a DSI service listener 26

3.5.4 Supported OSGi features 27

3.6 Data Types 28

3.6.1 Basic data types 28

3.6.2 constants 28

3.6.3 enums 29

3.6.4 bitfields 29

3.6.5 Data Container 29

3.6.6 Arrays 30

3.6.7 Exceptions 30

3.6.8 Default values for data types 31

3.7 Utility classes and functions 32

3.8 Naming Conventions for DSI 33

3.8.1 General 33

3.8.2 DSI Service 33

3.8.3 constants 33

3.8.4 enums 34

3.8.5 bitfields 34

3.8.6 Variables (Member of data containers) 34

3.8.7 Functions 35

3.8.8 Function Parameters 35

3.8.9 Classes 35

3.8.10 packages 35

3.8.11 DSI DSI service providers and service listener 35

3.8.12 Examples 36

3.8.13 Regular Expressions 37

3.9 Conventions for Java code 38

3.9.1 General 38

3.9.2 files and directory structure 40

3.9.3 DSI service provider 41

3.9.4 DSI service listener 43

3.9.5 Data Container 45

10.3 Document Conventions 47

11.3 Troubleshooting 47

3.11.1 Basic Requirements 47

3.11.2 Error Classes 47

3.11.3 Types of error handling 48

3.12 Startup 50

3.12.1 General 50

3.12.2 package org.dsi.ifc.base 50

3.12.3 package org.dsi.ifc.startup 50

3:13 Threading 51

3.13.1 Basic Requirements 51

3.13.2 Requests 51

3.13.3 Responses and Updates 51

3.13.4 Number of threads 51

3.14 Performance Requirements 51

4 Description of procedures 52

4.1 Startup the DSI interface 52

4.1.1 Registering the Service Admin interfaces on OSGi 53

4.1.2 Registering the DSI service provider on OSGi 55

4.1.3 Initializing DSI service providers 57

4.2 Instance Management at run time 58

4.2.1 Registering a DSI service listening for request-response 58

4.2.2 Registering a DSI service listener for notifications / Updates 59

4.3 flow of a request / response 60

4.3.1 Simple Request (no response) 60

4.3.2 Simple Request-Response 61

4.3.3 request with status response and value-Notification 63

4.4 Indication expiry of a 64

4.5 expiration of a Notification 65

5 Change History 66

Appendix A References and terms 67

A.1 References 67

A.2 Terms used 68

A.3 Abbreviations 70

Appendix B List of Figures and Tables 71

**1 Introduction**

**1.1 Purpose of the document**

This document describes the concepts of DSI interface.

**1.2 scope**

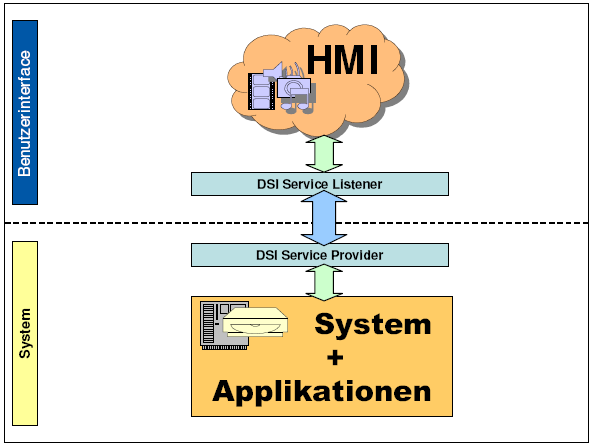
|  |  |
| --- | --- |
| Organizational: | MIB project |
| time: | For the duration of the project |

**2. Overview**

The Device Service Interface (DSI) defines the interface between the Human Machine Interface (HMI) and the underlying hardware or software applications. The DSI serves as an abstraction layer to allow access to the applications in the system by a device- and application-independent interface. The DSI is therefore also the boundary between the HMISite, which is located above the DSI and the system side, which is located below the DSI.

The DSI defines only the frame in which is a transport of data from the applications to the HMI, and vice versa. Conceptually, DSI provides two mechanisms to transmitted data to validate. Firstly, each update function has as the last parameter is a flag (the valid flag), which indicates whether the transmitted data is valid. ~~For others defines each request function two possible exceptions that can be thrown (DSIFatalExecption, DSINonFatalException).~~

These mechanisms validation of the data is possible. The specification is done, what conditions must comply with the transmitted data as well as the verification of compliance with these terms and as a notification for breach of the conditions is not part of the DSI concept and must be specified within the DSI Services.



<Figure 1: Overview of the DSI concept>

DSI is conceptually divided into two sections, the different issues on the HMI side and system side. On the HMI side there is a DSI service listener that receives the requested data and is informed of changes that occur independently of the HMI in the system (or in the respective area). At the system-side, a DSI service provider is defined, receives the requests from the HMI and the DSI service listener provides the desired data. In addition, data objects are specified with DSI that encapsulate related information.

Furthermore, the DSI is specified modular, that is, a partial specification includes a defined portion of the device functionalities provided by the (context). The classification of partial specifications is geared to the instrument's internal applications. The functions and objects defined for a subset are collectively referred to as DSI definition.

DSI DSI interface defines at least one pair per DSI-definition. Such a DSI interface pair, the so-called. DSI service consists of exactly one DSI service listener and a DSI service provider, however, several DSI Service instances (Chapter 3.1 see) at runtime may be present. Basically, here are all conceivable combinations of DSI service instance relationships (1: n, n: n, 1: 1, ...) are possible, for example, the coupling of certain tasks to individual DSI Service instances).

For used within a DSI definition of several DSI services constants (for enumeration and bit fields), it is also possible that a DSI definition contains a DSI interface constants. In this the constants used per DSI-definition shared by the various contained in the DSI-definition DSI Services are specified.

Furthermore, the DSI specified only the existing functions, but shall decide on the implementation of the functions no statement. The DSI thus corresponds to the "Programming-by-contract" model and defines the functions as Java interfaces or equivalent language constructs in other programming languages.

**2.1 The DSI model**

This chapter describes the concepts and relationships of the main structural elements of the DSI interface are explained. This is especially the overview, a detailed explanation or a fine structural division follows in subsequent chapters.

2.1.1 Hierarchy within the DSI model

Basically, the DSI is structured hierarchically, wherein the uppermost level represents the specification of the entire DSI interface for a particular infotainment project. The deeper levels of the hierarchy structure the DSI then into sections and sub-functions. Figure 2 shows the hierarchy levels in the form of nesting individual DSI elements.

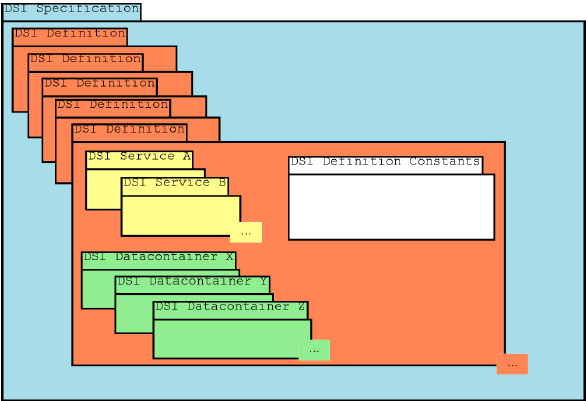


Figure 2: From the DSI concept proposed hierarchical levels

**2.1.1.1 DSI Meta Elements**

In the model of DSI, there are elements that can not be directly applied to an element of the programming language used. These are called the DSI meta model elements.

An example is an enum: The enumeration itself consists of several enumeration constants (which specifies each as a constant with the name and value as a language element of the programming language), these are represented by the parent name of the enum as belonging together.

**2.1.1.2 DSI Specification**

The DSI specification is the top element of the DSI (for a specific project infotainment) and defined in this context, the overall level of DSI interface.

**2.1.1.3 DSI-Definition**

A DSI-definition summarizes several related functions together, for example, the radio functionality with AM tuner, FM tuner, DAB tuner, SDARS tuner, ...

**2.1.1.4 DSI Service**

A DSI service specifies the functions and constants within the DSI for a clearly defined area of responsibility.

**2.1.2 Relationships of DSI elements of the model**

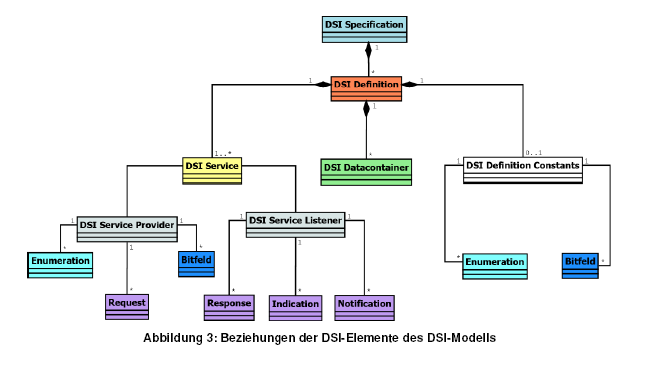


Figure 3: Relations between the elements of the DSI model

**2.1.3 Illustration of DSI elements of the model to Java language elements**

The following Table 1 shows the mapping of the main DSI elements on the language elements of Java.

**Table 1: Illustration of the basic DSI elements to elements of the Java programming language**

|  |  |
| --- | --- |
| **DSI element** | **Java element** |
| **DSI Specification** |  |
| **DSI-Definition** | **Java-Package** |
| **DSI-definition Constants** | **Java-Interface** |
| **DSI-Service** | **DSI meta element without mapping to Java language elements** |
| **DSI data container** | **Java class** |
| **DSI-Service-Provider** | **Java-Interface** |
| **DSI-Service-Listener** | **Java-Interface** |
| **DSI enumeration** | **DSI meta element without mapping to Java language elements** |
| **DSI enumeration constant** | **Java constant (declared as public static final)** |
| **DSI-bit field** | **DSI meta element without mapping to Java language elements** |
| **DSI-bit field constant** | **Java constant (declared as public static final)** |
| **DSI request function** | **Method (declared in Java interface of the DSI service provider)** |
| **DSI-Response-Funktion** | **Method (declared in Java interface of the DSI service provider)** |
| **DSI-Indication-Funktion** | **Method (declared in Java interface of the DSI service provider)** |
| **DSI-Notification-Funktion** | **Method (declared in Java interface of the DSI service provider)** |
|  |  |

**3 DSI specification**

Basic information relevant to any DSI Services functions, classes, constants, etc. are defined in the Java Package org.dsi.ifc.base.

**3.1 instances on DSI**

In connection with the DSI, the term instance is used in different contexts. The terms DSI Service instance and run-time instance be to allow a clear distinction for the DSI introduced. The meaning of these terms will be explained below.

**3.1.1 DSI Service instance**

A DSI Service instance corresponds to a logical or semantic partitioning of a DSI Services. The background is that there can be, for example, various address books (eg navigation addresses, phone book, ...) which use the same functions (DSI specification of the service) but work on other data.

To allow the system to assign the different DSI Service instances are registered with the OSGi own instance IDs. Thus, a semantic mapping is possible via the registration mechanisms by DSI.

**3.1.2 runtime instances**

The run-time instances corresponding to the known from object-oriented programming and thus represent instances created at runtime properties of certain classes.

**3.1.3 Special DSI definitions**

The DSI DSI concept exist two basic definitions that do not adhere to the rules applicable to all other DSI definitions and conventions themselves can. These are listed below and shown the differences.

**3.1.3.1 org.dsi.ifc.base: basic services and basic definitions**

In the special DSI-definition org.dsi.ifc.base all required by the DSI concept underlying mechanisms are provided. To this end, this DSI definition contains the following elements:

• Basisklassen, von denen die in einer DSI-Definition spezifizierten Klassen abgeleitet werden müssen

• Grundlegende Klassen für die Fehlerbehandlung

• OSGi-Dienste, die nicht einem DSI-Service entsprechen und grundlegende Dienste anbieten

All classes in this DSI definition may differ from the general conventions of the DSI DSI concept for a definition.

**3.1.3.2 org.dsi.ifc.global: Of all the DSI definitions usable data types**

The special DSI-definition org.dsi.ifc.global contains DSI data types that must be used by all definitions of a DSI DSI specification or can.

The only conventions violation of these special DSI DSI definition are the lack of services and the possibility of a DSI DSI constants interfaces for each context, all other conventions are adhered to by the DSI definition.

The data container classes org.dsi.ifc.global contained in the DSI and DSI definition of constants interfaces have a corresponding DSI-context name prefix (eg NavLocation or NavConstants).

**3.2 Communication mechanisms**

The communication via the DSI interface are generally asynchronous, the DSI interface defines two basic mechanisms, such as communication via the DSI interface is possible. The operation of these mechanisms will be explained in the following.

**3.2.1 request-response mechanism**

In this mechanism, the DSI service listener triggers a request for a particular function when DSIService- provider. Reasons for the initiation of a request by the DSI service listener can be:

• Changes to settings in the application (setting parameters)

• Triggering of certain functions (eg search for radio stations)

What reactions calling a Request a result, is specified in the respective DSI service for the request function based on the use cases. The possible reactions not limited to the functions (Responses, Notifications) that are defined within the respective DSI service, DSIübergreifende reactions are therefore also possible.

As typical reaction of the DSI service provider to request all to be notified when a DSI DSI Service instance of the service provider registered for the request-response mechanism DSI service listener via the corresponding response function. How to register for work described in Chapter 3.3.2.1 Responses are communicated to the DSI service listener on the response function What information is not determined by the DSI concept and will be specified in the respective DSI service.

The call to request functions continues by calling modalities (in the computer science called Reentrent- ability of a function) restriktiert. These define whether a request function may only be called if a previous call to the function or possibly other functions has not yet been processed by the system-side application. Table 2 shows the defined call-modalities.

**Table 2: Call modalities for request functions**

|  |  |
| --- | --- |
| Call-modality | importance |
| function-parallel | There are no restrictions in terms of calling Request.Funktionen with this modality |
| function-exclusive | A request function may only be called again when a previous call to this function is completed.  This is valid for a DSI Service instance. |
| cluster-exclusive | Here Request more functions are grouped into a cluster. Is a function of the cluster called all the other functions may not be invoked as long until the call has completed.  The cluster of interrelated functions here has a unique name within the DSI Services.  This is valid for a DSI Service instance. |
| instance-exclusive | After calling a request function of a DSI service all other functions of this request DSI Services may be called again after the call completes.  This is valid for a DSI Service instance. |

The concept defined by the DSI standard case is function-parallel.

From understanding forth the terms function-exclusive, cluster-exclusive and exclusive instance-to see primarily as a requirement for HMI-side implementation that needs to take account of these modalities. The modality function-parallel, primarily, is a requirement for the system-side application, since it assures the HMI degrees of freedom.

Be implemented on request functions with the exclusive modalities must for this one tracking that deals with a faulty calls. To this end, appropriate trigger must be specified for the request functions that serve as a signal for releasing a currently locked (because currently being processed) Request function. Signals that may indicate the completion of a request are:

• correct conclusion: the request is answered by specified response function

• regular erroneous conclusion: the request can not be executed regularly, because preconditions are not met for (this includes incorrect parameters). This is reported by asyncException () with the appropriate error code.

• regular rejection of a request: a request can not be performed because of its modality. This is reported by asyncException () with the error code ERROR\_REQUEST\_BUSY.

• Irregular erroneous conclusion: in the execution of a request unexpected error occurred that can also cause that none of the above feedback in the HMI arrive. For this purpose must be defined for the function of a timeout. After the timeout expires, the request call has failed.

~~As a guideline should functions that are specified with the exclusive modalities to be completed regularly in less than 100ms, the maximum timeout (including reserve) then is 1000ms, to keep the necessary waiting state in the HMI minimal. If for some longer-term functions timeouts longer necessary as a status response has to be defined which resets the timeout to view over longer time that a feature is still in progress. However, according to long-running functions should be an exception.~~

The implementation of the monitoring of call-by configuration procedures is switched on or off to implement.

Another requirement of the DSI concept is that a bad call behavior should not lead to a blocking of the DSI Services or the entire system.

**Note:** Functions marked as cluster-exclusive, this is an indication that these functions should be outsourced to a separate DSI service.

**3.2.2 Indications**

A special form of so-called Response is the indication. This is the call of a response, with no previous request has been triggered (also known as broadcast). Indications are specified as such and should be used only.

For the indications there are two different types. The first type of indications as a response only be shipped to the DSI Service listeners that have registered with the DSI DSI Service instance of the service provider that triggered the indication. An example of this is the already-defined basic concept asyncException (). An indication of this is automatically the first type if it is not explicitly marked otherwise.

The second type of Indications on the other hand, all registered DSI DSI Service listeners of all service instances shipped, no matter which DSI DSI Service instance of the service provider they have registered. They are labeled with the keyword all\_instances. An example of this is the org.dsi.ifc.organizer.DSIOrganizerListener.invalidData (...) method. This method must be shipped as an indication to all runtime instances for all DSI service instances of the DSI service listener as to manage the entries own DSI Service instance is specified. If an entry is deleted, this need to know all DSI Service instances, for example because the phone-context the deleted entry can no longer use for establishing a telephone connection.

Background is to ensure the notification of all running in the system run-time instances of the corresponding DSI service listener. The notification mechanism that is not possible, since the individual DSI service listener does not have to register for all update methods.

Example, the compounds of the audio management whose status changes (stopped, started, paused) in all cases to any existing audio management listeners to be communicated.

**3.2.3 Notification mechanism**

This mechanism is the implementation of a listener concept. To be informed about certain events or changes in system-side, the DSI service listener registered with the appropriate DSI service providers. If changes occur on the System page on which registered for this information DSI Service listeners are notified. As a DSI service listener registered for notification mechanism is described in Section 3.3.2.2.

It is possible to differentiate to register for specific information (attributes) for the implementations of the DSI service listener. There is always the opportunity to register for all attributes provided by the DSI Service provider, only for a part or individual attributes. For each attribute in the DSI service listener a matching function is defined, which is called by the DSI service provider for notification. The available attributes are defined in the DSI Service by enumerations (see also Section 3.2.4).

Addition to the parameters necessary for the specified notification update each of these functions an additional parameter indicating whether the transferred data are valid and can be used (the valid-flag). The valid flag is always the last parameter of an update function. The possible values for the valid flag and their meanings are shown in Table 3. Define the constants in the base interface for a DSI Service Listener (org.dsi.ifc.base.DSIListener).

A notification function can define several parameters for data transport and thus can have more than two parameters (data + Valid flag). However, this applies only for authorized by the DSI concept basic data types, enumerations and bit fields (see also chapter 3.6.1, 3.6.3, 3.6.4) and arrays of these types. The definition of more than one parameter whose type is a DSI data container is not permitted.

If a notification functions to update multiple data container classes as a separate transport stream container must be specified for this data to be updated at the same container. The same applies to the combination of a type parameter of a DSI data container with parameters of basic data types.

**Table 3: Possible values for the valid flag**

|  |  |  |
| --- | --- | --- |
| Valid-Flag | value | importance |
| ATTRVALIDFLAG\_UNKNOWN | 0x0 | The status of the transmitted data is unknown. |
| ATTRVALIDFLAG\_VALID | 0x1 | The transmitted data is valid |
| ATTRVALIDFLAG\_INVALID | 0x2 | The transmitted data is invalid.  An example of the use of this flag is when registering a DSI service listener for notification mechanism by setNotification (see Section 3.3.2.2). It may be that the Update method is called with invalid data, because currently there are no valid data. |

Furthermore, it is for the DSI service listener the opportunity to deregister according to the DSI service provider, ie to stop the alert. Again, this is again possible for all attributes, some or any of the available attributes per DSI service provider.

In addition, there are two methods for calling for the update functions. In "notify\_always" the update function is invoked (as it usually is here to Events), with "notify\_onChange" only if the data to be transmitted have also changed (compared to the last time the Notfication function). The use of "notify\_onChange" service to transfer the load on the DSI and the load in the HMI that is triggered by the receipt of the attributes to be minimized. The modality of an update function is defined statically (in the specification) and can not be configured.

**3.2.4 Identification of methods**

For unambiguous identification of methods each have their own constants are defined for the three types of functions that defines the DSI concept. The definition of the constants is performed per DSI service. The values of the constants are numbered for each DSI new service from the start of the range. The following table lists them with their properties.

Defines the constants for the identification of methods DSI DSI service provider of the respective service.

Table 4: Constants for the identification of methods

|  |  |  |
| --- | --- | --- |
| Name | description | range of values |
| Request constants | Identify a request function.  The name of the constants always starts with RT\_ (see also Section 3.8.3.1).  Use the find request constants as a reference when calling the asyncExecption () - function of the DSI service listener.  The defined for a DSI service request- constants are defined in the respective DSI service provider.  The special request RT\_NONE constant (with the value 0x0) is used if no request can be clearly assigned.  The constant RT\_NONE is defined in the base interface for a DSI Service Listener (org.dsi.ifc.base.DSIListener). | 1000…1999 |
| Response constants | Identify a response function.  The name of the constants always starts with RP\_ (see also Section 3.8.3.2).  Defined for a DSI Service Response constants are defined in respective DSI service provider. | 2000…2999 |
| Indication constants | Identify an indication function.  The name of the constants always starts with IN\_ (see also Section 3.8.3.3)  Defined for a DSI Service Indication constants are defined in respective DSI service provider. | 3000…3999 |
| Attribute Constants | Identify an update function.  The name of the constants always starts with ATTR\_ (see also Section 3.8.3.4).  The DSI defined for a service attribute constants are defined in respective DSI service provider. | 1…999 |

The assignment of the values of the request, response, Indication- and attribute constants does not take place-based methods, ie that the response constant is not necessarily linked with the number 2001 to the request with the request constant with the value 1001. An allocation over the last three digits is not possible.

**3.3 DSI-Service-Provider**

A DSI service provider is on the system page DSI interface. It defines the access capabilities through the HMI on the functionality provided by the applications. In addition to the functions that are specified according to the context for communication, for system administration functions (basis functions) are defined for each DSI service provider. The following Table 5 lists these functions. A detailed explanation of the functions is given in the following subsections.

Must continue to exist for testing and debugging purposes, access to the version of the implementation of each DSI service provider. How can happen to that information access is not explicitly defined here. Access must be uniform for all DSI service provider in a project but.

Table 5: Overview of the basis functions of the DSI service provider

|  |  |
| --- | --- |
| function | short Description |
| setNotification(…) | Methods for registering a DSI service listener at DSI Service Provider |
| clearNotification(…) | Methods to deregister a DSI service listener at DSI Service Provider |

**3.3.1 Administrative Functions**

**3.3.1.1 version of the DSI specification**

The version of the DSI DSI specifications of each service can be a VERSION constant determined (data type java.lang.String) of the associated DSI service provider. The version consists of up to four parts. Table 6 shows these components, valid values, and whether they are mandatory or optional.

The contents of the version contains constant succession by a point separated the parts MAJOR, MINOR, MICRO and the QUALIFIER (if specified). For numeric data no precision is required, it is therefore not padded with leading zeros. The content looks accordingly as follows (parts in square brackets "[]" are optional):

MAJOR.MINOR.MICRO [.QUALIFIER]

**Examples:** 

**Table 6: Valid values for the components of the version of the specification of a DSI Service**

|  |  |  |
| --- | --- | --- |
| component | modality | range of values |
| MAJOR | mandatory | 0 ≤ MAJOR ≤ – 1 (INT\_MAX) |
| MINOR | mandatory | 0 ≤ MINOR ≤ – 1 (INT\_MAX) |
| MICRO | mandatory | 0 ≤ MICRO ≤ – 1 (INT\_MAX) |
| QUALIFIER | optional | According to the specification of the OSGi version class  org.osgi.framework.Version:  ^[a-zA-Z0-9\_-]+$ |

The regular expression for the complete specification of the DSI Specification Version thus given by:

|  |
| --- |
| ^ [ 0 – 9 ] + \n. [ 0 – 9 ] + \n. [ 0 – 9 ] + ( \n. [ a – z A – Z 0 – 9 \_ - ] + ) ? $ |

**3.3.2 Registration / Deregistriegung a DSI Service Listener**

For the registration of a DSI service listener for a DSI service providers exist two mechanisms. Which one is used depends on which communication mechanism wants to register the DSI Service listeners that both mechanisms are explained below.

**3.3.2.1 registration Responses**

The registration / deregistration of a DSI service listener for a DSI service provider for the request-response mechanism via the OSGi framework. A DSI Service instance of a service provider receives a list of all registered for at DSI OSGi service listener and calls everyone for the corresponding DSI Service instance of the service provider on OSGi registered DSI Serivce listener the corresponding response function (broadcast). For another DSI DSI Service instance of the same service provider registered DSI Service listeners are not informed by Response. The registration of a DSI service listener is carried out for the request-response mechanism thus per DSI DSI Service instance of a service provider.

As the DSI service provider is established the current list of DSI service listener not prescribed by the concept, but this must be done with the opportunities provided by the OSGi implementation used (see also Section 3.5.4, in which the supported are listed OSGi features). Furthermore, it must be ensured that the DSI service provider knows a current list of registered DSI Service listener before calling the response.

**3.3.2.2 registering for notifications / Updates**

For a DSI service listener can register certain information relevant to him (attributes), the following functions are specified:

• setNotification(DSIListener)

• clearNotifcation(DSIListener)

• setNotification(Attribute, DSIListener)

• clearNotification(Attribute, DSIListener)

• setNotification(AttributeList, DSIListener)

• clearNotification(AttributeList, DSIListener)

The parameter attributes or attributes List (defined as an array in the interface specification) of the DSI service listener registered on an attribute or a number of attributes. For each attribute, the definition of the DSI service listener a method that is called by the DSI service provider for notification on this attribute.

Basically, a DSI service listener per attribute more than once added to the appropriate list of registered DSI service listener, even when the call is the setNotification () function repeatedly allowed. This means that the competent DSI service provider must consider a call to the method setNotification () if the passed in parameter DSI service listener is already registered for the attribute and the DSI service listener only the list of Add the attribute DSI registered service listener if this has not been registered yet.

The statement is to be registered for the attributes of the DSI service listener happens to the parameter attributes or attributes list, corresponding to a value or a list of values of an enumeration (the Attribute or Notification constants, see also chapter 3.2.3). If the method setNotification () or clearNotification) called (without specifying the attributes that instance passed the DSI service listener is registered for all attributes / unregistered.

If the method setNotification () or clearNotification () as a parameter an attribute that does not know the DSI service provider, a DSINonFatalException asyncException () with the request constants RT\_NONE is (specified in org.dsi.ifc.base. DSIListener) and the error code ERROR\_INVALID\_ATTRIBUTE thrown. This also applies to the value zero, which is also an invalid parameter. When an exception is thrown by the DSI service provider when you register a DSI service listener, the state of the list of registered DSI service listener is undefined.

The result of a call to one of the setNotification (...) functions always has a call to the relevant Notification / update function of the given DSI Serivce listener result. This serves to confirm the registration of the DSI service listener for the attributes. Furthermore, a call to the Notification / update Funtktion also occurs if the passed DSI service listener is already registered for the corresponding attributes. He thus has the opportunity to make explicit an update. However, a guaranteed way to get this returned valid data does not exist because zm time of the request currently no valid data may be available. If valid data, it is sent.

If the setNotification (...) function or clearNotification (...) is passed as a parameter an invalid attribute thrown a DSINonFatalException to signal an invalid parameters of the calling method.

**3.4 DSI Service Listener**

A DSI service listener defines the information presented on page HMI functions to which the DSI service provider can access for the flow of communication via the DSI interface.

Table 7: Overview of the basic functions of the DSI Service Listener

|  |  |
| --- | --- |
| function | short Description |
| asyncException() | The asyncException is part of the concept for error handling. This method is always used when due to the asynchronous operation of the DSI is a direct error feedback with an exception is no longer possible. This method is defined as indication and is sent to the registered for the DSI Service instance of the generating DSI DSI service provider service listener. |

The DSI service listener defines the function asyncException (), which informs to notify the HMI-side errors that have occurred during the execution of a request. For this purpose, the DSI service listener is notified in this function, the error code, a textual description of the error and the request function in which the error occurred.

The request function is passed as a request-constant, this serves as the ID of the function (see also section 3.2.4). In the event that the error is not attributable to any specific request function exists for these parameters in addition to the value "no request type" (RT\_NONE), which is specified in the interface org.dsi.ifc.base.DSIListener.

The possible error codes are defined in Section 3.9.

**3.5 DSI Service Directory**

The management of the entities involved in the DSI communicate via an OSGi-compliant directory service. The OSGi used corresponds to the OSGi specification version 3.0. These can be found in [5]. For performance reasons, the DSI concept but supports only a subset of the capabilities of the OSGi specification. Which they are described in Chapter 3.5.4.

To use the OSGi service directory are for DSI and DSI service provider service listener described in the following conditions.

**3.5.1 General Conventions**

OSGi required for registration in the OSGi service directory the following information:

• Parameter name: OSGi service name

• Parameter service: run-time instance of the to be registered OSGi service

• Parameter properties: collection of properties of the run-time instance of the OSGi services The properties must be stored as a string (the value for both the key as), and are therefore basically a string-string mapping. The only exception is for performance reasons the property of the instance ID, this is a string java.lang.Integer mapping so that parsing can be omitted.

The key to use names as constants in the interface org.dsi.ifc.base.DSIBase defined (DEVICE\_NAME, DEVICE\_INSTANCE).

As these parameters required by OSGi must be assigned values that describe the following chapters.

**3.5.2 Registration of a DSI service provider**

Registration is done OSGi-compliant on the method register service (java.lang.String name, java.lang.Object service, java.util.Dictionary properties) of the BundleContext with the following conditions for the parameters (Table 8).

Table 8: Parameters for the registration of a DSI service provider on OSGi

|  |  |
| --- | --- |
| Parameter | Conditions for the parameter |
| java.lang.String name | The name is the class name of the Java interfaces DSI DSI specification of the service provider of the corresponding DSI Services. The class name is conveyed via the corresponding object class of the Java interface with the getName () method.  **Example: AMFMTuner**  String name = org.dsi.ifc.amfmtuner.DSIAMFMTuner.class.getName () |
| java.lang.Object service | The transferred service is the run-time instance of the class that implements the Java interface of the corresponding DSI service provider.  **Example: AMFMTuner**  Object = new de.mib.dsi.impl.AMFMTunerImpl () |
| java.util.Dictionary properties | The properties are used to uniquely identify the DSI Serivce provider. For this purpose, the following entries must contain the transferred Dictionary:  1. Key: org.dsi.ifc.base.DSIBase.DEVICE\_NAME  Value: same as the parameter name  2. Key: org.dsi.ifc.base.DSIBase.DEVICE\_INSTANCE  Value: Instance ID (0 ... n) of the DSI service provider as  java.lang.Integer (DSI Service instance) |

**3.5.3 Registration of a DSI Service Listener**

Registration is done OSGi-compliant on the method

service register (String name, Object service, java.util.Dictionary dict)

the BundleContext with the following conditions for the parameters (Table 9).

Tabelle 9: Parameter für die Registrierung eines DSI-Service-Listeners am OSGi

|  |  |
| --- | --- |
| Parameter | Conditions for the parameter |
| java.lang.String name | The name is the class name of the base interfaces for each DSIListener (org.dsi.ifc.base.DSIListener). The class name is conveyed via the corresponding object class of the Java interface with the getName () method.  Example: AMFMTunerListener  String name = org.dsi.ifc.base.DSIListener.class.getName () |
| java.lang.Object service | The transferred service is the instance of the class that implements the Java interface of the corresponding DSI service listener.  Example: AMFMTuner  Object = new de.mib.dsi.impl.AMFMTunerImpl () |
| java.util.Dictionary dict | The properties are used to uniquely identify the DSI service listener. For this purpose, the following entries must contain the transferred Dictionary:  1. Key: org.dsi. i fc.base.DSIBase.DEVICE\_NAME  Value: Name of the specific DSI service listener through the class object of the jewei l sodium Java interface  (e.g. DSIAMFMTunerListener.class.getName ())  2. Key: org.dsi.ifc.base.DSIBase.DEVICE\_INSTANCE  Value: Instance ID (0 ... n) of the DSI service provider wishes to register for the DSI Service Listener as java.lang.Integer (DSI Service instance) |

**3.5.4 Supported OSGi features**

The data necessary for the operation of the DSI OSGi features are included in the HMI supplied OSGi implementation (LSD). The unsupported by the LSD features of the OSGi specification, which thus may not be used for the implementation of the DSI are included in the Java sources of the OSGi specification (Java source code of the interface definition) with the following listed in Table 10 JavaDoc tags in.

**Table 10: JavaDoc tags to identify the features supported by the LSD OSGi**

|  |  |
| --- | --- |
| JavaDoc-Tag | importance |
| @non.LSD | Marks non API functions / interfaces in LSD.  These methods / constants are not part of the runtime binary of the used OSGi implementation. Using this function May throw a java.lang.LinkageError During class loading. |
| @LSD.UnsupportedOperation | Marks unsupported methods in LSD but part of the API.  Calling synthesis methods will cause a  java.lang.UnsupportedOperationException. |

**3.6 Data Types**

The DSI makes no requirements concerning the data to be used types. The DSI, it is thus possible to define any data type (so-called. DSI data containers), if the transportable data can not be represented by a basic data type.

Should be noted, however, that the data types only serve as a transport medium, ie only encapsulate a lot of data. Content dependencies of data and consistency checks are not resolved by the data types themselves, but should be considered in the appropriate applications (an example of interdependent data would be a DSI data container for storing a radio station with the data "Name", "band", "transmitter frequency ", being not check whether the entered channel frequency matches the frequency band).

**3.6.1 Basic data types**

Basic data types are atomic and are already provided by the implementation language used Java. For DSI (corresponding Java) following basic data types are defined, the table is for information only case of doubt, the Java Language Specification (see [3]):

**Table 11: Overview of the basic data types**

|  |  |  |
| --- | --- | --- |
| Name | description | range of values |
| boolean | logical value | true, false |
| byte | Integer (8 bits) | -128 .. 127 |
| short | Integer (16 bits) | -32768 .. 32767 |
| Int | Integer (32 bits) | -2^31 .. 2^31 -1 |
| Long | Integer (64 bits) | -2^63 .. 2^63 -1 |
| Float | Single-precision floating-point number (32-bit) |  |
| Double | Single-precision floating-point number (64-bit) |  |
| String | Text (Unicode 2.1 16bit) | \n u0000 .. \n uFFFF |

**3.6.2 constants**

Constants define the time of implementation specified values. The definition of a constant always consists of the symbolic names of the constants and their values. Constants may be used only with their symbolic names.

All constants of a DSI service are defined in the DSI service provider. In exceptional cases constants may be defined in a separate interface class in DSI, for example, to increase the clarity for a large number of constants. The reasons for the exception are about to adequately document the class comment of the interface class.

**3.6.3 enumerations**

Enumerations are a group of constants, one of which is always exactly one may be used (eg as a parameter of a function call). The definition of an enumeration always consists of a symbolic name and several constants (enumeration). A combination of several enumeration is not permitted. If the enumeration by the implementation (such as Java 1.4 required) defined as constants are only allowed their names are not used with its value.

All enumerations of a DSI service are defined in the DSI service provider. In exceptional cases, individual enumerations may also be defined in a separate interface class in DSI, for example, to increase the clarity of a large number of enumeration. The reasons for the exception are about to adequately document the class comment of the interface class.

**3.6.4 Bit fields**

A bit field defines a collection of logic values (Flags), which are used as a single object. Bit fields are summarized logical values that can connect via bit-wise OR in an integer. Each bit in this case represents a flag and thus a parameter. The definition a bit field always consists of the symbolic name of the bit field and multiple constants (flags).

All bit fields of a DSI service are defined in the DSI service provider. In exceptional cases, individual bit fields may also be defined in a separate interface class in DSI, for example, to increase the clarity of a large number of enumeration. The reasons for the exception are about to adequately document the class comment of the interface class.

**3.6.5 Data Containers**

A data container is an object that defines a group of related data of any type. A data container is specified as a class and allows access to existing data (variables) only by access functions, not by direct access to the variables defined in the class (principle of encapsulation). For this purpose, the data container for each variable defines an access function for read access (getter function) and, optionally, a function for write access (setter function). The variables themselves are declared public for reasons of compatibility, a direct access to the variables, however, is not provided by the concept. By the runtime system, access to the contents of the variable can only be implemented via the corresponding getter functions. For additional software available in the runtime system is not at delivery (eg for code generators for the development or debugging tools), direct access is allowed on the variables.

Furthermore, two constructors for the production of an object at runtime are defined:

• a parameterless constructor, the data contained in the object initializes all with their default values. The default value is an appropriately for the particular application value is assigned in the body of the constructor. If no other default as defined, the default value for numeric values of 0, for boolean value false, and null for objects.

• a constructor, the values for all data stored on the class are passed in the call. The order of the parameters of the constructor corresponds to the order of the definition of the variables in Java source code.

Between variants in which the constructor is only passed a portion of the data to be stored in the class are not allowed.

In exceptional cases (eg for performance reasons or memory), it is necessary to store the data within the data container class in a different format. In this case, the internal data format and the necessary conversions are described in detail (this description must be sufficient for a non-expert developers to implement a conversion method). A definition of methods for the conversion o. Ä. directly in the data container class is not permitted. The only exception is the defintion of getter methods for the evaluation of individual fields of container transported by the DSI data bit fields. These are limited, however, to return a Boolean value which the information content of the corresponding bits of the bit field corresponds to (usually this is the masking of the transported bitwise AND and bit-field by checking the result on other than 0).

Optionally, a data container an overridden by java.lang.Object toString () provide that returns a string representation of the object and the stored values for debugging purposes. The format of the returned string is not defined.

Data containers are defined in the DSI service, which they are associated, but may be used by other DSI Services.

Furthermore, the data containers must not contain cyclic references, ie, a data container class XYZ must not contain any variable of another data container class ABC, if in the ABC class a variable of type XYZ is defined (this is the simplest case).

**3.6.6 Arrays**

Arrays are possible for the base data types as well as data containers used classes. The convention such as arrays in Java source code must be defined to Chapter 3.9.1.3.

It is accepted that in an array, individual elements than zero is transmitted if no valid data for individual elements. This can occur when, for a function, the array length is fixed.

**3.6.7 Exceptions**

The DSI Services can not define your own exceptions for error handling and must rely on the exceptions defined by the concept. For details see also Chapter 3.10.

**3.6.8 Default values for data types**

For the different data types DSI DSI defined concept different standard values used for specific applications. These are intended as a specification and are valid as long as these for the corresponding applications specifically allowed in the Javadoc documentation at the various DSI elements (functions, data containers) are specified differently (or equal to).

The hereafter defined default values are facing general criteria for performance (serialization / deserialization, message size, instantiation, ...) and memory usage selected (to avoid unnecessary memory Alloziierungen) and apply according to the above definition as long as no explicit specification the Javadoc documentation is present.

**3.6.8.1 Strings**

For strings standard values are defined for the following applications:

1. No valid data: data not available with the type string as data objects are transmitted as zero.

2. Empty strings: Empty strings are transmitted as strings of length 0 ..

**3.6.8.2 data container classes**

For data container standard values are defined for the following applications:

1. No valid data: if there are currently no valid data before and it should be called from other reasons, a function (eg, a response function as confirmation of a call to a request function) then the value zero is transmitted. This is also true for Notification functions (update methods on listener-DSI), the valid flag has a higher priority than a null parameter value. This means that with a "invalid" data regardless of their values are also invalid. In "valid" then the rule described above with zero values.

**3.6.8.3 arrays**

For arrays standard values are defined for the following applications:

1. Empty lists / arrays: If an empty list will be transferred (eg, no receivable radio stations available any more) this is done by means of the value zero. It is also possible for an array of length 0 to be transmitted as the treatment is usually the same as the null value. It is recommended to use null.

**3.7 Utility classes and functions**

For a DSI-definition, it is possible in addition to implement utility classes and / or functions. These are stored in a sub-package called ".util" of the respective DSI-definition-for example, in the package org.dsi.ifc.navigation.util for the DSI definition for navigation.

The implementation within this utility classes is not part of the respective DSI interface specification. An implementation of the interface must be without this utility classes and - be possible on the basis of interface documentation functions.

**3.8 Naming Conventions for DSI**

**3.8.1 General**

Basically naming conventions apply as specified by Sun Java (see [2]). In addition, existing or different conventions are described in the following chapters.

Names of classes, parameters, methods, etc. should always be meaningful (the purpose of it) and no information on the type of the calling object contained (no Hungarian notation such as strName for a string that particular name for Java interfaces begin NOT an "I" (capital "i")).

In Chapter 3.8.13 regular expressions are given that describe the naming conventions.

Furthermore, there are meta names that are also described herein. Meta names are names of DSI elements that are not directly mapped in Java elements but constitute the basis for naming conventions of other elements. This meta-names are featured in this chapter as follows: Service Name

**3.8.2 DSI Service**

The name of a DSI service (service name) begins with a capital letter, then follow lowercase. If the name consists of several parts, the first letter of each word part (CamelCase called.) Is capitalized. Come in the name abbreviations location (such as AM, FM, TV, DAB, ...), they are always written large.

**3.8.3 constants**

Constant names are written exclusively large, there is a name of several words, they are separated by an underscore.

Examples:

COUNTER\_MAX, LAST\_USER

**3.8.3.1 Request constants**

The name of a constant response basically follows the following pattern:

RT\_RequestName

Request Name is the name of the corresponding request function, the naming conventions for constants corresponding to upper case.

**3.8.3.2 Response constants**

The name of a constant response basically follows the following pattern:

RP\_ResponseName

Response name is the name of the corresponding response function, the naming conventions for constants corresponding to upper case.

3.8.3.3 Indication constants

The name of a constant indication generally follows the following pattern:

IN\_IndicationName

IndicationName is the name of the corresponding indication function, the naming conventions for constants corresponding to upper case.

3.8.3.4 Attribute Constants

The name of an attribute constant generally follows the following pattern:

ATTR\_ attributes Name

Attribute Name is the name of the corresponding attribute, the naming conventions for constants corresponding to upper case.

3.8.3.5 Error Codes

A constant name for an error code basically follows the following pattern:

ERROR\_FehlercodeName

Error code name is an error code descriptive name, the name of the constant overall compliance with the general conventions for constants.

3.8.4 enumerations

When naming enumerations both the enumeration itself and the possible values of the enumeration must be named. For this purpose, the following conventions:

• The name of the enumeration (the so-called. Enumerationsgruppe) always starts with a capital letter, the name consists of several parts, the first letter is written each subword large

• Possible values of an enumeration are named in uppercase as constants. The value name begins here followed by an underscore and a further meaningful description with the name of the enum (Enumerationsgruppe)

**Beispiel:**

|  |  |
| --- | --- |
| Name of the enum | Configuration |
| possible values | CONFIGURATION\_DEFAULT  CONFIGURATION\_SIMPLE  CONFIGURATION\_COMPLEX  CONFIGURATION\_VERY\_COMPLEX\_CASE\_WHICH\_SELDOM\_OCCURS |

3.8.5 Bit fields

For bit fields the same conventions as for enumerations apply.

3.8.6 Variables (Member of data containers)

The first letter is small, there is the name of several parts, the first letter (so-called CamelCase.) Is very important part of each word.

**3.8.7 functions**

The first letter is small, there is the name of several parts, the first letter (so-called CamelCase.) Is very important part of each word.

The name of getter functions always starts with a "get" (eg getStation ()) or Boolean values with "is" (example: isTMCStation ()), the name of setter functions always starts with a "set". Applies for update functions that these always begin with the literal "update", followed by the name of the corresponding attribute (AttributeName):

updateAttributeName

For request, response, and indication functions shall apply mutatis mutandis, that this can not begin with the literal "update".

The name of a cluster at request functions for the call-modality cluster-exclusive must also be defined as CamelCase beginning with a capital letter. Valid characters for this are the letters az, AZ and paragraph 0-9.

**3.8.8 Function Parameters**

For Function Parameters The same conventions as for functions apply.

**3.8.9 classes**

The first letter is capitalized, the name consists of several parts, the first letter (so-called CamelCase.) Is very important part of each word. In particular name for Java interfaces begin NOT with an "I" (capital "i").

**3.8.10 Packages**

Basically sensitive. Acceptable characters are all characters [a-z]. The package name is the name of the associated DSI service and always starts with org.dsi.ifc. A package name accordingly provides as follows:

org.dsi.ifc.ServiceName

wherein the ServiceName is lowercase.

**3.8.11 DSI service provider and DSI service listeners**

For DSI DSI service providers and service listener the same conventions as for classes apply. In addition, the structure of the class name is specified. The structure always follows the following pattern:

• For a DSI Service Provider: DSIServiceName

• For a DSI service listener: DSIServiceNameListener

where the term service name is the name of the DSI service.

**3.8.12 Examples**

**Table 12: Examples for the designation of classes for different DSI Services**

|  |  |  |  |
| --- | --- | --- | --- |
| DSI-Service | Package | DSI-Service-Provider  DSI-Service-Listener | Weitere Klassen |
| AMFMTuner | org.dsi.ifc.amfmtuner | DSIAMFMTuner  DSIAMFMTunerListener | RadioText  Station  WavebandInfo |
| Navigation | org.dsi.ifc.navigation | DSINavigation  DSINavigationListener | Route  RouteOptions  Location  … |

**3.8.13 Regular Expressions**

The regular expressions listed below in accordance with the conditions laid down in [4] syntax.

**Table 13: Regular Expressions for the naming conventions listed above**

|  |  |
| --- | --- |
| Java object | Regular expression |
| constants generally | ^[A-Z]{1}[A-Z0-9\_]+$ |
| Attribute Constants | ^ATTR\_[A-Z]{1}[A-Z0-9\_]+$ |
| Request clusters of functionality | ^[A-Z]{1}[a-zA-Z0-9]+$ |
| Request constants | ^RT\_[A-Z]{1}[A-Z0-9\_]+$ |
| Response constants | ^RP\_[A-Z]{1}[A-Z0-9\_]+$ |
| Indication constants | ^IN\_[A-Z]{1}[A-Z0-9\_]+$ |
| Version constant | ^VERSION$ |
| Bit field constant | ^[A-Z]{1}[A-Z0-9\_]+$ |
| Enumeration name | ^[A-Z]{1}[a-zA-Z0-9]+$ |
| Enumeration constant | ^[A-Z]{1}[A-Z0-9\_]+$ |
| Error code constant | ^ERROR\_[A-Z0-9\_]+$ |
| package | ^org\.dsi\.ifc\.[a-z]+$ |
| DSI-Service-Provider | ^DSI[A-Z]{1}[A-Za-z]+(?<!Listener)$ |
| DSI-Service-Listener | ^DSI[A-Z]{1}[A-Za-z]+Listener$ |
| classes generally | ^[A-Z]{1}[A-Za-z0-9]+$ |
| Request Features | ^(?!update)[a-z]{1}[A-Za-z0-9]+$ |
| Response functions | ^(?!update)[a-z]{1}[A-Za-z0-9]+$ |
| Indication functions | ^(?!update)[a-z]{1}[A-Za-z0-9]+$ |
| Update features | ^update[A-Z]{1}[A-Za-z0-9]+$ |

**3.9 Conventions for Java code**

This chapter only describes the conventions regarding the Java language syntax. In addition to those specified herein conventions are the basic "Java Code Conventions".

The files containing the Java source code (hereinafter Java sources) itself may be used in addition to the specification of the DSI as a basis for various Wider software and documentation that is either generated or extracted with tool support needed information from the Java sources. Compliance with the conditions laid down in this document and in [1] conventions is therefore imperative.

**3.9.1 General**

**3.9.1.1 Copyright Notice**

Any file that contains Java code that receives the file begins with a copyright notice, which looks like this:

|  |
| --- |
| /\*  \* Copyright (c) Volkswagen AG. All Rights Reserved.  \*/ |

**3.9.1.2 Return value of methods**

The communication via DSI is specified asynchronous. Thus, each method has a return value void.

**3.9.1.3 Definition of arrays**

Arrays are defined in the DSI, the brackets used to label must be given after the type of the array and not the variable name:

|  |
| --- |
| public void updateMyIntValueList(int[] intValueList);  public void updateMyDataContainerList(MyDataContainer[] dataContainerList); |

**3.9.1.4 Importing classes**

There are generally no complete packages imported, but explicitly the required classes.

**3.9.1.5 Structure of the Java source files**

A file containing Java source code basically consists of several areas. These areas should be separated by a blank line from each other. Optionally, just outside the area features a separate comment that identifies this area.

1. File comment:

In the file comment is that described in chapter 3.9.1.1 File comment.

2. package declaration (package statement):

This section contains the package statement according to Java. The name of the Java packages must transmit the naming conventions for a Java package match for a DSI service (see Section 3.8.10).

3. Import of other classes (import statement):

This area contains the corresponding Java import statement. In this case, that must be met conventions are described in Section 3.9.1.4.

4. class or interface definition:

in this area the class or interface is defined. The case to be followed conventions are described in the following chapters. Implicitly required in the definition is (see [1]) with documentation of the Java elements DSI documentation guidelines.

Basically just a class or an interface is defined in a file.

**3.9.2 files and directory structure**

**3.9.2.1 Files included in the DSI specification**

A DSI specification contains a JAR file with the compiled classes, a JAR file with the corresponding Java source code as well as a ZIP archive of the generated HTML documentation.

Listed in Table 14 are the files that contains a supply of a DSI specification.

**Table 14: Files and its content, including a supply of a DSI specification**

|  |  |
| --- | --- |
| file name | description |
| dsiifc.jar | JAR file containing the compiled Java classes (files with the extension  ".class") Contains. |
| dsiifc-src.jar | JAR-Archiv, das die Dateien mit dem Java-Quellcode der DSI-Spezifikation  (Dateien mit der Endung „.java“) enthält |
| dsiifc-api.zip | ZIP-Archiv, das die generierte Dokumentation (JavaDoc) der DSI-Spezifikation  enthält |

**3.9.3 DSI service provider**

A DSI service provider is specified as a Java interface, which extends the base interface for a DSI service provider (org.dsi.ifc.base.DSIBase). The following elements are defined in the DSI service provider of the respective service:

• Version constants

• Attribute Constants

• Request constant

• Response constants

• Indication constants

• Error codes

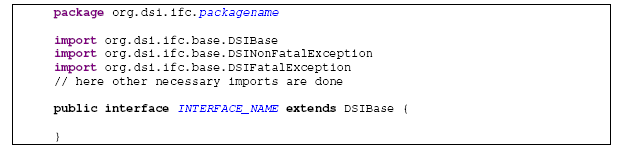
• Enumeration constants

• bitfield constants

• Request functions

**3.9.3.1 Definition of the interface**

Each DSI service provider is derived from the basic interface org.dsi.ifc.base.DSIBase. This has to be imported. Furthermore, the concept defined by the DSI exceptions (org.dsi.ifc.base.DSINonFatalException and org.dsi.ifc.base.DSIFatalException) must be imported because each request function defined in the throws this statement (see Section 3.9.3.3)



The name of the interface (INTERFACE\_NAME) must conform to the rules for a DSI service provider correspond (see Section 3.8.11).

The name of the Java package (package name) must conform to the rules for a Java package match for a DSI service (see Section 3.8.10).

**3.9.3.2 Definition of constants**

Each constant in the DSI is defined according to the following scheme:

|  |
| --- |
| public static final int CONSTANT\_NAME = CONSTANT\_VALUE; |

The name of the constant (CONSTANT\_NAME) must the defined constants for the respective type name conventions (see section 3.8).

The value of the constant is defined by constant\_value.

**3.9.3.3 Definition of request functions**

Each request function can trigger a DSINonFatalException or DSIFatalException by definition. This must be imported in the import-block (see also section 3.9.3.1).

|  |
| --- |
| public void METHOD\_NAME (PARAMETERS) throws DSINonFatalException,  DSIFatalException; |

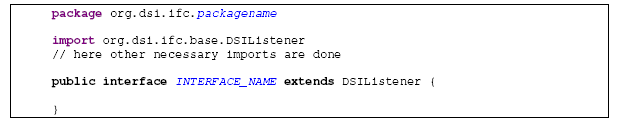
The name of the method (METHOD\_NAME) must comply with the conventions for a request function match (see chapter 3.8.7).

Defined by PARAMETERS Parameters of the function must match the conventions for function parameters (see Section 3.8.8).

**3.9.4 DSI Service Listener**

**3.9.4.1 Definition of the interface**

Each DSI service listener is derived from the base interface org.dsi.ifc.base.DSIListener. This has to be imported.



The name of the interface (INTERFACE\_NAME) must conform to the rules for a DSI service provider correspond (see Section 3.8.11).

The name of the Java package (package name) must conform to the rules for a Java package match for a DSI service (see Section 3.8.10).

**3.9.4.2 Definition of response functions**

Response functions are defined as follows:

|  |
| --- |
| public void METHOD\_NAME (PARAMETERS); |

The name of the method (METHOD\_NAME) must comply with the conventions for an update function match (see chapter 3.8.7).

Defined by PARAMETERS Parameters of the function must match the conventions for function parameters (see Section 3.8.8).

**3.9.4.3 Definition of Indication functions**

The definition of a response function which is used as the indication occurs identical to the definition of a normal response function (see section 3.9.4.2) and differs only in the type of documentation (see [1]).

**3.9.4.4 Definition of update functions**

Update functions are defined as follows:

|  |
| --- |
| public void updateAttributeName (ATTRIBUTES, int validFlag); |

The name of the method consists of the word update and the name of the attributes (name) for the update function is defined. The name of the method must be the conventions for an update function match (see Section 3.8.7)

ATTRIBUTES here are defined by the Update function parameters in the form TYPE attribute name.

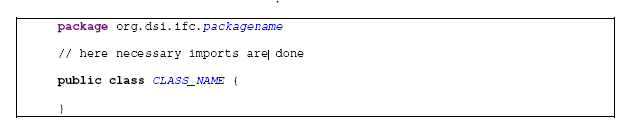
The attribute name is named the conventions for function parameters accordingly (see Section 3.8.8).

TYPE is the data type of the attribute.

**3.9.5 Data Containers**

**3.9.5.1 Definition of the class**

Data containers are defined in the package, which they are associated.



The name of the class (CLASS\_NAME) must comply with the naming conventions for classes (see Section 3.8.9).

The name of the Java package (package name) must conform to the rules for a Java package match for a DSI service (see Section 3.8.10).

**3.9.5.2 Definition of constructors**

Each data container contains two constructors. The first creates the object and initializes all variables with default values as for the variables of the respective DSI data container or specified with the following specified by the DSI concept default values if no explicit default value is specified:

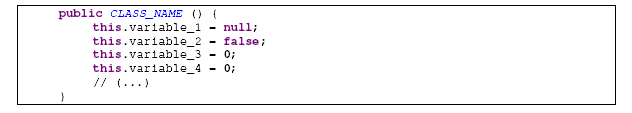
• Objects with zero

• numeric types with 0

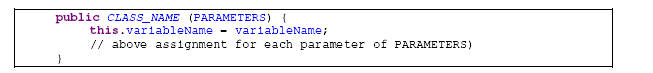
• Boolean variables to false

The second constructor initializes all variables passed in as parameter values.

Constructor for initialization with default values:



Constructor for initialization of all the variables:



The constructor contains a parameter for each declared in the class variable. The order of the parameters (PARAMETERS) corresponds to the order of the declaration of the variable in class. The name of the parameter is identical to the name of the declared variables (variable name)

**3.9.5.3 Definition of the variables (Member)**

Variables are declared public in data containers.



variable is the name of the variable. This must be the name conventions (see Section 3.8.6).

TYPE is the data type of the variable.

**3.9.5.4 Definition of getter methods**

For variables for all data types except boolean:



For variables with the data type boolean:



Variable is the name of the variable with a large initial letters of the name of the method overall compliance with the naming conventions for functions (see section 3.8.7).

TYPE is the data type of the variable.

**3.9.5.5 Definition of setter methods**

If in a data container for a variable a setter method is defined (Note: this is optional), this is defined as follows



Variable name and variable name is the name of the variable. As parameters according to the naming conventions for function parameters (see Section 3.8.8), as part of the method name with a large initial letters so that the name of the whole method meets the naming conventions for functions (see section 3.8.7) corresponds.

TYPE is the data type of the variable.

**3.10 Document Conventions**

The conventions for the static and dynamic documentation are described in another document (see [1]).

**3.11 Troubleshooting**

~~Basically, any request function of a DSI service provider with the possible exceptions DSIFatalException, DSINonFatalException declared (for more information see Table 15). If errors occur that must not be treated within the request function is triggered one of the two exceptions depending on the severity. Is it no longer possible to create one of the two exceptions and to the calling instance return (eg because of errors in a further called below the DSI asynchronous function occurs),~~

~~Invalid parameters must be checked immediately by the called request function and reported back by DSINonFatalException. This applies only to the static definition of the parameters. If certain parameters only valid in a particular context (of substance dependence), this will be reported to the mechanisms described above back as soon possible (when known) accordingly.~~

**3.11.1 Basic Requirements**

Basically for the implementation of the DSI, the demand for robustness, ie the errors may not lead to running threads are terminated. That must each thread in connection with the DSI Catching errors, handle the error, and then continue to run normally.

**3.11.2 Error classes**

At DSI different classes of errors can occur. Specifically, these are:

• Run-time error that must be corrected for the implementation of time (to be corrected in the source code of software errors, called bugs). Examples are calling a function with an invalid parameter, NullPointerExceptions

• Run-time error that can not be handled by the HMI. These include, for example, I / O exceptions, error when serializing / deserializing, OutOfMemoryExceptions.

**3.11.3 Types of error handling**

**3.11.3.1 Logging in the transport layer**

If errors within the transport layer to be logged in using these laid down by the DSI org.dsi.ifc.base.DSIException. The generated this log message contains a minimum of the Java stack trace of the exception that occurred, the name of the called request function and the parameters with which the request function was called. The packaging of the error in the DSIException is required to search within the logs after these errors (eg, using tools such as grep) and be able to filter them. The output of the log message is done in the standard log-channel of the device for the development time must log depression be configurable (static, eg via properties file).

Note: To log in as an appropriate method, all parameters of a function call itself offers introspection in Java for example via the Reflection API, since the treatment of this error by logging is not time-critical .. An appropriate implementation includes a few hundred lines of code, a name resolution Logging is not required, this can be done offline in a further step in the generated log (post-processing).

**3.11.3.2 feedback via asyncException ()**

For reporting errors in the processing of a request by return of the function call of the request function (eg in case of errors in the native part of the system-side application) the asyncException (...) function of the DSI service listener is used (defined in the base DSI interface for a service listener org.dsi.ifc.base.DSIListener). This function is an indication function by definition and is therefore called on all registered listeners DSI Service (broadcast).

This general error codes and their meanings are defined in the interface org.dsi.ifc.base.DSIError.

It is also possible for a DSI service to define your own error codes. These are defined as constants of type int in the DSI service provider. The value of the constant must be positive and greater than 10,000. The name of the constant must comply with the conditions laid down in section 3.8.3.5 naming conventions.

**Table 15: From the DSI concept defined exceptions**

|  |  |
| --- | --- |
| Exception | description |
| DSIFatalException | A non-critical error is present when the error does not affect the functionality of the device, ie that the system further is in a consistent and stable state. One reason for a non-critical error would, for example, the call of a request with an invalid parameter.  Excpetion thrown if an error occured which does not cause a malfuntion of the DSI framework. For example this exception is thrown when a request is called with a out-of-range or non-applicable parameter. A domain throwing this exception should still be in a konsistent and usable state. |
| DSINonFatalException | A critical error has occurred when the system unit by the error in an inconsistent state, so that the device can not be used and a restart or reset is necessary.  Exception thrown if an error occured, which causes malfunction in the DSI framework. This usually means that a domain (e.g. AMFMTuner) crashed and can only be reactivated be rebooting the system. |

**3.12 Startup**

**3.12.1 General**

The concepts described herein are intended to give a brief overview of the startup of the DSI interface. They serve only to get an overview. The exact specifications in this regard are described in external documents (see also [6], [7], [8]).

**3.12.2 package org.dsi.ifc.base**

To manage the DSI service provider at runtime the special service exists Service Admin (~~org.dsi.ifc.boot.DSIBoot) JDSIAdmin~~. This offers the possibility of run-time system-side

Implementation of a DSI Services (DSI service provider) to start, stop and restart. Run a DSI service provider initially only means that the instance is registered in the OSGi. In how far this then initialized and ready depends on the start-up concept, particularly whether the DSI service belongs to a domain specified (see Section 3.12.3). DSI services that are not domain members must be initialized by starting with the Service Admins by the JDSIAdmin.

**3.12.3 package org.dsi.ifc.startup**

The DSI service startup offers the possibility to control the startup from the HMI. For this purpose, different DSI Services can be grouped into so-called domains. These domains can then be moved to different intermediate states and thus initialized gradually.

The background is the ability to control the boot sequence dynamically from the HMI, are in the first DSI unnecessary services not yet fully loaded / initialized. Thus, the time for the start of the HMI (to be operated by the user) decreased.

What DSI Services are grouped into domains which, as the dependencies are resolved among themselves which intermediates are defined for the domain and other questions are related to the the Startup specified in external documents (see [7], [8]).

**3:13 Threading**

**3.13.1 Basic Requirements**

For the threading represents the DSI concept a few basic requirements:

• messages within a DSI Services may be not overtake

• It can also blocking calls from request functions exist

**3.13.2 requests**

Requests must be thread-safe. However, it must limitations of the call sequence or the speed done with the call sequences exist. Thus, for example, be ensured in the implementation in MMI3G, simultaneous requests to the DSIAMFMTuner serialized correctly and sent to the native application, but the application can provide inconsistent results due to rapid sequences of Views tuneWaveband and select station.

**3.13.3 Responses and Updates**

Responses and Updates may not be made in the HMI thread but only from the DSI thread. When calling a request from the HMI thread may, for example, not be the response in the same thread, but the request method must return immediately. There must be at least one thread for the DSI as a whole, but it may also, for example, give one thread per DSI service.

It is permissible to call a Response or update method that is called by the DSI, again one or more requests. Requests may therefore be made from the HMI and the DSI thread, whereas responses and updates may only be from the DSI thread.

**3.13.4 Number of threads**

According to the requirements of Section 3.13.1 follows that there must be at least two threads for the entire DSI DSI interface and per-service maximum one thread.

A possible implementation of the requirements of the DSI concept of threading would be a thread pool. This is to be evaluated as a proposal and not a requirement.

**3.14 Performance Requirements**

The DSI concept itself does not specify requirements on the performance of the DSI interface. Defines the performance requirements in the specifications of the individual DSI Services (in the Functional Requirements, Use Cases, dynamic processes). In any form (eg, timeouts, maximum waiting time for an update, etc.) requirements which remain placed on the performance is not fixed.

The conditions in which must pass the proof of the required performance is also defined by the individual DSI Services.

**4 Description of the processes**

For the procedures explained here, the fictitious DSI Weather Service is used for managing the current weather data from a weather station installed.

**4.1 Startup the DSI interface**

The startup of the DSI interface uses the Admin service from the package org.dsi.ifc.base. Here in note form the sequence:

1. Download the Jarfiles with the OSGi compliant manifest (specify the Bundle Activator, etc.)
2. Go to the Bundle Activator, which registers the implementation of the Service Admin on OSGi OSGi Service

3. The HMI initiated the registration of the individual DSI Services on OSGi using the determined from the OSGi Service Admin using the method start service (String name, int instance). This method does only the instantiation of the implementation of the corresponding DSI service and registers the instance on OSGi with the corresponding properties. When instantiating the DSI service implementations Only the minimum necessary initializations make (exactly what they must be specified for each DSI service, among other things, here also play things like "Early services," a role). In particular, a DSI service at instantiation assume to be able to use any other DSI service.

4. The HMI starts each DSI Services gradually using the DSI Service Startup (org.dsi.ifc.startup). This DSI Services start up in the currently required order (the order is again not fixed but dynamic and depends for example. from the last device state (load mode) before switching off from) and in the required forms (for time optimization of starting DSI individual services can be started only in part, to launch the full functionality later).

Points 1 and 2 are preparation points for the DSI. Applies to points 3 and 4, that it does not have to be carried out sequentially, but registering and starting the DSI Services according to the requirements can be adapted to the startup. That can not yet be registered that certain DSI Services already registered and started during others.

Name and location of Jarfiles are configured in a properties file. Name and location of these properties file and key name and valid values (or format of the values) must be specified.

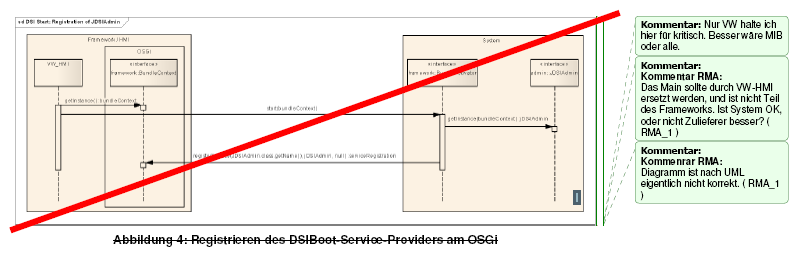
**4.1.1 Registering the Service Admin interfaces on OSGi**

Sign up for the implementation of the OSGi Service Admin carried compliant on the method register service (java.lang.String name, java.lang.Object service, java.util.Dictionary properties) of the BundleContext with the conditions listed in Table 16 for the parameters. Figure 4 shows the process of registering the Service Admin Deployment on OSGi~~. In order for the Service Admin can start the system-side implementation of the individual DSI Service Provider (DSI service provider) that must save the passed to it BundleContext locally.~~

The individual DSI Services themselves are NOT started automatically by the system-side implementation of the DSI. The start of each DSI Services is controlled by the HMI (with the aid of the implementation of the Service Admins).

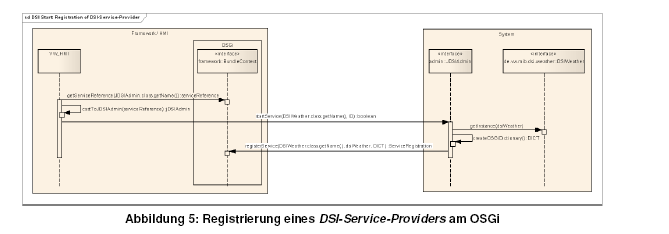
**Table 16: Parameters for the registration of the Service Admin on OSGi**

|  |  |
| --- | --- |
| Parameter | Conditions for the parameter |
| java.lang.String name | The name is the class name of the service admin interface org.dsi.ifc.base.ServiceAdmin. The class name will be on the associated object class of the Java interface with the method  getName () returns:  String name = org.dsi.ifc.base.ServiceAdmin.class.getName() |
| java.lang.Object service | The transferred service is the instance of the Implemetation of Service Admins.  example:  Object = new de.mib.dsi.impl.base.DSIServiceAdminImpl () |
| java.util.Dictionary properties | The transferred dictionary must contain the following entries:  1. Key: org.dsi.ifc.base.DSIBase.DEVICE\_NAME  Value: class name of the Service Admin  org.dsi.ifc.base.ServiceAdmin.class.getName ()  2. Key: org.dsi.ifc.base.DSIBase.DEVICE\_INSTANCE  Value: Instance ID of the Service Admin. This is always 0, because there can only be one instance of the Service Admin. |



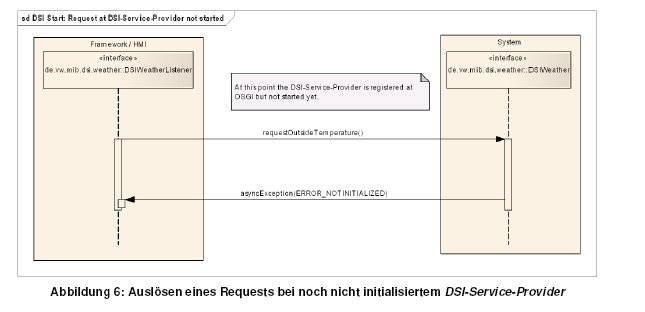
**4.1.2 Registering the DSI service provider on OSGi**

The Sign of the DSI service provider on OSGi is done from the main program. For this purpose, the main program identifies the most OSGi registered instance of JDSIAdmin (getServiceReference ()), the necessary parameters are derived from the description of the registration of the JDSIAdmin on OSGi (see Section 4.1.1).

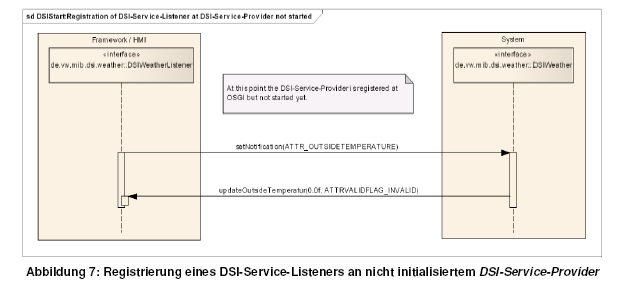
The main program then prompts the instance of JDSIAdmin the registration of the requested DSI service provider on OSGi using the method start service (...) of the JDSIAdmin. The order will be registered in the DSI service provider is not fixed. In particular, it is also possible to make certain DSI service provider not initially register (as they are initially not required) and to do so at a later time. The expiry of the registration of a DSI service provider on OSGi shows figure 5. 

It does not specify that initially all DSI service provider must be registered on OSGi before the DSI service provider be started (see Section 4.1.3), ie, a DSI service provider can already be started while another is not yet registered on the OSGi.

A registered in the OSGi DSI Service Provider is not functional for use, ie Although it can be triggered requests, however, since the underlying hardware has not yet started, the answer is a asyncException with the error code ERROR\_NOTINITIALIZED (see Figure 6).



However, the registration of a DSI service listener by setNotification () is already possible that the data provided with the update not valid are marked with the validFlag the value ATTRVALIDFLAG\_INVALID or ATTRVALIDFLAG\_UNKNOWN (the exact behavior is in each DSI Services specified) the registration of a DSI service listener on a not yet initialized DSI service provider is shown in Figure. 7



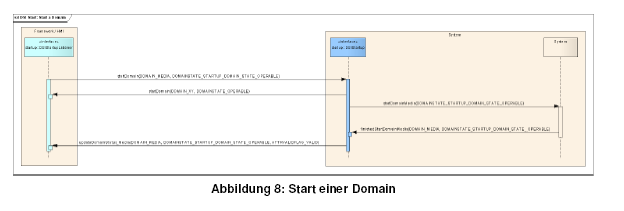
The initialization of the DSI service provider, refer to Chapter 4.1.3).

**4.1.3 Initializing DSI service providers**

The initialization of the DSI service providers can occur in two forms. If the DSI service (and thus the DSI service provider) part of a domain, a phased initialization can be performed according to the start of the domain. If the DSI service is not part of a domain, it must be initialized when you register on OSGi. For details, see section 3.12.

For starting a domain of special DSI Service Startup (org.dsi.ifc.startup) is used. The main program uses the corresponding OSGi mechanisms, the instance of the DSI service provider org.dsi.ifc.startup.DSIStartup and calls on this domain the request start (...) on. The case passed parameters specify the domain you are starting (parameter domainID) and the desired initialization of the domain (parameter desiredDomainState). The call is acknowledged with the response start domain (domain ID, currentDomain state).

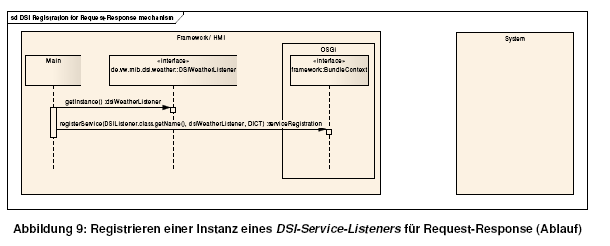
Then for the domain of the State domain started the native applications and initialized according to the specifications defined for the domain and the domain state (the requirements are for the domain in external documents defined) on the system side. Finally, the State now reached domain is also reported by notification to registered DSIStartupListener.



**4.2 Instance Management at runtime**

**4.2.1 Registering a DSI service listening for request-response**

The registration of a DSI service listening for the request-response mechanism via the OSGi framework, an explicit registration with a DSI service provider shall not be necessary. If a response (or indication) to be shipped, determined the DSI service provider to be notified DSI service listener with the available mechanisms of OSGi (which mechanisms are available, refer to Chapter 3.5.4). The conditions for registration on OSGi, refer to Chapter 3.5.3 (Table 9).

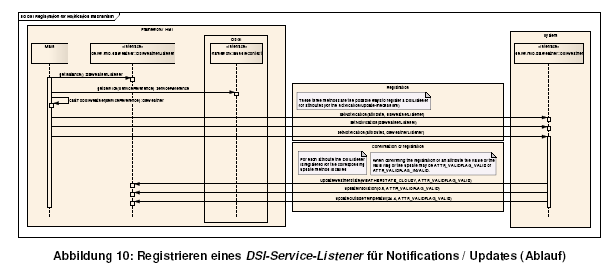


**4.2.2 Registering a DSI service listener for notifications / Updates**

For the registration of a DSI service listener for notifications / Updates defined in applicable to all DSI service provider base interface methods setNotification be (...) (see also Section 3.3.2.2) is used.

To register for notifications / updates the corresponding instance of the DSI service provider's method is first determined from the OSGi and on this the appropriate setNotification (...) is called (depending on whether the DSI service listener for an attribute more to be registered or all attributes).

In response to the registration to the DSI service listener or the corresponding Notification. Called Update method for each attribute that has been registered on the DSI service listener at the DSI service provider. This serves to confirm your registration. It is possible that there are no valid data for an attribute, invalid data is then passed to the Notification, this is indicated by the valid flag with the value ATTRVALIDFLAG\_INVALID (see also Section 3.2.3).



**4.3 flow of a request / response**

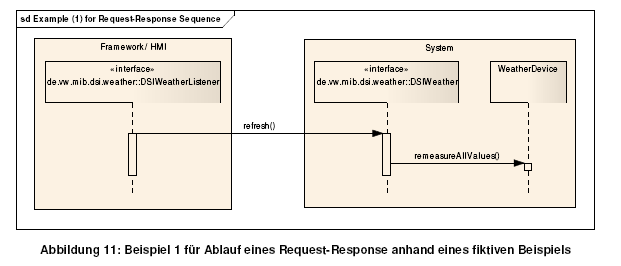
When a request is triggered on a DSI service provider, there are various ways in which the following process looks like. Basically, the process can operate at any given from the DSI concept options. So how exactly the processes after triggering a request look is defined individually for each request. The options presented here represent the most common variants, an exhaustive does not exist. The process for each request is specified by the respective DSI Services.

**4.3.1 Simple request (without response)**

The basic request without response corresponds to a normal function call. The only possible reaction of the DSI service provider to the appeal of such request relates to error handling, it can both exceptions are thrown and a asyncException () are triggered.

Typical application:

Initiating a refresh for all values, the reaction is then implicitly by Notifications with the current values.



**Notes to the example:**

The example in Figure 11 shows the request from the HMI to update all the weather data offered by the DSIWeather (re-read values from the sensors). For the request does not own response or other response is defined.

As an indirect reaction updating all weather data on the Notfication mechanism occurs if the values have changed.

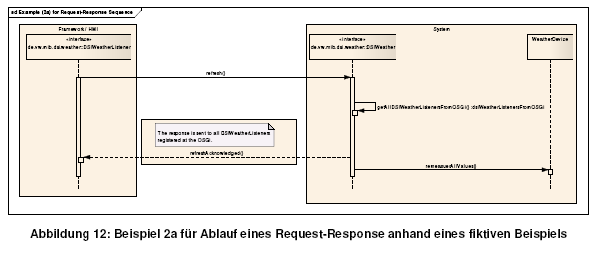
**4.3.2 Simple Request-Response**

In a simple request-response from the DSI service listener a request from the DSI service provider is called. In response informs the DSI all registered service providers for the response mechanism DSI service listener with the corresponding response.

Typical application:

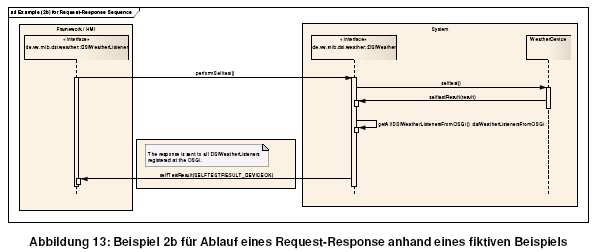
• Implementation of a mechanism acknowledge that the object of the request was accepted.

• Return the value determined by the request



Explanations for the Example 2a:

Figure 12 shows the call of a request with confirmation. The DSI service listener calls the request to update all offered by the DSI Service DSIWeather weather data (refresh ()).In response, the DSI service provider calls the corresponding response function on all registered on OSGi DSIWeatherListener and then starts processing the request. More reactions by the request explicitly (already implicitly by the context, as for all the weather data in a change notification is called) is defined.



Explanations for the Example 2b:

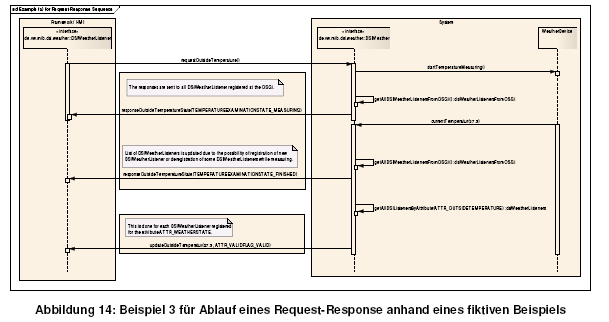
In Figure 13, a request can be seen with a response to return the result of the request. The DSI service listener requests initiated by a self-test of the DSI service provider DSIWeather. This implement them, and communicate the result to all registered in the OSGi DSIWeatherListenern with.

**4.3.3 request with status response and value-Notification**

When a request with a long or indefinite term of the DSI service provider informs the DSI registered service listener with status messages about the progress of the execution of the request in accordance with a defined response. Finally, the result of the request is indicated by a Notification. It is also possible that the notification of the result on the notification does not apply if the Notification is defined asnotify\_onChange and the data has not changed since the last update. If the Notification defined asnotify\_always, the update is performed in each case.

Typical application:

Tuning the radio.



Notes to the example:

The example in Figure 14 shows the flow of a request with an indefinite duration. From the DSI service listener, first the request is triggered, the current outdoor temperature is to be measured again. The DSI service provider DSIWeather starts out the measurement and indicates this with the corresponding response (all currently on OSGi registered DSIWeatherListener) and a parameter that tells the DSIWeatherListener that the measurement currently running (status message "Measurement in progress").

If the DSI service provider DSIWeather the system receives the value of the measurement, it signals this by calling the same response and the corresponding parameters (status message "Measurement completed") (all currently on OSGi registered DSIWeatherListener, the list needs to be recalculated because in the meantime may have unregistered or newly registered DSIWeatherListener!).

He then updates the measured value newly determined with the help of the associated Notification (registered at all for the attribute DSIWeatherListenern).

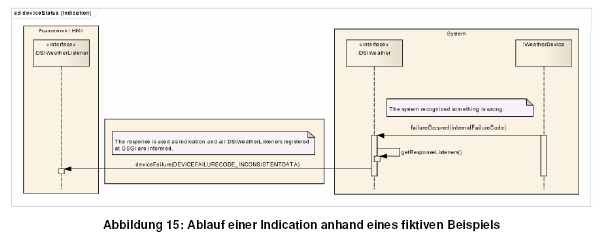
**4.4 Indication expiry of a**

An indication is itself a response that is called without a prior call the associated request function by DSI-service provider for all registered on the OSGi Service DSI DSI listener of the corresponding services. For the definition of an indication see chapter 3.2.2.

Typical application:

A typical application is the notification of the failure of a component, so that the HMI knows that currently no requests can be triggered it.

Another application is the notification of the construction and dismantling of a physical audio connection in the audio management.



Notes to the example:

The example shows how an error is detected in the system and the HMI communicates the error by indication.

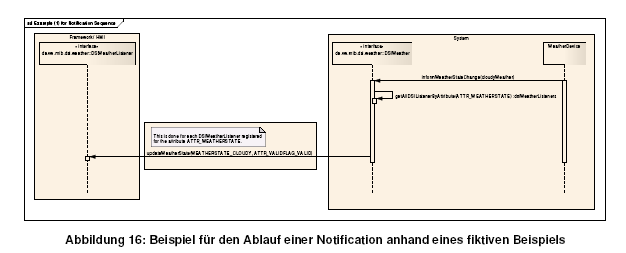
**4.5 expiration of a Notification**

To perform a notification for an attribute must perform the DSI Sevice Provider following steps:

1. Identify all registered for the attribute DSI service listener from its internal list maintained. These are all DSI Service listeners that at the DSI service provider with the setNotification (...) method and have the attribute belonging to the attribute constant registered (and have not deregistered again).
2. Call the appropriate update function on every registered for the attribute DSI Serivce listener (with the information for the attribute and the corresponding value for the valid flag).

Typical application:

A relevant for the HMI value has changed (eg the current played track on the CD).



Notes to the example:

Figure 16 shows the sequence of a notification based on a fictitious example. In the example shown, the HMI is informed of a change in the weather.

**5 History of changes**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | date | arranger | changes |
| V01E | 22.04.2008 | Kevin Hallbauer | first draft |
| V02E | 17.06.2008 | Kevin Hallbauer | Comments entered by Audi (results of the DSI Core Team RegelTelko) |
| V03E | 18.06.2008 | Kevin Hallbauer | • Add the chapter "Coding Conventions" and Apply the contents from the Audi document "DSI\_CodeConventions\_V1.0\_20080606" (Doris link:  https: //dorisappl.wob.vw.vwg/doris/documents/376  2684? Version = 1 & useInlineContent = false  • Add the chapter "DSI Service Design Patterns" |
| V04E | 12.08.2008 | Kevin Hallbauer | Unnecessary / worked off in the comments and painted texts removed |
| V05E | 28.08.2008 | Kevin Hallbauer | Unnecessary / worked off in the comments and painted texts removed |
| V06E | 09.04.2009 | Kevin Hallbauer | Completed Final Draft |
| V07E | 27.07.2009 | Kevin Hallbauer | Revision based on review results from 22.07.2009 and 23.07.2009 |
| V08E | 31.07.2009 | Kevin Hallbauer | Revision based on the decisions of the DSI core team rule Telco from 30.07.2009 |
| V09E | 25.11.2009 | Kevin Hallbauer | Revision based on reviews and Populate adopted changes to the DSI concept |
| V10E | 13.01.2010 | Kevin Hallbauer | Changes to the DSI Exception Handling entered accordingly workshop results |
| V11E | 17.02.2010 | Kevin Hallbauer | Revision of the chapter on error handling Adding a reglärern expressions |
| V12E | 11.03.2010 | Kevin Hallbauer | Revision of the chapter on the request procedures |
| V13E | 16.03.2010 | Kevin Hallbauer | Populate the review results of the VW chapter with the request procedures  Revision of the chapter on error handling  Determination of the only two special DSI Packages  org.dsi.ifc.base  org.dsi.ifc.global |
| V14E | 11.04.2011 | Kevin Hallbauer | Introduction of the chapter for the specification of default values for DSI data types  Sales data types added to a partially filled array in the chapter array.  Passages with DSIBoot / JDSIAdmin revised (new: org.dsi.ifc.base.ServiceAdmin)  Spelling / grammar corrections |

**Appendix A References and terms**

**A.1 References**

|  |  |
| --- | --- |
| reference | document |
| [1] | DSI Documentation Guideline  MIB\_HU\_DSI\_Documentation\_Guideline\_20090810\_V6E.doc |
| [2] | Sun Java naming conventions  http://java.sun.com/docs/codeconv/html/CodeConventions.doc8.html |
| [3] | Java Language Specification  http://java.sun.com/docs/books/jls/second\_edition/html/j.title.doc.html |
| [4] | Regular expression syntax (Specify in java.util.regex.Pattern, Java SE 1.4.2):  http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/Pattern.html |
| [5] | OSGi specifications  http://www.osgi.org/Specifications/HomePage |
| [6] | Specification of the DSI Service Admin  (Javadoc / Java source code of the corresponding DSI version) |
| [7] | Specification of the DSI Service Startup  (Javadoc / Java source code of the corresponding DSI version) |
| [8] | Description of the startup of the HMI  HMI\_Codegen\_StartUpSupplier.doc  https: //dorisappl.wob.vw.vwg/doris/documents/5896438 version = \* & useInlineContent = false? |
| [9] |  |

**A.2 Terminology used**

|  |  |
| --- | --- |
| term | explanation |
| activators | Defined way to enable the system-side implementation of the DSI Interface (OSGi) activator |
| application | An application is a specified by the system functionality. |
| attribute | An attribute denotes an information for which a DSI service listener can register for a DSI service provider and is transmitted by means of the ⇒Notification mechanism. |
| Attribute constant | Constant for the identification of an attribute or the associated update function |
| basic functions | Functions that are defined independently of context and superior |
| data object | Container to encapsulation of related data |
| DSI-Interface-Pair | Pair of each one belonging together⇒DSI-Service-Provider und⇒DSI-Service-Listener |
| DSI-Listener | Synonymous with the term ⇒DSI Service Listener |
| DSI-Provider | Synonymous with the term ⇒DSI service provider |
| DSI-Service | Within a context defined objects. Includes the definition of the DSI service provider, DSI service listener as well as the data objects |
| DSI-Service-Listener | The DSI DSI Service listener functions on HMI page specified |
| DSI-Service-Provider | In the DSI service provider the DSI functions are specified at the application side |
| function | Summary of a self-contained part of the program which is executed by calling the function. |
| HMI | Abbreviation for Human Machine Interface referred to herein collectively also above the DSI interface software running |
| HMI page | Is the side of the DSI interface which is on the finding that ⇒HMI. |
| Indication | Broadcast function for notifying the DSI Service Listener |
| context | Defined completed portion of functions for which a DSI service is defined (eg AMFMTuner, Bluetooth) |
| Context functions | Functions that are dependent on context defined |
| Runtime system | The runtime system referred to in this document delivered to the customer hardware and software. |
| method | Synomyn for the term ⇒Function  (The term comes from the object-oriented programming) |
| north side | Synonymous with the term ⇒HMI page |
| Notification | A notification is the notification of a DSI service listener through the DSI service provider over a change in the state of the system |
| Request | A request is a function call in a defined DSI service provider function by DSI Service Listener |
| Request constant | Constant to identify a particular request function |
| Response | A response is the reaction of a DSI service provider to a request. The DSI service provider calls to the appropriate defined in the DSI service listener function |
| Response constant | Constant for the identification of a particular response function |
| south side | Synonymous with the term ⇒ System-side |
| System-DSI | Synonyms for the implementation of the DSI interface on the system side |
| System-side | If the side of the DSI interface on which to find the system (hardware, applications) are |
| Update | Synonymous with the term ⇒Notification |
| Update function | An update function is defined in the DSI service listener function for the ⇒Notification mechanism |
| Update constant | Synonymous with the term ⇒Attribut constant |
| Update method | Synonymous with the term ⇒Update function |

|  |  |
| --- | --- |
| abbreviation | Definition |
| DSI | Device Service Interface |
| HMI | Human Machine Interface |
| LSD | Lightweight Service Directory |
| OSGI | Open Service Gateway iniative |
|  |  |

Appendix B List of Figures and Tables

Figure 1: Overview of the DSI concept 5

Figure 2: provided by DSI concept hierarchy levels 7

Figure 3: Relations between the elements of the DSI DSI model 8

Figure 4: Sign of the DSIBoot service provider on OSGi 54

Figure 5: Registration of a DSI service provider on OSGi 55

Figure 6: initiating a request for not yet initialized DSI service provider 56

Figure 7: Register a DSI service listener to uninitialized DSI service provider 56

Figure 8: Start a Domain 57

Figure 9: Register an instance of a DSI service listening for request-response (Proceedings) 58

Figure 10: Register a DSI service listener for notifications / updates (Proceedings) 59

Figure 11: Example 1 for flow of a request-response based on a fictitious example 60

Figure 12: Example 2a for the end of a request-response based on a fictitious example 61

Figure 13: Example 2b for the end of a request-response based on a fictitious example 62

Figure 14: Example 3 for flow of a request-response based on a fictitious example 63

Figure 15: Sequence of an indication based on a fictitious example 64

Figure 16: Example of the sequence of a notification based on a fictitious example 65

Table 1: Illustration of the basic DSI elements to elements of the Java programming language 9

Table 2: Call modalities for request functions 13

Table 3: Possible values for the valid flag 16

Table 4: Constants for the identification of methods 17

Table 5: Overview of the basis functions of the DSI service provider 19

Table 6: Valid values for the components of the version of the specification of a DSI Service 20

Table 7: Overview of the basic functions of the DSI service listener 23

Table 8: Parameters for the registration of a DSI service provider on OSGi 25

Table 9: Parameters for the registration of a DSI service listener on OSGi 26

Table 10: JavaDoc tags to identify the features supported by the OSGi LSD 27

Table 11: Overview of the basic data types 28

Table 12: Examples for the designation of classes for different DSI Services 36

Table 13: Regular Expressions for the above naming conventions 37

Table 14: Files and its content, including a supply of a DSI specification 40

Table 15: From the DSI concept defined Exceptions 49

Table 16: Parameters for the registration of the Service Admin on OSGi 53