*s*View – Architecture Overview and System Description

Markus Bylund

Swedish Institute of Computer Science, Box 1263, SE-164 29 Kista, SWEDEN
bylund@sics.se

Abstract. This report presents an architecture overview and a system description of the *s*View system. The system provides developers, service providers, and users of electronic services with an open and extendible service infrastructure that allows far-reaching user control. This is accomplished by collecting the services of an individual user in a virtual briefcase. Services come in the form of self-contained service components (i.e. including both logic and data), and the briefcase is mobile to allow it to follow as the user moves between different hosts and terminals. A specification of how to build such service components and the infrastructure for handling briefcases is presented. A reference implementation of the specification as well as extensions in the form of service components is also described. The purpose of the report is to serve as a technical reference for developers of sView services and software infrastructure that builds on sView technology.

Keywords. Electronic services, personal service environments, user interfaces, mobility, personalization, service collaboration, component-based software engineering.

May 2001
SICS Technical Report T2001/06
ISSN 1100-3154
ISRN: SICS-T--2001/06-SE

1. Introduction

The use of electronic services is spreading more and more to an increasingly broader group of users, and there is a growing need for support for continuous interaction with multiple services, via different types of devices, and from all sorts of places and locations. Further more, it is desirable that this is done in a way that assures the user control over personal information that services gather and maintain. The user should also be able to control what services do and whether or not, and how, they collaborate with each other.

All these demands represent current research topics such as privacy in the context of electronic service usage, service collaboration, and ubiquitous user interface design. The *s*View system has been designed as a solution to some of these research topics, and to cater for further research on others. The system assumes a client server model. But instead of having a uniform client without service specific functionality for access to all servers (as in the case with the world wide web), access to the servers is channeled through a virtual service briefcase. The briefcase in turn, supports access from many different types of devices and user interfaces. It is also private to an individual user, and it can store service components containing both service logic and data from service providers. This allows the service provider to split the services in two parts. One part with commonly used functionality and user specific data that executes and is stored within the virtual briefcase. The other part provides network-based functionality and data that is common between all users. Finally, the service briefcase is mobile and it can follow its user from host to host. This allows local and partially network independent access to the service components in the briefcase.

At a high level, the *s*View system consists of two parts. The *core sView specification* provides APIs (Application Programming Interfaces) to developers of service components and service infrastructure that builds on *s*View technology. Implementing these APIs and adhering to the design guidelines that accompany the APIs, assures compatibility between *s*View services and service infrastructure of different origin. The *sView reference implementation* provides developers with a development and runtime environment for service components as well as a sample implementation of an *s*View server

The report is structured as follows. Section 2 describes a number of basic concepts and entities. Section 3, specifies the main requirements that has influenced the design of the *s*View system. Section 4 provides a detailed description of the core *s*View specification. Section 5 describes the *s*View reference implementation, and Section 6 concludes with a summary.

2. Basic Concepts

The *s*View system builds on the concept of *personal service environments* [1]. A personal service environment is an individually collected and tailored set of services, available to the user at all times. The services are retrieved from service providers around the Internet, but after retrieval they are at least partially independent of Internet access. The personal service environment itself is mobile, following its user around in the network. The interaction state of the services is saved as the personal service environment moves between hosts on the Internet. This allows for continuous interaction sessions as the user of the services switches between different interaction devices. In the reminder of this text, the personal service environment is referred to as the service environment or simply the environment.

The *s*View system defines three main entities: *service components*, *service briefcases*, and *service briefcase servers*.

- A service component is an entity that provides *services* to the user, and/or other service components within the same service environment. It is a collection of class definitions and resources that together define a component that can be loaded and executed in a personal service environment. This allows service components to collaborate about e.g. content provision, personalization, and user interface handling.
- A service briefcase is a data structure in which a personal service environment is stored. A service briefcase contains service component definitions, saved execution states of service components, and settings. It also includes functionality for loading, saving, and creating new service components.
- A service briefcase server constitutes an API that offers service briefcase handling such as create new service briefcase, start service briefcase (i.e. create a personal service environment based on a service briefcase), and synchronization between instances of a service briefcase on different service briefcase servers.

An illustration of the different parts of the *s*View system and their relations is given in **Fig. 1**. On the computer marked I a briefcase server and a service environment is executing. In this case the user is sitting next to the same computer as the service environment (represented by the cloud together with service components A, B, and C) is executing on. This makes it possible to use a standard GUI for user-service interaction. The computer marked II hosts service briefcases and environments for several users, which use remote interfaces. One user is using a web-kiosk with a web browser for user-service interaction (III) and another user uses a WAP phone (IV). Stored service environments, in the form of service briefcases (illustrated between I and II), can migrate between any computers that run a briefcase server.

Finally, in the reminder of this document I will refer to an *sView server* as a combination of a service briefcase server and the server software with which personal service environments execute.

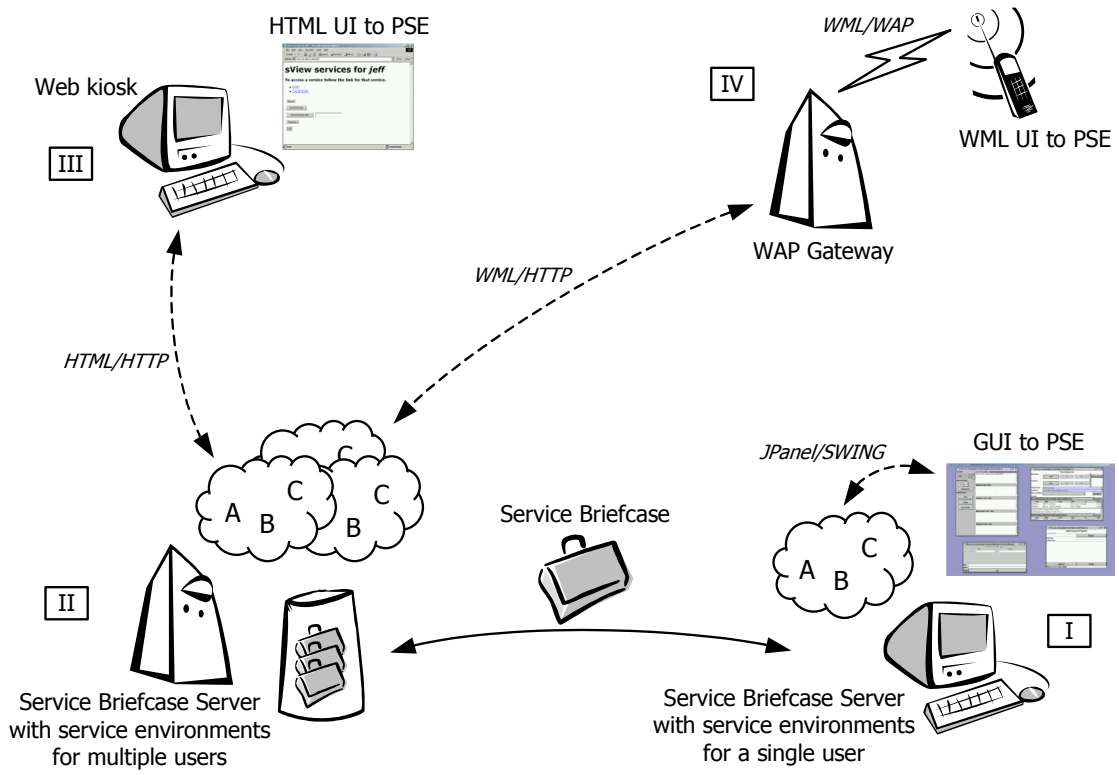


Fig. 1. The main entities of the *s*View system and their relations.

3. Design Requirements

The high-level design goals with the *s*View system have been specified as *openness* and *user control* [1]. Openness implies that it should be possible to add service components and users to the system without affecting other parts of the system. User control implies that the system should give the user control over which services to use, what information about the user that services handle, how services collaborate, etc. Some users may chose not to use make use of this control, but the possibility for user control should always exist. Furthermore, the user should be in control of the usage situation. In practice this means that services should be reachable from everywhere using many different types of devices, both the user's own devices and publicly available devices.

Openness and user control can be further analyzed in the terms of five more specific requirements: *heterogeneity*, *extendibility*, *accessibility*, *adaptability*, and *continuity*. The three former requirements are closely related to the personal service environment concept, and are discussed in more detail in [1]. The two latter requirements however, heterogeneity and extendibility, have had a profound impact on the design and implementation of *s*View and are discussed further below.

3.1. Heterogeneity

Many electronic services already exist, both in the form of commercial and research products. A sound requirement on an open infrastructure for user-service interaction is to allow a heterogeneous mix of service components to utilize features of each other. We approach the requirement on heterogeneity in a number of ways.

The *s*View system is implemented in Java. This brings at least two advantages: a reasonable chance of creating a platform independent system and easy integration of other Java based electronic service infrastructures [2-5]. The *s*View system puts few constraints on the implementation of the service components, which makes the integration of other service infrastructures straightforward.

Developers of an *s*View service component can chose between implementing all of the functionality in the service component, or placing all functionality on a server in the network (in which case the *s*View service component only serves as a proxy to the network based functionality). Any combination of the two alternatives is also possible. This allows for integration of already existing network-based services into the *s*View system.

*s*View service components are free to communicate with external resources (such as network-based services) using any protocol of their like. This communication is not in any way limited by the *s*View system.

3.2. Extendibility

Openness also implies a demand for extendibility. As new services are added to the system it should be possible to add support for new protocols. This would make it possible to add functionality for user-service interaction, communication between service components, collaboration between services of different kinds, enhanced security handling, etc. With the current design of *s*View, we approach the requirement on extendibility in three different ways.

Firstly, the functionality of an *s*View service component need not be targeted towards the user of the system, but can instead provide functionality to other service components in the user's service environment. This makes it easy to extend the *s*View system with new functionality. For this purpose, it is useful that *s*View service components can build on, and include in its distribution, existing Java packages.

Secondly, the *s*View system is separated in two parts: a core specification and a reference implementation. The core specification includes the APIs that are necessary in order to implement *s*View servers that are compatible with each other. The API also ensures that all *s*View service components are executable in any implementation of an *s*View server.

Thirdly, the core specification includes a method for *s*View servers to dynamically load new implementations of server-server communication protocols. *s*View servers can therefore communicate in any protocol that can be implemented in Java.

Class/*Interface*	Service Component	Server Functionality
Constants	Can implement/Must use	Must use
Mobile	Can implement	Must use
Monitor		Must use
Persistent	Can implement	Must use
ServerProxy		Can implement/Must use
ServiceBriefcase		Must use
ServiceBriefcaseServer		Must implement
ServiceComponent	Must implement	Must use
ServiceComponentPermission		Must use
ServiceContainer		Must use
ServiceContext		Must implement
ServiceInterfaceFactory	Can implement	Must use
ServiceListener		Must implement
ServiceProxy		Can implement/Must use
TransactionCoordinator		Can implement/Must use
TransactionInitiator		Can implement/Must use
TransactionParticipant		Can implement/Must use

Table 1. The main classes and interfaces in the core specification (`se.sics.sview.core`).

4. The Core *s*View Specification

The core *s*View specification consists of about 60 Java classes and interfaces that are needed in order to implement service components and *s*View servers. The total size of the specification is less than 40 KB. **Table 1** lists the most important classes and interfaces and relates them to either service components or server functionality. See Appendix I for a full listing of all classes and interfaces.

4.1. API Overview

The basic architecture of the core *s*View specification can be described with four entities (see **Fig. 2**): service component, service briefcase, service briefcase server, and service context.

The three former entities were briefly described in Section 2. The latter entity, the service context, constitutes the context in which a service component executes. The service context offers a service component an API that allows the component to e.g. register services, subscribe to other services, and manage other service components.

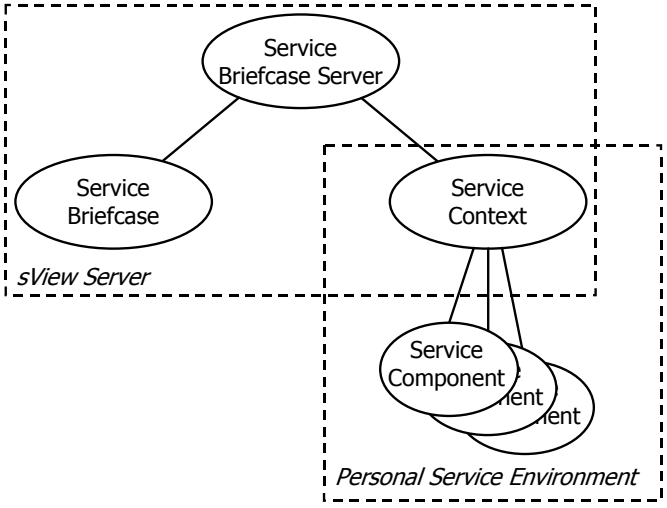


Fig. 2. Overview of the basic architecture of the core *s*View specification.

4.2. Service Component

An *s*View service component is created by implementing the Java interface `se.sics.sview.core.ServiceComponent`. The class definitions of the service component needs to be packaged in a JAR file together with a manifest with information about the component. During runtime, the service component follows a lifecycle that is defined by a set of states and a state transition graph. Finally, the service component can be extended to allow persistence and mobility.

Packaging and Distribution of a Service Component. A service component is packaged and distributed as a JAR file [6]. All class definitions and resources of the service component should be included in this file, as well as information about e.g. the name and structure of the service component.

The most important part of the JAR file is the set of class files that define the functionality of the service component. Class files can be included in the JAR file in two ways: either directly as main JAR entries (which is the usual way), or in nested JAR files. The latter way is convenient if the service component depends upon external packages in JAR files. If classes are included in this way an internal class path must be given in the JAR manifest (see below).

If the service component registers services for other service components to use, it should export class definitions from its own JAR file to these other service components. Every class or interface definition that is needed in order to use the service should be exported. In the simplest case, only a single interface is needed, but for more advanced services whole packages might have to be exported. Class definition exports are specified in the JAR manifest (see below).

Resources in the form of images, databases or just about anything that can be stored in a file can be included in the JAR file. Upon request, resources are made available to the service component (via the service context) in the form of byte arrays.

```
Manifest-Version: 1.0
ServiceComponentName: Sample 1 Service Component
ServiceComponentActivator: Sample1
ServiceComponentClasspath: ., javamail.jar, servlet.jar
ServiceComponentExport: Sample1ServiceInterface
ServiceComponentPermission: ServiceComponent, ServiceEnvironment
```

Listing 1. An example of a JAR manifest of a service component. When listing permissions from the package `se.sics.sview.core.permission`, the package name can be omitted.

The JAR file of a service component must contain a manifest with information about the service component. Following is a list of entries that can (must) be specified in the manifest.

- `ServiceComponentName` (*mandatory*) – a symbolic name of the service component.
- `ServiceComponentActivator` (*mandatory*) – the fully qualified class name of the class of the service component that implements the interface `se.sics.sview.core.ServiceComponent`.
- `ServiceComponentClasspath` – the internal class path of the JAR file. Should be a comma separated list of JAR entries (being themselves JAR files) or '.' (which stands for the classes in the root JAR file). List entries are searched for class definitions in order of appearance.
- `ServiceComponentExport` – a comma separated list of package names or fully qualified class names that should be exported to other service components.
- `ServiceComponentDepend` – a list of names of services (offered by other service components) that this service component depends upon.
- `ServiceComponentPermission` – a list of permissions that grants the service component rights to functionality in the system (see Section 4.3).

An example of a JAR manifest for a service component is given in **Listing 1**. The JAR file includes two nested JAR files (javamail.jar and servlet.jar) that are both included in the service component classpath. The service component also exports a class definition: the class `Sample1ServiceInterface`. Finally, the service component is given two permissions: `ServiceComponent` and `ServiceEnvironment`.

Service Component Lifecycle. The lifecycle of a service component is described by a set of states and a state transition graph (see **Fig. 3**). Half of the transitions are initiated by the service context and the other half by the service component. Service context initiated state changes always occur as a result of the service context calling one of the methods of the service component (initialize, start, suspend, resume, or stop). Service component initiated transitions can occur in one of two ways. The service component either initiates the state change by returning the value of the new state from the methods that the service context calls, or if the state change should be delayed after returning from the method, by explicitly setting the state by calling the `setState` method of the service context.

Following is a description of the different states of the service component.

- `INACTIVE` – The service component is either newly created and not yet added, or recently removed from, a service environment. In this state the service component

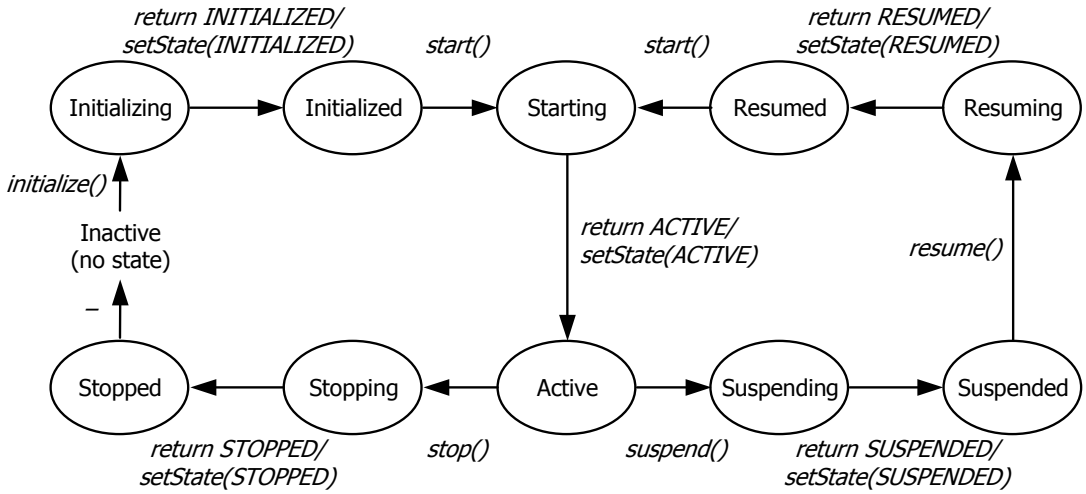


Fig. 3. The state graph describing the lifecycle of a service component.

is not allowed to interact with either its service context or with other service components.

- `INITIALIZING` – The service component automatically reaches this state when the service context calls the `initialize` method of the service component. This is done as a first step to add the component to the service environment. In this state, the service component is expected to perform initialization that is only done once during the lifetime of a service component. This is the first chance of the service component to interact with the service context, but interaction with other service components is not allowed yet. The service component signals that initialization is done either by having the `initialize` method return `INITIALIZED`, or, if initialization continues after returning from the `initialize` method, by calling `setState(INITIALIZED)` on the service context. In the latter case a negative number should be returned by the `initialize` method to signal that initialization is not finished.
- `INITIALIZED` – The service component reaches this state when it has finished initialization.
- `STARTING` – The service component automatically reaches this state when the service context calls the `start` method of the service component. In this state, the service component should perform tasks that should be done every time it is about to start. Interaction with the service context is allowed, but not with other service components. The service component signals that starting is done either by having the `start` method return `ACTIVE`, or, if starting continues after returning from the `start` method, by calling `setState(ACTIVE)` on the service context. In the latter case a negative number should be returned by the `start` method to signal that starting is not finished.
- `ACTIVE` – The service component reaches this state when it has finished starting. This is the state where most of the lifecycle of a service component is spent. The service component is allowed to interact with both the service context and other service components from here.

- `SUSPENDING` – The service component automatically reaches this state when the service context calls the `suspend` method of the service component. This is done as a first step to suspend the component. In this state, the service component is expected to unregister all services that it offers other service components, as well as unsubscribe to services of other service components. The service component is allowed to interact with the service context in this state. It is also allowed to interact with other service components, but only for the purpose of handling unsubscriptions and unregistrations. The service component signals that suspension is done either by having the `suspend` method return `SUSPENDED`, or, if suspension continues after returning from the `suspend` method, by calling `setState(SUSPENDED)` on the service context. In the latter case a negative number should be returned by the `suspend` method to signal that suspension is not finished.
- `SUSPENDED` – The service component reaches this state when it has finished suspension. In this state the service component is not allowed to interact with either its service context or other service components. The service component can now be saved to persistent media or moved to another server.
- `RESUMING` – The service component automatically reaches this state when the service context calls the `resume` method of the service component. This is done as a first step to resume the component after suspension. This state is comparable to the `INITIALIZING` state, with the exception that the state can occur more than once. The service component signals that resumption is done either by having the `resume` method return `RESUMED`, or, if resumption continues after returning from the `resume` method, by calling `setState(RESUMED)` on the service context. In the latter case a negative number should be returned by the `resume` method to signal that resumption is not finished.
- `RESUMED` – The service component reaches this state when it has finished resumption.
- `STOPPING` – The service component automatically reaches this state when the service context calls the `stop` method of the service component. This is done as a first step to stop the component. In this state, the service component is expected to unregister all services that it offers other service components, as well as unsubscribe to services of other service components. The service component is allowed to interact with the service context in this state. It is also allowed to interact with other service components, but only for the purpose of handling unsubscriptions and unregistrations. The service component signals that stopping is done either by having the `stop` method return `STOPPED`, or, if stopping continues after returning from the `stop` method, by calling `setState(STOPPED)` on the service context. In the latter case a negative number should be returned by the `stop` method to signal that stopping is not finished.
- `STOPPED` – The service component reaches this state when it has finished stopping. In this state the service component has reached the end of its lifecycle. Only a reload of a previously saved copy or creating a new instance of the service component can bring the service component back to the service environment. In this state the service component is not allowed to interact with either its service context or other service components.

```
import se.sics.sview.core.*;
import se.sics.sview.core.event.*;

public class Sample1 implements Constants, ServiceComponent, Runnable {
  Thread ct;
  ServiceContext sc;
  ServiceContextEvent ce;

  // Implementations of interface ServiceComponent

  public int initialize(ServiceContext context, ServiceContextEvent evt) {
    // do initialize here - NOT computation intensive
    return INITIALIZED;
  }

  public int start(ServiceContext context, ServiceContextEvent evt) {
    // do start here
    sc = context;
    new Thread(this).start();
    return -1;
  }

  public int suspend(ServiceContext context, ServiceContextEvent evt) {
    if (ct==null) {
      // already suspended
      return SUSPENDED;
    } else {
      interrupt(context, evt);
      return -1;
    }
  }

  public int resume(ServiceContext context, ServiceContextEvent evt) {
    // do resume here - NOT computation intensive
    return RESUMED;
  }

  public int stop(ServiceContext context, ServiceContextEvent evt) {
    if (ct==null) {
      // already stopped
      return STOPPED;
    } else {
      interrupt(context, evt);
      return -1;
    }
  }

  // Implementations of interface Runnable

  public void run() {
    sc.setState(ACTIVE);
    ct = Thread.currentThread();

    try {
      // do run here - computation intensive
    } catch (InterruptedException e) {
      if (ce instanceof SuspendEvent) {
              // do suspend here - computation intensive
              sc.setState(SUSPENDED);
      } else if (ce instanceof StopEvent) {
              // do stop here  - computation intensive
              sc.setState(STOPPED);
      }
    }
    ct = null;
  }

  // Misc.

  public void interrupt(ServiceContext context, ServiceContextEvent evt) {
    ce = evt;
    sc = context;
    ct.interrupt();
  }
}
```

Listing 2. A sample implementation of a threaded service component with state handling.

```
import se.sics.sview.core.*;
import se.sics.sview.core.event.*;

public class Sample2 implements Constants, ServiceComponent, Persistent {
  Vector users = new Vector(42);              // a vector for user information
  Hashtable mediaCache = new Hashtable();     // a media cache for video clips

  <snip>

  // Implementations of interface Persistent

  public void freeze() {
    users.trimToSize();                       // compact the vector of users
    mediaCache = null;                        // remove the media cache
  }

  public void thaw() {
    mediaCache = new Hashtable();             // recreate the media cache
  }
}
```

Listing 3. An example of an implementation the interface `Persistent`.

Listing 2 gives an example of the state handling of a threaded service component. The purpose is to have code that requires lots of computation (in the example the calls to `start`, `suspend`, and `stop`) execute in a separate thread. Initialization and resumption, which are not computation intensive in the example, are run from the thread of the service context (i.e. within the call to `initialize` and `resume`). In the `start` method, the thread of the service component is started, but state `ACTIVE` is not entered until the service component executes in its own thread. Suspension and stopping is also handled within the thread of the service component, but only if `suspend` or `stop` are called while the thread of the service component is running. Otherwise it is handled in the same way as `initialize` and `resume`.

Persistence and Mobility. Service components can be made persistent and have its execution state (as a serialization of the objects that constitute the service component) saved in the service briefcase. They can also be made mobile which means that they will follow the service briefcase as it migrates between servers.

A service component is made persistent by implementing the interface `se.sics.sview.core.Persistent`. This requires the service component to implement two methods: `freeze` and `thaw` (see **Listing 3** for an example).

The service briefcase calls the `freeze` method when it saves the service component. This occurs after the service component has reached state `SUSPENDED`, but before state `RESUMING` is reached. The `freeze` method should be used to prepare for serialization by optimizing or removing data structures. The service component could e.g. compact a hash table or empty a media cache for more efficient storage After returning from the `freeze` method all external references (such as references to the service context, file and socket handles etc.) must have been set to `null`[1].

The service briefcase calls the `thaw` method when a saved version of the service component is loaded. This occurs after the `freeze` method has been called (possibly in a different VM and on a different host), but before state `RESUMING` is reached. The `thaw` method should be used to, if needed, recreate data structures that were removed

[1] Unless the reference is declared as `transient`.

Mobile / Persistent	Yes	No
Yes	Follows the user and preserves its state (e.g. a calendar).	Does not follow the user but preserves its state (e.g. a printer queue).
No	Follows the user but does not preserve its state (e.g. a proxy to a web based service).	Does not follow the user nor preserve its state (e.g. a driver for a loudspeaker).

Table 2. Examples of four types of service components.

or converted in the `freeze` method. It should also be used to re-associate references that were set to `null` in the `freeze` method or during serialization.

A service component is made mobile by implementing the marker interface[2] `se.sics.sview.core.Mobile`. This will allow the service briefcase to include the service component when migrating to other hosts.

The properties of service component persistence and mobility are orthogonal. A persistent service component that is not mobile can save its state locally, but not migrate to a different node. A mobile service component cannot save its state, neither locally nor while migrating; every time such a service component is restarted it will start from scratch (which is fine for many services). The most powerful service component however combines the two properties. Such a service component can both save its state and migrate. **Table 2** lists and exemplifies the four possible combinations of the two properties.

4.3. Service Context

The service context provides runtime handling of service components. It controls the lifecycle of service components by setting the states of the components. While doing so, the context informs the service component about the reason for the state change by sending an event. The service context gives service components access to three different kinds of properties (simple databases for storing settings). The context also provides an API for communication between service components, as well as handling of other service components and even the server itself. For the latter part, service components needs privileges that are granted to the component based on permissions.

Events. The service context controls the state of service components by calling the methods `initialize`, `start`, `suspend`, `resume`, and `stop` on the activator objects of the components. These methods take two arguments: the first is a reference to the service context itself. The second is an event, a subclass of `se.sics.sview.core.ServiceContextEvent`, with information about the reason

[2] A marker interface is an interface with an empty body. The purpose of such an interface is to signal that the implementation of the interface should be considered to have a certain property. In this case to be mobile.

behind the state change. The service component might want to use this information when deciding how to act upon the state change.

There are three main types of events.

- `StartEvent` – is used to take the service component from state `INACTIVE` and `RESUMED` through all the states to `ACTIVE`. Examples of this event include `CreateEvent` and `LoadEvent`.
- `SuspendEvent` – is used to take the service component from all states except `INACTIVE`, `STOPPING`, and `STOPPED` through the states to `SUSPENDED`. Examples of this event include `SaveEvent` and `SynchronizeEvent`.
- `StopEvent` – is used to take the service component from all states except `INACTIVE` and `STOPPED` to the state `STOPPED`. Examples of this event include `RemoveEvent` and `ReloadEvent`.

The documentation of for the events in package `se.sics.sview.core.event` contains a more detailed description of the information that individual events carry (see Appendix I).

Properties. The service context administers three sets of properties for storing settings of different kinds.

- *Local properties* deal with settings that are shared between all personal service environments on a particular server (such as references to means of interacting with the user from the server). Local properties cannot be set or changed by the service context or individual service components; instead, the administrator of the server controls these properties.
- *Stationary properties* are not shared between users, but they are still bound to a particular server. They can for example store settings such as the user's UI preferences, which is likely to differ between servers. Stationary properties can be created and modified by the service context and individual service components.
- *Mobile properties* are personal, just like stationary properties, but in contrast they do not vary with server. Mobile properties typically deal with settings that are not location dependent (e.g. user information such as name, address, etc.).

The three types of properties are convenient for pushing server settings to service components (local properties) and for saving and sharing information between service components (stationary and mobile properties).

Service Component Communication. Service components can communicate and collaborate by offering services to each other. The establishment of a service provider/consumer relationship is described in **Fig. 4**, in which service component A (SCA) offers service component B (SCB) a service.

I. The process is initiated by SCA by registering its service (S1) to the service context, during which SCA passes two parameters: a name of the service and a *service interface factory*. The latter should be an implementation of the interface `se.sics.sview.core.ServiceInterfaceProxy`, which is used by the service context to create interfaces to the service.

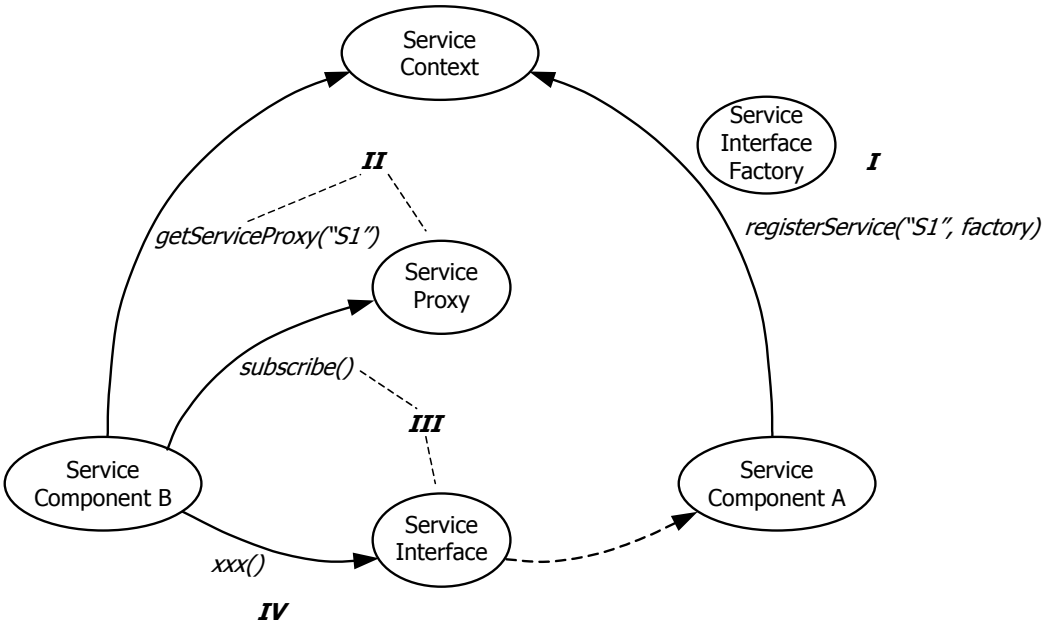


Fig. 4. A graph describing the establishment of a service provider–consumer relationship.

II. SCB requests a proxy to the service that SCA registered from the service context. The service proxy is of the same type regardless of which service is requested (`se.sics.sview.core.ServiceProxy`).
III. Using the service proxy, SCB starts a subscription to service S1. The service proxy uses the service interface factory that SCA provided to create an interface the service.
IV. SCB can now use service S1 by invoking methods on the service interface. Note that SCB needs to know the type of the service interface for this scheme to be effective.

The above description is only an example of how a relationship can be established; alternative ways are also possible. Phase II could for example happen before phase I (even without the existence of SCA). In such a case, SCB could attempt to start a subscription to a service that was not registered already, resulting in a `null`-reference instead of a reference to a service interface. The service proxy includes functionality for handling such situations. SCB could for example specify that if the requested service is not registered, the call should wait until it is. The service component could also register itself as a listener to (un)register notifications of the service, in which case SCB would know when to start the subscription.

The providing service component can unregister its services at any time. Consumers of those services are then obliged to unsubscribe and to stop using the services as soon as possible.

Server and Service Component Handling. The service context provides an API that allows service components to handle its server as well as other service components. The API lets service components reload, save, and synchronize the service environment, as well as shutting down the service environment (or, in the case of a single-user server, the server).

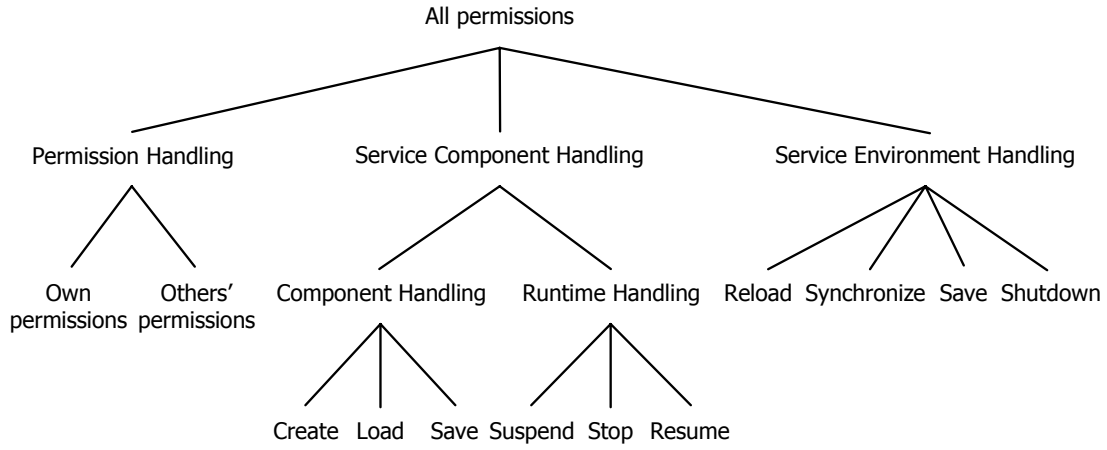


Fig. 5. The hierarchy of permissions for accessing service context functionality.

The API allows service components to create, load, and save other service components. It also allows service components to partially control the lifecycle of other components in that they can force other components to suspend, resume, and stop[3].

Finally, service components can also control their own lifecycle with this API by requesting the service context to start, suspend, resume, and stop the service component.

Permissions. The service context functionality for handling the service environment and other service components is protected by permissions. Permissions specifications are included in the core specification, and they are arranged in a hierarchy so as to allow both specific and general permissions (see **Fig. 5**). By extending an interface with several permissions at the same time, combinations of permissions can be implemented.

Permissions for a service component are specified by a comma-separated list of the class names that corresponds to the permissions should be included in the JAR manifest (see **Listing 1**).

4.4. Service Briefcase

The service briefcase (`se.sics.sview.core.ServiceBriefcase`) contains functionality for creating, loading and saving service components. It also provides storage of the JAR files of service components, persistent service components, and properties.

The service briefcase is serializable and it can be stored on persistent media and sent between servers, or have its content synchronized with service briefcases on other servers.

Much of the functionality of the service briefcase is delegated to *service containers* (`se.sics.sview.core.ServiceContainer`), of which there is one for each

[3] Initialize occurs implicitly as a result of adding a component to the service environment and start occurs automatically whenever a component has finished initializing or resuming.

service component in the briefcase. The service container provides storage and serialization handling of individual service components. It includes functionality for creating, loading, and saving service components, storing persistent service components, and caching the JAR file of service components.

Service component creation and loading requires that a class loader is provided by the server implementation. The server typically uses separate class loaders for every service component in the system. This ensures that no service component should be able to manipulate other service components without permission.

Service Briefcase State. An important step when synchronizing content between different service briefcases is to compare the *state of service briefcases*. The service briefcase state includes the names, keys, creation dates, change dates, and JAR status, of every mobile service component in the service briefcase.

User id and password. Most of the functionality of the service briefcase is protected with a user id and a password. Upon creation of the service briefcase, the user has to provide a user id and a password. The user id is used to uniquely identify the owner of the briefcase when moving briefcases between servers or synchronizing content between several instances of the same briefcase. However, *s*View does not provide a method of assigning unique identifiers to every user. Users are instead encouraged to use an already existing unique Internet identifier (such as an e-mail address) as their user id.

The password is encrypted using the MD5 Message-Digest Algorithm [7] and stored in the service briefcase as a 128 bit long 'fingerprint' of the password. In order to use the protected functionality, the user has to provide the password, which is encrypted and compared with the original password 'fingerprint'.

Note that the user id and password by no means represents a complete, or even partially satisfying, protection of the service briefcase. The scheme is merely used as an illustration of the need for protection. A true protection of the service briefcase must include at least two parts: authentication and encryption of the content. This should be implemented as a plugable solution, allowing the user to freely select which implementation, and therefore also which algorithm, for each of the two parts to use.

4.5. Service Briefcase Server

The service briefcase server provides an API for service briefcase handling such as creating new and removing existing briefcases, as well as starting and stopping the execution of personal service environments. The API also includes functionality for moving service briefcases between servers, and for synchronizing content between different instances of a briefcase on different servers. The API is specified as a Java interface (`se.sics.sview.core.ServiceBriefcaseServer`).

Server-Server Communication. Since this server is designed to communicate with servers on other hosts, a Java interface will not be sufficient for most purposes. What is missing is a protocol that is capable of wrapping the server interface (e.g. Java RMI

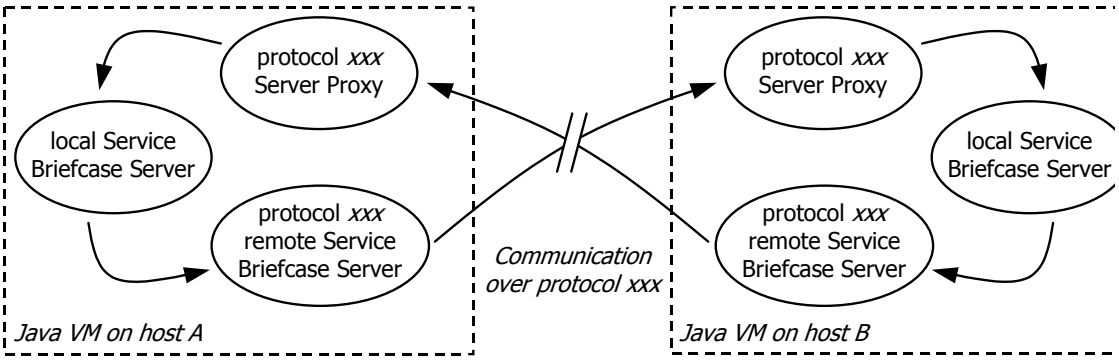


Fig. 6. A graph describing communication between two service briefcase servers over the fictive protocol *xxx*.

[8], SOAP[9], and HTTP [10]). However, every such protocol has its special characteristics with both strengths and weaknesses, and it would be impossible to select one or a few as the preferred protocols for *s*View. For this reason we have chosen not to specify any protocol at all in the core *s*View specification. Instead, an interface that provides access to implementations of service briefcase servers is specified (`se.sics.sview.core.ServerProxy`).

Note that this solution for server-server communication allows the implementation of different types of secure communication schemes. A server proxy could e.g. implement a protocol for secure authentication to avoid synchronizing service briefcases with fake servers. Different types of channel encryption and protocols to ensure information integrity could also easily be implemented.

Fig. 6 illustrates a communication path between two service briefcase servers. Without knowing anything about the communication protocol itself, local service briefcase servers can establish a communication link by creating instances of the 'protocol *xxx* Server Proxy' (which must implement the `se.sics.sview.core.ServerProxy` interface). Upon request, the server proxy creates the 'protocol *xxx* remote Service Briefcase Server'. In the above example, the server proxy acts as a server for incoming protocol *xxx* communication. It would also be possible let the remote service briefcase server take on this role, in which case the server proxy would only act as a factory for remote service briefcase servers.

Service Briefcase Synchronization. Service briefcase synchronization is a process that involves two or more service briefcase servers, and whose purpose is to synchronize the content of instances of a service briefcase on different servers. Note that this process concerns synchronization of the service briefcase instances of one user at a time. It can be described in a number of steps.

1. The initiating server (the initiator) requests the states of the service briefcase instances on the other servers (the participants).
2. The initiator requests the mobile properties of the service briefcase instances on the participants.
3. Based on its own and the participants states and mobile properties, the initiator generates a new state and a new set of mobile properties that represent the most up-to-date state and mobile property set of the service briefcase.

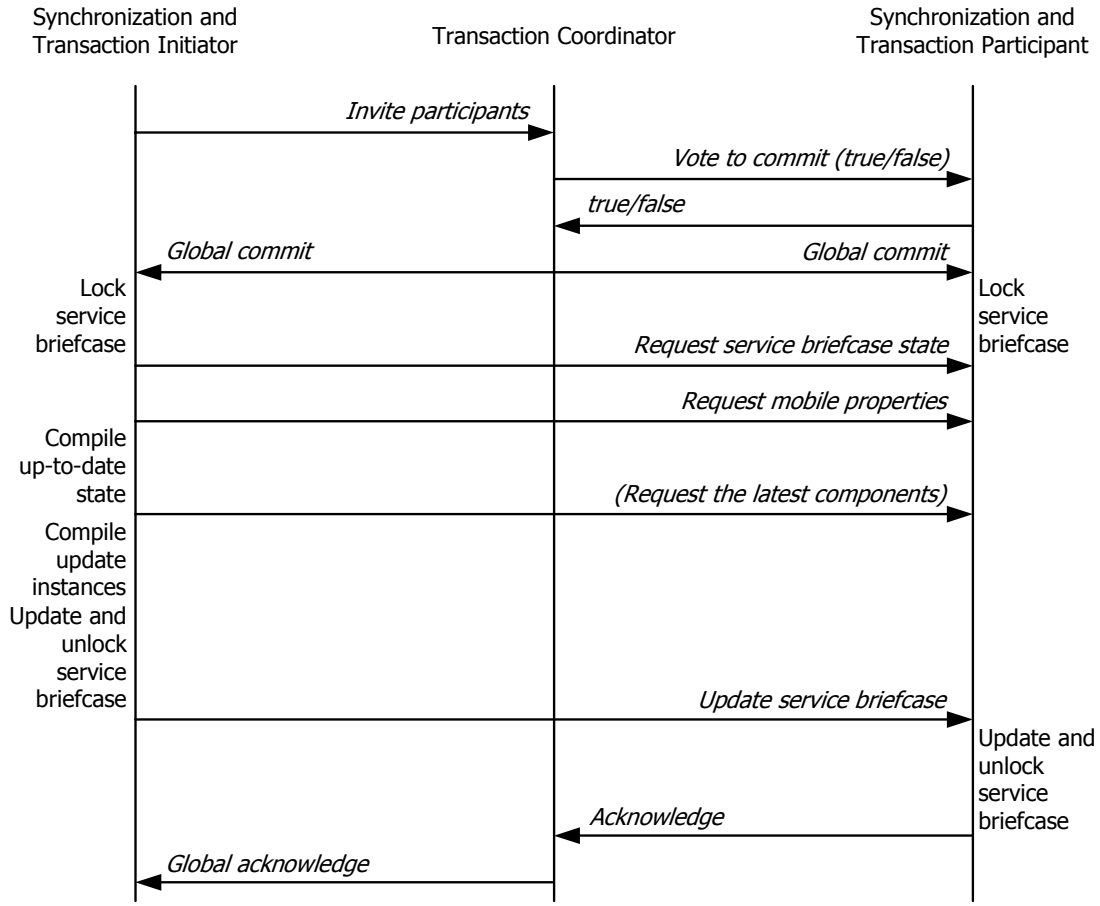


Fig. 7. A diagram describing the synchronization process with transaction handling.

4. The initiator generates a new instance of the service briefcase that reflects the latest state. This may involve requests of service components from one or more participants.
5. The initiator generates update instances of the new service briefcase. This is done exclusively for each participant, taking into account only the information that is needed to make that participant up-to-date.
6. The update instances are sent to the participants.

During this process, it is crucial that the service briefcase instances of the initiator and the participants are not modified, or else consistent behavior cannot be guaranteed. For this purpose, the service briefcase is equipped with a monitor (`se.sics.sview.core.Monitor`) that allows the service briefcase server to prevent modifications of the service briefcase. The monitor is designed to allow for concurrent modification of the service briefcase while unlocked.

It is also important to be able to handle both initiator and participant failure during the synchronization process. This is accomplished by wrapping the synchronization process in a modified version of the two-phase commit transaction protocol.

The whole process (i.e. both synchronization and transaction handling) is described in **Fig. 7**. At any time, the transaction coordinator may send an abort message to both initiator and participants. Participants that receive an abort message before getting the 'Update service briefcase' message will simply reset the transaction and unlock the

```
java.awt                          java.swing.event
java.awt.event                    java.swing.table
java.io                           java.text
java.net                          java.util
java.lang                         java.util.zip
java.swing                        java.util.jar
```

Listing 4. The packages that the reference implementation is built upon.

service briefcase. This will also happen if the response time from either the transaction coordinator or the initiator is too long. Abort messages that participants receive after the service briefcase update are discarded.

If the initiator receives an abort message before the 'Global acknowledge' message, the transaction is reset and the synchronization has to be restarted. However, if the initiator has updated its service briefcase before abort is received, the synchronization process is likely to require fewer steps than otherwise since the initiator has an up-to-date instance of the service briefcase. Note that it does not matter if an abort occurs when some of the participants have updated their briefcases and some have not. The briefcases that have not been updated will be so during a following retry.

5. The *s*View Reference Implementation

The reference implementation of the *s*View system was developed for two purposes. Firstly, it should serve as a development and runtime environment for developers of *s*View service components. Secondly, it should serve as a sample implementation of the core *s*View specification for developers of *s*View server functionality. It is freely available for download from the *s*View web site (http://sview.sics.se/) for everyone to use.

5.1. Current Implementation

The reference implementation is based on J2SE (Java 2 Platform, Standard Edition) version 1.3. The implementation is, apart from the core *s*View specification, only based on a number of packages from the J2SE runtime libraries (see **Listing 4**).

The current version of the reference implementation (version 2.0, alpha 1) supports most of the features of the core sView specification. However, it is not intended as an optimized, secure, and fully scalable runtime environment. The support for such features is therefore limited or non-existent. It is also limited to serving one personal service environment at a time and it does not support briefcase retrieval by date (see `se.sics.sview.core.ServiceBriefcaseServer`). The implementation consists of about 40 classes and its size is less than 125 KB.

5.2. Extensions

For server-server communication, the reference implementation includes an IP socket based implementation of the server proxy communication wrapper (described in Section 4.5). This allows different implementations *s*View servers to communicate over an IP socket based protocol.

In order to be open and customizable, the core *s*View specification does not include UI handling. This is instead left as a task for service components to handle. The reference implementation includes service components that handle user interfaces of three types: GUIs specified in Java Swing as well as HTML and WML user interfaces.

To complement the set of user interface managers, the reference implementation includes a set of service components for handling of other miscellaneous functionality. The *IntraCom (Intra Communication) manager* let service components register a mailbox to which other service components can post messages. This allows spontaneous communication between service components that are new to each other. The *Preference manager* offers rudimentary handling of preference entries (key and value pairs) of the user. Service components can store and fetch entries, as well as subscribe to changes in the preference database. The user can inspect the database, and control which services should be allowed access to which entries. The Preference manager stores its database of preference entries as mobile properties.

6. Conclusions

We have described the overall architecture and the basic design and implementation of the *s*View system. In general, the design is motivated by the two requirements openness and user control. In particular, demands on heterogeneity and extendibility have influenced the design.

In order to allow extensions to the system it is separated into two parts: one core specification that provides APIs to main components of the system, and one reference implementation that provides developers of *s*View components and server functionality with a development and runtime environment.

The core specification builds roughly on three main components: a service component, a service briefcase, and a service briefcase server. In combination these three components provides developers, service providers, and end-users of electronic services with an open and extensible service infrastructure that allows far-reaching user control.

7. Acknowledgements

The design and implementation described in this report builds upon the author's experiences from participating in the development of similar systems [11-14].

The work presented in this report has been funded by The Swedish Institute for Information Technology (www.siti.se). Thanks to the members of the HUMLE

laboratory at the Swedish Institute of Computer Science (www.sics.se/humle), in particular Fredrik Espinoza, for inspiration and thoughtful comments. Special thanks to Mikael Boman and Anna Sandin for help with the implementation of *s*View.

References

[1] M. Bylund and A. Waern, "Personal Service Environments – Openness and User Control in User-Service Interaction," Swedish Institute of Computer Science, Kista, Sweden, SICS Technical Report T2001:07, May, 2001.

[2] "OSGi Service Gateway Specification Release 1.0," Open Services Gateway Initiative, May, 2000.

[3] H. L. Chen, "Developing a Dynamic Distributed Intelligent Agent Framework Based on the Jini Architecture," M.Sc. thesis, University of Maryland Baltimore County, Baltimore, 2000.

[4] N. Minar, M. Gray, O. Roup, R. Krikorian, and P. Maes, "Hive: Distributed Agents for Networking Things," presented at First International Symposium on Agent Systems and Applications, Third International Symposium on Mobile Agents featuring the Third Dartmouth Workshop on Transportable Agents, Rancho Las Palmas Marriott's Resort and Spa, Palm Springs, CA, 1999.

[5] C. Pullela, L. Xu, D. Chakraborty, and A. Joshi, "A Component Based Architecture for Mobile Information Access," Department of Computing Science and Electrical Engineering, University of Maryland Baltimore County, Technical Report, TR-CS-00-05, March 31, 2000.

[6] "JAR File Specification," Sun Microsystems, Inc., available at: http://java.sun.com/j2se/1.3/docs/guide/jar/jar.html [2001, April 18], 1999.

[7] R. Rivest. "RFC1321: The MD5 Message Digest Algorithm," MIT and RSA Data Security, Inc., available at: http://www.faqs.org/rfcs/rfc1321.html [2001, April 17], 1992.

[8] "Java Remote Method Invokation Specification," Sun Microsystems, Inc., available at: http://java.sun.com/j2se/1.3/docs/guide/rmi/spec/rmiTOC.html [2001, April 18], 1999.

[9] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, "Simple Object Access Protocol (SOAP) 1.1," World Wide Web Consortium, W3C Note 27 July 1999, May 8, 2000.

[10] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. "RFC2616: Hypertext Transfer Protocol - HTTP/1.1," World Wide Web Consortium, available at: http://www.w3c.org/Protocols/rfc2616/rfc2616.txt [2001, April 17], 1999.

[11] P. Charlton, Y. Chen, F. Espinoza, A. Mamdani, O. Olsson, J. Pitt, F. Somers, and A. Waern, "An Open Agent Architecture Supporting Multimedia Services on Public Information Kiosks," presented at Practical Applications of Intelligent Agents and Multi-Agent Systems, PAAM'97, London, UK, 1997.

[12] F. Espinoza, "sicsDAIS: Managing User Interaction with Multiple Agents," Ph.Lic. thesis, The Royal Institute of Technology and Stockholm University, Stockholm, 1998.

[13] F. Espinoza, "sicsDAIS: A Multi-Agent Interaction System for the Internet," presented at WebNet 99—World Conference on the WWW and Internet, Hawaii, 1999.

[14] M. Tierney, “ConCall: An Exercise in Designing Open Service Architectures,” Ph.Lic. thesis, The Royal Institute of Technology and Stockholm University, Stockholm, Sweden, 2000.

Appendix I

This appendix contains the full Java documentation for the *s*View core specification, i.e. documentation of the following packages:

- `se.sics.sview.core`,
- `se.sics.sview.core.event`, and
- `se.sics.sview.core.permission`.

The documentation was automatically generated using the Javadoc tool[1] and the MIF doclet[2].

[1] http://java.sun.com/j2se/javadoc/index.html
[2] http://java.sun.com/j2se/javadoc/mifdoclet/index.html

Package
se.sics.sview.core

Class Summary	
Interfaces	
CallbackListener	A listener for callbacks from a Callback thread.
Constants	A set of constants used by service briefcases, service contexts, servers, etc.
Mobile	A service component that implements this interface will be included during service briefcase synchronization.
Persistent	A service component that implements this interface will be offered to save its state during service briefcase synchronization and save.
ServerProxy	This interface should be used to wrap implementations of remote service briefcase server communication and transaction initiator to transaction participant communication.
ServiceBriefcaseServer	This interfcase specifies an API to sView service briefcase servers.
ServiceComponent	Should be implemented by service components that wish to execute in an sView service briefcase.
ServiceComponentListener	A listener to service component state changes.
ServiceComponentPermission	Superclass of all service component permissions.
ServiceContext	This interface specifies an API to the runtime environment of a service briefcase.
ServiceContextListener	A listener to events from the service context.
ServiceInterfaceFactory	Service components that wish to register services for other service components to use must come with an implementation of this interface.
ServiceListener	The listener to service events.
ServiceProxy	A service component that wish to subscribe to a service requests a service proxy to the service from its service context.
TransactionCoordinator	A transaction wraps the steps in service briefcase synchronization in order to make it atomic, and to provide exception handling.
TransactionInitiator	A transaction wraps the steps in service briefcase synchronization in order to make it atomic, and to provide exception handling.
TransactionParticipant	A transaction wraps the steps in service briefcase synchronization in order to make it atomic, and to provide exception handling.
Classes	
Callback	A listener for callbacks from a Callback thread.
Monitor	Implements a 'one-to-many monitor' for exclusive access to critical sections.

Class Summary

ObjectInputStream-Loader	This subclass of ObjectInputStream delegates loading of classes to an existing Class-Loader.
ServiceBriefcase	Contains functionality for creating, loading and saving service components.
ServiceComponentEvent	The super class of all service component event.
ServiceContainer	A ServiceContainer wraps a service component, a JAR cache, and information about the service.
ServiceContextEvent	Superclass of all service context events.

Exceptions

PermissionDeniedEx-ception	Thrown if a service component attempts to call a method that it does not hold the permission to call.
ServiceBriefcaseServ-erException	This exception is thrown from the service briefcase server whenever a request experience fatal errors.
ServiceContextExcep-tion	This exception is thrown from the service context whenever a request experiences fatal errors.

se.sics.sview.core
Callback

Declaration
```
public class Callback implements java.lang.Runnable

java.lang.Object
  |
  +--se.sics.sview.core.Callback
```

All Implemented Interfaces: java.lang.Runnable

Description
A listener for callbacks from a `Callback` thread.

Member Summary	
Constructors	
`public`	`Callback(String, Object[], CallbackListener)` Creates a new Callback object with a callback of a given type, with an array of parameters, and with a reference to a `CallbackListener` object.
Methods	
`public void`	`run()`

Inherited Member Summary
Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

Callback(String, Object[], CallbackListener)

```
public Callback(java.lang.String type, java.lang.Object[] args,
                CallbackListener listener)
```

Creates a new Callback object with a callback of a given type, with an array of parameters, and with a reference to a `CallbackListener` object.

Parameters:
`type` - the callback type

run()

`args` - parameters to the callback

`listener` - the listener to call

Methods

run()

```
public void run()
```

Specified By: `run()` in interface `Callback`

se.sics.sview.core
CallbackListener

Declaration
```
public interface CallbackListener
```

Description
A listener for callbacks from a `Callback` thread.

Member Summary	
Methods	
`public void`	`callback(String, Object[])` Is called by the callback thread as soon as it starts executing.

Methods

callback(String, Object[])

```
public void callback(java.lang.String type, java.lang.Object[] args)
```

Is called by the callback thread as soon as it starts executing.

Parameters:

`type` - the callback type

`args` - parameters to the callback

se.sics.sview.core
Constants

Declaration
`public interface` **`Constants`**

All Known Subinterfaces: `ServiceContext`

Description
A set of constants used by service briefcases, service contexts, servers, etc. Implement this interface to get easy access to the constants.

Member Summary

Fields

`public static final` `ACTIVE`
The service component reaches this state when it has finished starting.

`public static final` `INACTIVE`
The service component is either newly created and not yet added, or recently removed from, a service environment.

`public static final` `INITIALIZED`
The service component reaches this state when it has finished initialization.

`public static final` `INITIALIZING`
The service component automatically reaches this state when the service context calls the initialize method of the service component.

`public static final` `JAR_ACTIVATOR`
The JAR manifest key to the activator of a service component.

`public static final` `JAR_CLASSPATH`
The JAR manifest key to the JAR-internal classpath that should be used when loading the service component.

`public static final` `JAR_DEPEND`
The JAR manifest key to the list of names of services (offered by other service components) that this service component depends upon.

`public static final` `JAR_EXPORT`
The JAR manifest key to the list of classes that this service component exports to other components.

`public static final` `JAR_IMPORT`
Currently not used.

`public static final` `JAR_NAME`
The JAR manifest key to the symbolic name of the service component.

`public static final` `JAR_PERMISSION`
The JAR manifest key to the list of permissions that grants the service component rights to functionality of the system.

`public static final` `RESUMED`
The service component reaches this state when it has finished resumption.

`public static final` `RESUMING`
The service component automatically reaches this state when the service context calls the resume method of the service component.

Member Summary	
public static final	SP_HOSTS The stationary property key to the list of servers that the service briefcase has visited.
public static final	STARTING The service component automatically reaches this state when the service context calls the start method of the service component.
public static final	stateNames An array of symbolic names of the states of the service component.
public static final	STOPPED The service component reaches this state when it has finished stopping.
public static final	STOPPING The service component automatically reaches this state when the service context calls the stop method of the service component.
public static final	SUSPENDED The service component reaches this state when it has finished suspension.
public static final	SUSPENDING The service component automatically reaches this state when the service context calls the suspend method of the service component.

Fields

ACTIVE

```
public static final int ACTIVE
```

The service component reaches this state when it has finished starting. This is the state where most of the lifecycle of a service component is spent. The service component is allowed to interact with both the service context and other service components from here.

INACTIVE

```
public static final int INACTIVE
```

The service component is either newly created and not yet added, or recently removed from, a service environment. In this state the service component is not allowed to interact with either its service context or with other service components.

INITIALIZED

```
public static final int INITIALIZED
```

The service component reaches this state when it has finished initialization.

INITIALIZING

```
public static final int INITIALIZING
```

The service component automatically reaches this state when the service context calls the initialize method of the service component. This is done as a first step to add the component to the service environment. In this state, the service component is expected to perform initialization that is only done once during the lifetime of a service component. This is the first chance of the service component to interact with the service context, but interaction with other service components is not allowed yet. The service component signals that initialization is done either by having the initialize method return `INITIALIZED`, or, if initialization continues after returning from the initialize method, by calling `ServiceContext.setState(int)`

on the service context. In the latter case a negative number should be returned by the initialize method to signal that initialization is not finished.

JAR_ACTIVATOR

```
public static final java.lang.String JAR_ACTIVATOR
```

The JAR manifest key to the activator of a service component. The value of this key should be the fully qualified class name of the class of the service component that implements the interface `ServiceComponent` .

JAR_CLASSPATH

```
public static final java.lang.String JAR_CLASSPATH
```

The JAR manifest key to the JAR-internal classpath that should be used when loading the service component. The value of this key should be a comma separated list of JAR entries (being themselves JAR files) or '.' (which stands for the classes in the root JAR file). List entries are searched for class definitions in order of appearance.

JAR_DEPEND

```
public static final java.lang.String JAR_DEPEND
```

The JAR manifest key to the list of names of services (offered by other service components) that this service component depends upon. The value of this key should be a comma separated list of service names.

JAR_EXPORT

```
public static final java.lang.String JAR_EXPORT
```

The JAR manifest key to the list of classes that this service component exports to other components. The value of this key should be a comma separated list of package names or fully qualified class names.

JAR_IMPORT

```
public static final java.lang.String JAR_IMPORT
```

Currently not used.

JAR_NAME

```
public static final java.lang.String JAR_NAME
```

The JAR manifest key to the symbolic name of the service component.

JAR_PERMISSION

```
public static final java.lang.String JAR_PERMISSION
```

The JAR manifest key to the list of permissions that grants the service component rights to functionality of the system. The value of this key should be a comma separated list of permission interfaces in package `se.sics.sview.core.permission` or fully qualified class names that implement (or extend) one or more of the permission interfaces in se.sics.sview.core.permission.

RESUMED

```
public static final int RESUMED
```

The service component reaches this state when it has finished resumption.

RESUMING

```
public static final int RESUMING
```

The service component automatically reaches this state when the service context calls the resume method of the service component. This is done as a first step to resume the component after suspension. This state is comparable to the `INITIALIZING` state, with the exception that the state can occur more than once. The service component signals that resumption is done either by having the resume method return `RESUMED`, or, if resumption continues after returning from the resume method, by calling `ServiceContext.setState(int)` on the service context. In the latter case a negative number should be returned by the resume method to signal that resumption is not finished.

SP_HOSTS

```
public static final java.lang.String SP_HOSTS
```

The stationary property key to the list of servers that the service briefcase has visited. The value of this key is used when the service briefcase is synchronized with other servers.

STARTING

```
public static final int STARTING
```

The service component automatically reaches this state when the service context calls the start method of the service component. In this state, the service component should perform tasks that should be done every time it is about to start. Interaction with the service context is allowed, but not with other service components. The service component signals that starting is done either by having the start method return `ACTIVE`, or, if starting continues after returning from the start method, by calling `ServiceContext.setState(int)` on the service context. In the latter case a negative number should be returned by the start method to signal that starting is not finished.

stateNames

```
public static final java.lang.String[] stateNames
```

An array of symbolic names of the states of the service component. The value of the state variables of this class work as index to its corresponding name.

STOPPED

```
public static final int STOPPED
```

The service component reaches this state when it has finished stopping. In this state the service component has reached the end of its lifecycle. Only a reload of a previously saved copy or creating a new instance of the service component can bring the service component back to the service environment. In this state the service component is not allowed to interact with either its service context or other service components.

STOPPING

```
public static final int STOPPING
```

The service component automatically reaches this state when the service context calls the stop method of the service component. This is done as a first step to stop the component. In this state, the service component is expected to unregister all services that it offers other service components, as well as unsubscribe to

services of other service components. The service component is allowed to interact with the service context in this state. It is also allowed to interact with other service components, but only for the purpose of handling unsubscriptions and unregistrations. The service component signals that stopping is done either by having the stop method return `STOPPED`, or, if stopping continues after returning from the stop method, by calling `ServiceContext.setState(int)` on the service context. In the latter case a negative number should be returned by the stop method to signal that stopping is not finished.

SUSPENDED

```
public static final int SUSPENDED
```

The service component reaches this state when it has finished suspension. In this state the service component is not allowed to interact with either its service context or other service components. The service component can now be saved to persistent media or moved to another server.

SUSPENDING

```
public static final int SUSPENDING
```

The service component automatically reaches this state when the service context calls the suspend method of the service component. This is done as a first step to suspend the component. In this state, the service component is expected to unregister all services that it offers other service components, as well as unsubscribe to services of other service components. The service component is allowed to interact with the service context in this state. It is also allowed to interact with other service components, but only for the purpose of handling unsubscriptions and unregistrations. The service component signals that suspension is done either by having the suspend method return `SUSPENDED`, or, if suspension continues after returning from the suspend method, by calling `ServiceContext.setState(int)` on the service context. In the latter case a negative number should be returned by the suspend method to signal that suspension is not finished.

se.sics.sview.core
Mobile

Declaration
```
public interface Mobile
```

Description
A `service component` that implements this interface will be included during service briefcase synchronization.

se.sics.sview.core
Monitor

Declaration

```
public class Monitor

java.lang.Object
  |
  +--se.sics.sview.core.Monitor
```

Description

Implements a 'one-to-many monitor' for exclusive access to critical sections. When no one has arrogated ownership of the monitor, everyone are free to enter and exit at will. Simultaneous consumers are not synchronized (except during the very brief call to method `enter`).

A call to `arrogate` will claim ownership of the monitor, and with that exclusive access to sections that are guarded by this monitor. Method `renounce` release ownership of the monitor.

For example, protect a code section with:

```
...
enter();
// perform protected actions
exit();
...
```

and claim ownership with:

```
...
Object monitorReference = new Object();
synchronized(monitorReference);
 arrogate(monitorReference);
 // perform actions that require exclusive ownership
 renounce();
}
...
```

Member Summary

Constructors

public	`Monitor()`

Methods

public synchronized void	`arrogate(Object)` Arrogate exclusive access to the monitor.
public void	`enter()` Enter monitor.
public synchronized void	`exit()` Exit monitor.

Member Summary	
public void	renounce() Renounce exclusive access to the monitor.

Inherited Member Summary
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

Monitor()

```
public Monitor()
```

Methods

arrogate(Object)

```
public synchronized void arrogate(java.lang.Object ref)
            throws InterruptedException
```

Arrogate exclusive access to the monitor. This method will block until exclusive ownership can be realized. This requires two conditions to be met: all consumers must leave the monitor and the monitor cannot be arrogated by someone else.

Parameters:
`ref` - the object on which the monitor will synchronize consumers

Throws:
`InterruptedException` - if interrupted while waiting for another owner to renounce ownership or the consumers to leave the monitor

enter()

```
public void enter()
```

Enter monitor. If monitor is locked, this method will block.

exit()

```
public synchronized void exit()
```

Exit monitor.

renounce()

```
public void renounce()
```

Renounce exclusive access to the monitor.

se.sics.sview.core
ObjectInputStreamLoader

Declaration

```
public class ObjectInputStreamLoader extends java.io.ObjectInputStream

java.lang.Object
  |
  +--java.io.InputStream
        |
        +--java.io.ObjectInputStream
              |
              +--se.sics.sview.core.ObjectInputStreamLoader
```

All Implemented Interfaces: java.io.DataInput, java.io.ObjectInput, java.io.ObjectStreamConstants

Description

This subclass of ObjectInputStream delegates loading of classes to an existing ClassLoader. This code is adopted from the SUN MICROSYSTEM's BDK 1.1 release. Excerpt from BDK 1.1 License: *2. Redistribution of Demonstration Files. Sun grants Licensee the right to use, modify and redistribute the Beans example and demonstration code, including the Bean Box ("Demos"), in both source and binary code form provided that (i) Licensee does not utilize the Demos in a manner which is disparaging to Sun; and (ii) Licensee indemnifies and holds Sun harmless from all claims relating to any such use or distribution of the Demos. Such distribution is limited to the source and binary code of the Demos and specifically excludes any rights to modify or distribute any graphical images contained in the demos.*

Member Summary

Constructors

public	`ObjectInputStreamLoader(InputStream, ClassLoader)`
	Loader must be non-null;

Methods

public ClassLoader	`getClassLoader()`
protected Class	`resolveClass(ObjectStreamClass)`
	Use the given ClassLoader rather than using the system class

Inherited Member Summary

Fields inherited from interface java.io.ObjectStreamConstants

ObjectInputStreamLoader(InputStream, ClassLoader)

Inherited Member Summary

PROTOCOL_VERSION_1, PROTOCOL_VERSION_2, SC_BLOCK_DATA, SC_EXTERNALIZABLE, SC_SERIALIZABLE, SC_WRITE_METHOD, STREAM_MAGIC, STREAM_VERSION, SUBCLASS_IMPLEMENTATION_PERMISSION, SUBSTITUTION_PERMISSION, TC_ARRAY, TC_BASE, TC_BLOCKDATA, TC_BLOCKDATALONG, TC_CLASS, TC_CLASSDESC, TC_ENDBLOCKDATA, TC_EXCEPTION, TC_LONGSTRING, TC_MAX, TC_NULL, TC_OBJECT, TC_PROXYCLASSDESC, TC_REFERENCE, TC_RESET, TC_STRING, baseWireHandle

Methods inherited from class java.io.InputStream

mark, markSupported, read, reset, skip

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methods inherited from interface java.io.ObjectInput

read, skip

Methods inherited from class java.io.ObjectInputStream

available, close, defaultReadObject, enableResolveObject, read, read, readBoolean, readByte, readChar, readClassDescriptor, readDouble, readFields, readFloat, readFully, readFully, readInt, readLine, readLong, readObject, readObjectOverride, readShort, readStreamHeader, readUTF, readUnsignedByte, readUnsignedShort, registerValidation, resolveObject, resolveProxyClass, skipBytes

Constructors

ObjectInputStreamLoader(InputStream, ClassLoader)

```
public ObjectInputStreamLoader(java.io.InputStream in, java.lang.ClassLoader loader)
            throws IOException, StreamCorruptedException
```

Loader must be non-null;

Throws:

StreamCorruptedException, IOException

Methods

getClassLoader()

```
public java.lang.ClassLoader getClassLoader()
```

resolveClass(ObjectStreamClass)

```
protected java.lang.Class resolveClass(java.io.ObjectStreamClass classDesc)
            throws IOException, ClassNotFoundException
```

Use the given ClassLoader rather than using the system class

Overrides: java.io.ObjectInputStream.resolveClass(java.io.ObjectStreamClass) in class java.io.ObjectInputStream

Throws:
`ClassNotFoundException`, `IOException`

se.sics.sview.core
PermissionDeniedException

Declaration

```
public class PermissionDeniedException extends java.lang.RuntimeException

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--se.sics.sview.core.PermissionDeniedException
```

All Implemented Interfaces: java.io.Serializable

Description

Thrown if a service component attempts to call a method that it does not hold the permission to call.

Member Summary	
Constructors	
public	PermissionDeniedException() Constructs an PermissionDeniedException with null as its error detail message.
public	PermissionDeniedException(String) Constructs an PermissionDeniedException with the specified detail message.

Inherited Member Summary
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait
Methods inherited from class java.lang.Throwable
fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

Constructors

PermissionDeniedException()

```
public PermissionDeniedException()
```

Constructs an `PermissionDeniedException` with `null` as its error detail message.

PermissionDeniedException(String)

```
public PermissionDeniedException(java.lang.String s)
```

Constructs an `PermissionDeniedException` with the specified detail message. The error message string `s` can later be retrieved by the `java.lang.Throwable.getMessage()` method of class `java.lang.Throwable`.

Parameters:

`s` - the detail message.

se.sics.sview.core
Persistent

Declaration
```
public interface Persistent extends java.io.Serializable
```

All Superinterfaces: java.io.Serializable

Description
A `service component` that implements this interface will be offered to save its state during service briefcase synchronization and save.

Member Summary	
Methods	
`public boolean`	`freeze()` The service briefcase calls the `freeze` method when it saves the service component.
`public void`	`thaw()` The service briefcase calls the thaw method when a saved version of the service component is loaded.

Methods

freeze()
```
public boolean freeze()
```

The service briefcase calls the `freeze` method when it saves the service component. This occurs after the service component has reached state `SUSPENDED`, but before state `RESUMING` is reached. The `freeze` method should be used to prepare for serialization by optimizing or removing data structures. The service component could e.g. compact a hash table or empty a media cache for more efficient storage After returning from the `freeze` method all external references (such as references to the service context, file and socket handles etc.) must have been set to `null`.

Returns: `true` if the component has changed since last save, `false` otherwise.

thaw()
```
public void thaw()
```

The service briefcase calls the thaw method when a saved version of the service component is loaded. This occurs after the `freeze` method has been called (possibly in a different VM and on a different host), but before state `RESUMING` is reached. The `thaw` method should be used to, if needed, recreate data structures that were removed or converted in the `freeze` method. It should also be used to re-associate references that were set to `null` in the `freeze` method or during serialization.

se.sics.sview.core
ServerProxy

Declaration
```
public interface ServerProxy
```

Description
This interface should be used to wrap implementations of remote service briefcase server communication and transaction initiator to transaction participant communication.

Member Summary	
Methods	
public String	getProtocol() Returns the protocol that this server proxy implements.
public ServiceBriefcaseServer	getServiceBriefcaseServerProxy(String) Creates a new proxy to a service briefcase server.
public TransactionParticipant	getTransactionParticipantProxy(String, String) Creates a new proxy to a transaction participant.
public void	initialize(ServiceBriefcaseServer, String[]) This method should be called by the service briefcase server before any calls to getServiceBriefcaseServerProxy(String) or getTransactionParticipantProxy(String, String) are made.

Methods

getProtocol()
```
public java.lang.String getProtocol()
```
Returns the protocol that this server proxy implements.

Returns: the protocol that this server proxy implements

getServiceBriefcaseServerProxy(String)
```
public ServiceBriefcaseServer getServiceBriefcaseServerProxy(java.lang.String uri)
            throws Exception
```
Creates a new proxy to a service briefcase server.

Parameters:
`uri` - the URI to the service briefcase server to connect to.

Returns: a reference to a service briefcase server that represents the remote server.

Throws:
`Exception`

getTransactionParticipantProxy(String, String)

```
public TransactionParticipant getTransactionParticipantProxy(java.lang.String uri,
            java.lang.String id)
            throws Exception
```

Creates a new proxy to a transaction participant.

Parameters:

`uri` - the URI to the transaction server to connect to.

Returns: a reference to a transaction server that represents the remote server.

Throws:

`Exception`

initialize(ServiceBriefcaseServer, String[])

```
public void initialize(ServiceBriefcaseServer localServer, java.lang.String[] args)
            throws Exception
```

This method should be called by the service briefcase server before any calls to `getServiceBriefcaseServerProxy(String)` or `getTransactionParticipant-Proxy(String, String)` are made.

Parameters:

`localServer` - a reference to the local service briefcase server.

`args` - parameters to the service briefcase server.

Throws:

`Exception`

se.sics.sview.core
ServiceBriefcase

Declaration

```
public class ServiceBriefcase implements java.io.Serializable

java.lang.Object
  |
  +--se.sics.sview.core.ServiceBriefcase
```

All Implemented Interfaces: java.io.Serializable

Description

Contains functionality for creating, loading and saving service components. It also provides storage of the JAR files of service components, persistent service components, and properties.

The service briefcase is serializable and it can be stored on persistent media and sent between servers, or have its content synchronized with service briefcases on other servers.

Much of the functionality of the service briefcase is delegated to service containers `ServiceContainer` , of which there is one for each service component in the briefcase. The service container provides storage and serialization handling of individual service components. It includes functionality for creating, loading, and saving service components, storing persistent service components, and caching the JAR file of service components.

Service component creation and loading requires that a class loader is provided by the server implementation. The server typically uses separate class loaders for every service component in the system. This ensures that no service component should be able to manipulate other service components without permission.

Member Summary

Constructors

public	`ServiceBriefcase(Properties, Properties, String, String)` Creates a new service briefcase with predefined mobile and stationary properties.
public	`ServiceBriefcase(String, String)` Creates a new empty service briefcase.

Methods

public void	`arrogateMonitor(Object)` Locks this briefcase (see `Monitor.arrogate(Object)`).
public final boolean	`authenticate(String, String)` Authennticates the owner of this briefcase.
public final void	`changePassword(String, String, String)` Changes the password of this briefcase.
public void	`enterMonitor()` Enter monitor.
public void	`exitMonitor()` Exit monitor (see `Monitor.exit()`).
public Properties	`getMobileProperties(String, String)` Returns the mobile properties of this briefcase.

ServiceBriefcase(Properties, Properties, String, String)

Member Summary

public ServiceContainer	`getServiceComponents(String[], String, String)` Returns the service components that corresponds to the given set of keys.
public ServiceContainer	`getServiceContainer(String, String, String)` Returns the service container that corresponds to the given key.
public synchronized String	`getServiceKeys(String, String)` Returns an array that contains the keys of the service containers in this briefcase.
public Properties	`getState(String, String)` Returns the current state of this briefcase.
public Properties	`getStationaryProperties(String, String)` Returns the stationary properties of this briefcase.
public static ServiceBriefcase	`load(InputStream)` Loads a serialized service briefcase from a given input stream.
public synchronized void	`putServiceContainer(ServiceContainer, String, String)` Adds/overwrites a service container.
public synchronized void	`removeServiceContainer(String, String, String)` Removes the service container that corresponds to the given key.
public void	`renounceMonitor()` Unlocks this briefcase (see `Monitor.renounce()`).
public static void	`save(ServiceBriefcase, OutputStream)` Saves service briefcase to a given output stream.
public void	`setMobileProperties(Properties, String, String)` Sets the mobile properties of this briefcase.
public synchronized void	`setMonitor(Monitor)` Sets the monitor for this briefcase (see `Monitor`).
public void	`setStationaryProperties(Properties, String, String)` Sets the stationary properties of this briefcase.
public ServiceBriefcase	`toMobile(String, String)` Creates a new service briefcase with all mobile properties and service components of this briefcase.
public void	`updateServiceBriefcase(ServiceContainer[], Properties, String, String)` Updates this briefcase with a new set of service components and mobile properties.

Inherited Member Summary

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

ServiceBriefcase(Properties, Properties, String, String)

```
public ServiceBriefcase(java.util.Properties mobileProps,
            java.util.Properties stationaryProps, java.lang.String uid,
            java.lang.String pwd)
```

Creates a new service briefcase with predefined mobile and stationary properties.

Parameters:

`mobileProps` - mobile properties for the new briefcase

`stationaryProps` - stationary properties for the new briefcase

`uid` - the user id of the owner of this briefcase

`pwd` - the password of the owner of this briefcase

ServiceBriefcase(String, String)

```
public ServiceBriefcase(java.lang.String uid, java.lang.String pwd)
```

Creates a new empty service briefcase.

Parameters:

`uid` - the user id of the owner of this briefcase

`pwd` - the password of the owner of this briefcase

Methods

arrogateMonitor(Object)

```
public void arrogateMonitor(java.lang.Object ref)
            throws InterruptedException
```

Locks this briefcase (see `Monitor.arrogate(Object)`).

Parameters:

`ref` - the object on which the monitor of the service briefcase will synchronize consumers

Throws:

`InterruptedException`

authenticate(String, String)

```
public final boolean authenticate(java.lang.String uid, java.lang.String pwd)
```

Authennticates the owner of this briefcase.

Parameters:

`uid` - the user id of the owner of this briefcase

`pwd` - the password of the owner of this briefcase

Returns: `true` if the authentication succeeds, `false` otherwise

changePassword(String, String, String)

```
public final void changePassword(java.lang.String uid, java.lang.String oldPwd,
            java.lang.String newPwd)
```

Changes the password of this briefcase.

Parameters:

`uid` - the user id of the owner of this briefcase

`oldPwd` - the old password of the owner of this briefcase

`newPwd` - the new password of the owner of this briefcase

enterMonitor()

```
public void enterMonitor()
```

Enter monitor. If monitor is locked, this method will block (see `Monitor.enter()`).

exitMonitor()

```
public void exitMonitor()
```

Exit monitor (see `Monitor.exit()`).

getMobileProperties(String, String)

```
public java.util.Properties getMobileProperties(java.lang.String uid,
            java.lang.String pwd)
```

Returns the mobile properties of this briefcase.

Parameters:

`uid` - the user id of the owner of this briefcase

`pwd` - the password of the owner of this briefcase

Returns: the mobile properties of this service briefcase

getServiceComponents(String[], String, String)

```
public ServiceContainer[] getServiceComponents(java.lang.String[] keys,
            java.lang.String uid, java.lang.String pwd)
```

Returns the service components that corresponds to the given set of keys.

Parameters:

`keys` - an array of keys that corresponds to the set of service components to get

`uid` - the user id of the owner of this briefcase

`pwd` - the password of the owner of this briefcase

Returns: an array of service container that corresponds to the specified array of service component keys

getServiceContainer(String, String, String)

```
public ServiceContainer getServiceContainer(java.lang.String key, java.lang.String uid,
            java.lang.String pwd)
```

Returns the service container that corresponds to the given key.

Parameters:

`key` - the key of the service container

`uid` - the user id of the owner of this briefcase

`pwd` - the password of the owner of this briefcase

Returns: the service container with the specified key

getServiceKeys(String, String)

```
public synchronized java.lang.String[] getServiceKeys(java.lang.String uid,
            java.lang.String pwd)
```

Returns an array that contains the keys of the service containers in this briefcase.

Parameters:
`uid` - the user id of the owner of this briefcase

`pwd` - the password of the owner of this briefcase

Returns: an array of the keys of the service containers in this briefcase

getState(String, String)

```
public java.util.Properties[] getState(java.lang.String uid, java.lang.String pwd)
```

Returns the current state of this briefcase.

Parameters:
`uid` - the user id of the owner of this briefcase

`pwd` - the password of the owner of this briefcase

Returns: the state of this service briefcase as an array of service container properties

getStationaryProperties(String, String)

```
public java.util.Properties getStationaryProperties(java.lang.String uid,
            java.lang.String pwd)
```

Returns the stationary properties of this briefcase.

Parameters:
`uid` - the user id of the owner of this briefcase

`pwd` - the password of the owner of this briefcase

Returns: the stationary properties of this service briefcase

load(InputStream)

```
public static ServiceBriefcase load(java.io.InputStream is)
            throws IOException, ClassNotFoundException
```

Loads a serialized service briefcase from a given input stream.

NOTE! Service briefcases, in order to be loaded properly, must be loaded with this method.

Parameters:
`is` - an input stream from which a serialized service briefcase should be read

Returns: the loaded service briefcase

Throws:
`ClassNotFoundException`, `IOException`

putServiceContainer(ServiceContainer, String, String)

```
public synchronized void putServiceContainer(ServiceContainer service,
            java.lang.String uid, java.lang.String pwd)
```

Adds/overwrites a service container.

Parameters:
`service` - the new service container

`uid` - the user id of the owner of this briefcase

`pwd` - the password of the owner of this briefcase

removeServiceContainer(String, String, String)

```
public synchronized void removeServiceContainer(java.lang.String key,
            java.lang.String uid, java.lang.String pwd)
```

Removes the service container that corresponds to the given key.

Parameters:

`key` - the key of the service container

`uid` - the user id of the owner of this briefcase

`pwd` - the password of the owner of this briefcase

renounceMonitor()

```
public void renounceMonitor()
```

Unlocks this briefcase (see `Monitor.renounce()`).

save(ServiceBriefcase, OutputStream)

```
public static void save(ServiceBriefcase sb, java.io.OutputStream os)
            throws IOException
```

Saves service briefcase to a given output stream.

NOTE! Service briefcases, in order to be saved properly, must be saved with this method.

Parameters:

`sb` - the service briefcase to save

`os` - the output stream to save the briefcase to save

Throws:

`IOException`

setMobileProperties(Properties, String, String)

```
public void setMobileProperties(java.util.Properties props, java.lang.String uid,
            java.lang.String pwd)
```

Sets the mobile properties of this briefcase.

Parameters:

`props` - the new mobile properties

`uid` - the user id of the owner of this briefcase

`pwd` - the password of the owner of this briefcase

setMonitor(Monitor)

```
public synchronized void setMonitor(Monitor monitor)
```

Sets the monitor for this briefcase (see `Monitor`).

Parameters:

`monitor` - the new monitor

setStationaryProperties(Properties, String, String)

```
public void setStationaryProperties(java.util.Properties props, java.lang.String uid,
            java.lang.String pwd)
```

Sets the stationary properties of this briefcase.

Parameters:

`props` - the new stationary properties

`uid` - the user id of the owner of this briefcase

`pwd` - the password of the owner of this briefcase

toMobile(String, String)

```
public ServiceBriefcase toMobile(java.lang.String uid, java.lang.String pwd)
```

Creates a new service briefcase with all mobile properties and service components of this briefcase.

Parameters:

`uid` - the user id of the owner of this briefcase

`pwd` - the password of the owner of this briefcase

Returns: a clone of this service briefcase containing only the mobile service components and the mobile properties

updateServiceBriefcase(ServiceContainer[], Properties, String, String)

```
public void updateServiceBriefcase(ServiceContainer[] serviceContainers,
            java.util.Properties mobileProperties, java.lang.String uid,
            java.lang.String pwd)
```

Updates this briefcase with a new set of service components and mobile properties.

Parameters:

`keys` - an array of service containers that should be updated

`uid` - the user id of the owner of this briefcase

`pwd` - the password of the owner of this briefcase

se.sics.sview.core
ServiceBriefcaseServer

Declaration

```
public interface ServiceBriefcaseServer
```

Description

This interfcase specifies an API to sView service briefcase servers. It specifies methods for exchanging service briefcases and starting and stopping PSEs.

Member Summary

Methods

public Properties	`getMobileProperties(String, String, String)` Returns the mobile properties of a service briefcase.
public String	`getRegisteredUsers()` Returns the users that has a service briefcase on this server.
public ServiceBriefcase	`getServiceBriefcase(String, String)` Returns a service briefcase.
public ServiceBriefcase	`getServiceBriefcase(String, String, Date)` Returns a backuped service briefcase.
public Properties	`getServiceBriefcaseState(String, String, String)` Returns an array containing the keys of the service components.
public ServiceContainer	`getServiceComponents(String, String, String[], String)` Returns an array of service components that corresponds to an array of service component keys.
public void	`newServiceBriefcase(String, String)` Creates a new service briefcase.
public void	`removeServiceBriefcase(String, String)` Removes a service briefcase from this server.
public void	`startPse(String, String)` Starts a service environment.
public void	`stopPse(String, String)` Stops a service environment.
public void	`updateServiceBriefcase(String, String, ServiceContainer[], Properties, String)` Updates a service briefcase of a remote service briefcase server with new service containers and properties.

Methods

getMobileProperties(String, String, String)

```
public java.util.Properties getMobileProperties(java.lang.String uid,
            java.lang.String pwd, java.lang.String transactionId)
            throws ServiceBriefcaseServerException
```

Returns the mobile properties of a service briefcase.

Parameters:
`uid` - the user id of the service briefcase's owner

`pwd` - the password of the service briefcase's owner

`transactionId` - an identifier of the transaction that the method responds to

Returns: mobile properties of a service briefcase

Throws:
`ServiceBriefcaseServerException` - if the service briefcase server does not contain a service briefcase for the specified uid

getRegisteredUsers()

```
public java.lang.String[] getRegisteredUsers()
            throws ServiceBriefcaseServerException
```

Returns the users that has a service briefcase on this server.

Returns: an array of service user names

Throws:
`ServiceBriefcaseServerException`

getServiceBriefcase(String, String)

```
public ServiceBriefcase getServiceBriefcase(java.lang.String uid, java.lang.String pwd)
            throws ServiceBriefcaseServerException
```

Returns a service briefcase.

Parameters:
`uid` - the user id of the service briefcase's owner

`pwd` - the password of the service briefcase's owner

Returns: the service briefcase of a user

Throws:
`ServiceBriefcaseServerException` - if the service briefcase server does not contain a service briefcase for the specified uid

getServiceBriefcase(String, String, Date)

```
public ServiceBriefcase getServiceBriefcase(java.lang.String uid, java.lang.String pwd,
            java.util.Date date)
            throws ServiceBriefcaseServerException
```

Returns a backuped service briefcase. The will return the version that was the latest at the time specified by the parameter date.

Parameters:
`uid` - the user id of the service briefcase's owner

`pwd` - the password of the service briefcase's owner

`date` - the lates timestamp of the service briefcase backup

Returns: a backuped service briefcase of a user

getServiceBriefcaseState(String, String, String)

Throws:

`ServiceBriefcaseServerException` - if the service briefcase server does not contain a service briefcase for the specified uid

`java.lang.UnsupportedOperationException` - if the service briefcase server does implement backup of service briefcases

getServiceBriefcaseState(String, String, String)

```
public java.util.Properties[] getServiceBriefcaseState(java.lang.String uid,
            java.lang.String pwd, java.lang.String transactionId)
            throws ServiceBriefcaseServerException
```

Returns an array containing the keys of the service components.

Parameters:

`uid` - the user id of the service briefcase's owner

`pwd` - the password of the service briefcase's owner

`serviceContainerProps` - an array of properties describing the invoker's set of mobile service components

`transactionId` - an identifier of the trunsaction that the method responds to

Returns: an array with update information

Throws:

`ServiceBriefcaseServerException` - if the service briefcase server does not contain a service briefcase for the specified uid

getServiceComponents(String, String, String[], String)

```
public ServiceContainer[] getServiceComponents(java.lang.String uid,
            java.lang.String pwd, java.lang.String[] keys,
            java.lang.String transactionId)
            throws ServiceBriefcaseServerException
```

Returns an array of service components that corresponds to an array of service component keys.

Parameters:

`uid` - the user id of the service briefcase's owner

`pwd` - the password of the service briefcase's owner

`keys` - an array of keys describing the service containers that should be fetched

`transactionId` - an identifier of the trunsaction that the method responds to

Returns: an array of service components

Throws:

`ServiceBriefcaseServerException` - if the service briefcase server does not contain a service briefcase for the specified uid

newServiceBriefcase(String, String)

```
public void newServiceBriefcase(java.lang.String uid, java.lang.String pwd)
            throws ServiceBriefcaseServerException
```

Creates a new service briefcase.

Parameters:

`uid` - the user id of the service briefcase's owner

`pwd` - the password of the service briefcase's owner

Throws:

`ServiceBriefcaseServerException` - if the service briefcase server already contains a service briefcase for the specified uid

removeServiceBriefcase(String, String)

```
public void removeServiceBriefcase(java.lang.String uid, java.lang.String pwd)
            throws ServiceBriefcaseServerException
```

Removes a service briefcase from this server.

Parameters:

`uid` - the user id of the service briefcase's owner

`pwd` - the password of the service briefcase's owner

Throws:

`ServiceBriefcaseServerException` - if the service briefcase server does not contain a service briefcase for the specified uid

startPse(String, String)

```
public void startPse(java.lang.String uid, java.lang.String pwd)
            throws ServiceBriefcaseServerException
```

Starts a service environment.

Parameters:

`uid` - the user id of the service briefcase's owner

`pwd` - the password of the service briefcase's owner

Throws:

`ServiceBriefcaseServerException` - if the service briefcase server does not contain a service briefcase for the specified uid

stopPse(String, String)

```
public void stopPse(java.lang.String uid, java.lang.String pwd)
            throws ServiceBriefcaseServerException
```

Stops a service environment.

Parameters:

`uid` - the user id of the service briefcase's owner

`pwd` - the password of the service briefcase's owner

Throws:

`ServiceBriefcaseServerException` - if the service briefcase server does not contain a service briefcase for the specified uid

updateServiceBriefcase(String, String, ServiceContainer[], Properties, String)

```
public void updateServiceBriefcase(java.lang.String uid, java.lang.String pwd,
            ServiceContainer[] serviceComponents,
```

updateServiceBriefcase(String, String, ServiceContainer[], Properties, String)

```
java.util.Properties mobileProperties, java.lang.String transactionId)
throws ServiceBriefcaseServerException
```

Updates a service briefcase of a remote service briefcase server with new service containers and properties.

Parameters:

`uid` - the user id of the service briefcase's owner

`pwd` - the password of the service briefcase's owner

`serviceComponents` - new service components for the service briefcase

`mobileProperties` - new mobile properties for the service briefcase

`transactionId` - an identifier of the trunsaction that the method responds to

Throws:

`ServiceBriefcaseServerException` - if the service briefcase server does not contain a service briefcase for the specified uid

se.sics.sview.core
ServiceBriefcaseServerException

Declaration

```
public class ServiceBriefcaseServerException extends java.lang.Exception

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--se.sics.sview.core.ServiceBriefcaseServerException
```

All Implemented Interfaces: java.io.Serializable

Description

This exception is thrown from the service briefcase server whenever a request experience fatal errors.

Member Summary

Fields

public	`detail` Nested Exception to hold wrapped exception.

Constructors

public	`ServiceBriefcaseServerException()` Constructs a `ServiceBriefcaseServerException` with no specified detail message.
public	`ServiceBriefcaseServerException(String)` Constructs a `ServiceBriefcaseServerException` with the specified detail message.
public	`ServiceBriefcaseServerException(String, Throwable)` Constructs a `ServiceBriefcaseServerException` with the specified detail message and nested exception.

Methods

public String	`getMessage()` Returns the detail message, including the message from the nested exception if there is one.
public void	`printStackTrace()` Prints the composite message to `System.err`.
public void	`printStackTrace(PrintStream)` Prints the composite message and the embedded stack trace to the specified stream `ps`.
public void	`printStackTrace(PrintWriter)` Prints the composite message and the embedded stack trace to the specified print writer `pw`

detail

Inherited Member Summary

Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`
Methods inherited from class java.lang.Throwable
`fillInStackTrace, getLocalizedMessage, toString`

Fields

detail

```
public java.lang.Throwable detail
```

Nested Exception to hold wrapped exception.

Constructors

ServiceBriefcaseServerException()

```
public ServiceBriefcaseServerException()
```

Constructs a `ServiceBriefcaseServerException` with no specified detail message.

ServiceBriefcaseServerException(String)

```
public ServiceBriefcaseServerException(java.lang.String s)
```

Constructs a `ServiceBriefcaseServerException` with the specified detail message.

Parameters:

`s` - the detail message

ServiceBriefcaseServerException(String, Throwable)

```
public ServiceBriefcaseServerException(java.lang.String s, java.lang.Throwable ex)
```

Constructs a `ServiceBriefcaseServerException` with the specified detail message and nested exception.

Parameters:

`s` - the detail message

`ex` - the nested exception

Methods

getMessage()

```
public java.lang.String getMessage()
```

Returns the detail message, including the message from the nested exception if there is one.

Overrides: java.lang.Throwable.getMessage() in class java.lang.Throwable

printStackTrace()

```
public void printStackTrace()
```

Prints the composite message to `System.err`.

Overrides: java.lang.Throwable.printStackTrace() in class java.lang.Throwable

printStackTrace(PrintStream)

```
public void printStackTrace(java.io.PrintStream ps)
```

Prints the composite message and the embedded stack trace to the specified stream `ps`.

Overrides: java.lang.Throwable.printStackTrace(java.io.PrintStream) in class java.lang.Throwable

Parameters:
`ps` - the print stream

printStackTrace(PrintWriter)

```
public void printStackTrace(java.io.PrintWriter pw)
```

Prints the composite message and the embedded stack trace to the specified print writer `pw`

Overrides: java.lang.Throwable.printStackTrace(java.io.PrintWriter) in class java.lang.Throwable

Parameters:
`pw` - the print writer

se.sics.sview.core
ServiceComponent

Declaration
```
public interface ServiceComponent
```

Description
Should be implemented by service components that wish to execute in an sView service briefcase. See `ServiceContext` for a description of the context in which the service component will execute.

Member Summary	
Methods	
`public int`	`initialize(ServiceContext, ServiceContextEvent)` Instructs the service component to initialize.
`public int`	`resume(ServiceContext, ServiceContextEvent)` Instructs the service component to resume.
`public int`	`start(ServiceContext, ServiceContextEvent)` Instructs the service component to start.
`public int`	`stop(ServiceContext, ServiceContextEvent)` Instructs the service component to stop.
`public int`	`suspend(ServiceContext, ServiceContextEvent)` Instructs the service component to suspend.

Methods

initialize(ServiceContext, ServiceContextEvent)

```
public int initialize(ServiceContext context, ServiceContextEvent evt)
```

Instructs the service component to initialize. The implementation of this method should execute fast. If initialization finish before the method terminates, it should return `Constants.INITIALIZED`. Otherwise it should return a negative value to indicate that initialization is ongoing. In this case the service component must call `ServiceContext.setState(int)` with `Constants.INITIALIZED` when initialization is done to signal that the service component is ready to start.

Parameters:

`context` - a handle to the service context

`evt` - an event with information regarding the cause of the state change

Returns: `Constants.INITIALIZED` if initialization is done, or a negative value if initialization is ongoing

See Also: `ServiceContextEvent`

resume(ServiceContext, ServiceContextEvent)

```
public int resume(ServiceContext context, ServiceContextEvent evt)
```

Instructs the service component to resume. The implementation of this method should execute fast. If resumption finish before the method terminates, it should return `Constants.RESUMED`. Otherwhise it should return a negative value to indicate that resumption is ongoing. In this case the service component must call `ServiceContext.setState(int)` with `Constants.RESUMED` when resumption is done to signal that the service component is ready to start.

Parameters:

`context` - a handle to the service context

`evt` - an event with information regarding the cause of the state change

Returns: `Constants.RESUMED` if resumption is done, or a negative value if resumption is ongoing

See Also: `ServiceContextEvent`

start(ServiceContext, ServiceContextEvent)

```
public int start(ServiceContext context, ServiceContextEvent evt)
```

Instructs the service component to start. The implementation of this method should execute fast. If the service component is started before the method terminates, it should return `Constants.ACTIVE`. Otherwhise it should return a negative value to indicate that starting is ongoing. In this case the service component must call `ServiceContext.setState(int)` with `Constants.ACTIVE` when the service component is started to signal that the service component is active.

Parameters:

`context` - a handle to the service context

`evt` - an event with information regarding the cause of the state change

Returns: `Constants.ACTIVE` if the service component is started, or a negative value if starting is ongoing

See Also: `ServiceContextEvent`

stop(ServiceContext, ServiceContextEvent)

```
public int stop(ServiceContext context, ServiceContextEvent evt)
```

Instructs the service component to stop. The implementation of this method should execute fast. If the service component is stopped before the method terminates, it should return `Constants.STOPPED`. Otherwhise it should return a negative value to indicate that stopping is ongoing. In this case the service component must call `ServiceContext.setState(int)` with `Constants.STOPPED` when the service component is stopped to signal that the service component can be terminated.

Parameters:

`context` - a handle to the service context

`evt` - an event with information regarding the cause of the state change

Returns: `Constants.STOPPED` if the service component is stopped, or a negative value if stoping is ongoing

See Also: `ServiceContextEvent`

suspend(ServiceContext, ServiceContextEvent)

```
public int suspend(ServiceContext context, ServiceContextEvent evt)
```

suspend(ServiceContext, ServiceContextEvent)

Instructs the service component to suspend. The implementation of this method should execute fast. If suspension finish before the method terminates, it should return `Constants.SUSPENDED`. Otherwhise it should return a negative value to indicate that suspension is ongoing. In this case the service component must call `ServiceContext.setState(int)` with `Constants.SUSPENDED` when suspension is done to signal that the service component is suspended.

Parameters:

`context` - a handle to the service context

`evt` - an event with information regarding the cause of the state change

Returns: `Constants.SUSPENDED` if suspension is done, or a negative value if suspension is ongoing

See Also: `ServiceContextEvent`

se.sics.sview.core
ServiceComponentEvent

Declaration

```
public class ServiceComponentEvent

java.lang.Object
  |
  +--se.sics.sview.core.ServiceComponentEvent
```

Description

The super class of all service component event.

Member Summary	
Constructors	
public	ServiceComponentEvent(String, String, int) Creates a new service component event with a service component key, name, and state.
Methods	
public String	getKey() Returns the key of the service component
public String	getName() Returns the name of the service component
public int	getState() Returns the state of the service component

Inherited Member Summary
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

ServiceComponentEvent(String, String, int)

```
public ServiceComponentEvent(java.lang.String key, java.lang.String name, int state)
```

Creates a new service component event with a service component key, name, and state.

Parameters:

key - the key of the service component

getKey()

`name` - the name of the service component

`state` - current state of the service component

Methods

getKey()

```
public java.lang.String getKey()
```

Returns the key of the service component

Returns: the key of the service component

getName()

```
public java.lang.String getName()
```

Returns the name of the service component

Returns: the name of the service component

getState()

```
public int getState()
```

Returns the state of the service component

Returns: the state of the service component

se.sics.sview.core
ServiceComponentListener

Declaration
`public interface` **`ServiceComponentListener`**

Description
A listener to service component state changes. See also `ServiceContext.addServiceComponentListener(String, ServiceComponentListener)` .

Member Summary	
Methods	
`public void`	`stateChanged(ServiceComponentEvent)` Called whenever the state of a service component has changed.

Methods

stateChanged(ServiceComponentEvent)

`public void` **`stateChanged`**`(ServiceComponentEvent evt)`

Called whenever the state of a service component has changed.

Parameters:

`evt` - an event describing the new state of a service component

se.sics.sview.core
ServiceComponentPermission

Declaration
`public interface` **`ServiceComponentPermission`**

All Known Subinterfaces:
`se.sics.sview.core.permission.AllPermissions`, `se.sics.sview.core.permission.ComponentHandling`, `se.sics.sview.core.permission.CreateComponent`, `se.sics.sview.core.permission.LoadComponent`, `se.sics.sview.core.permission.OtherPermissionHandling`, `se.sics.sview.core.permission.OwnPermissionHandling`, `se.sics.sview.core.permission.PermissionHandling`, `se.sics.sview.core.permission.ReloadEnvironment`, `se.sics.sview.core.permission.RemoveComponent`, `se.sics.sview.core.permission.ResumeComponent`, `se.sics.sview.core.permission.RuntimeHandling`, `se.sics.sview.core.permission.SaveComponent`, `se.sics.sview.core.permission.SaveEnvironment`, `se.sics.sview.core.permission.ServiceComponentHandling`, `se.sics.sview.core.permission.ServiceEnvironmentHandling`, `se.sics.sview.core.permission.ShutdownEnvironment`, `se.sics.sview.core.permission.StopComponent`, `se.sics.sview.core.permission.SuspendComponent`, `se.sics.sview.core.permission.SynchronizeEnvironment`

Description
Superclass of all service component permissions. See the interfaces in package `se.sics.sview.core.permission` for a full listing of predefined permissions.

Member Summary	
Fields	
`public static final`	`description` A textual description of the permission.

Fields

description
`public static final java.lang.String` **`description`**

A textual description of the permission. Override this field to describe what the permission grants access to.

se.sics.sview.core
ServiceContainer

Declaration

```
public class ServiceContainer implements java.io.Serializable, java.lang.Cloneable

java.lang.Object
  |
  +--se.sics.sview.core.ServiceContainer
```

All Implemented Interfaces: java.lang.Cloneable, java.io.Serializable

Description

A ServiceContainer wraps a service component, a JAR cache, and information about the service.

Member Summary

Fields

protected	activator
protected	changeDate
protected	creationDate
protected	jarCache
protected	key
protected	mobile
protected transient	monitor
public static final	P_CACHEDATE
public static final	P_CHANGEDATE
public static final	P_CREATIONDATE
public static final	P_JARURL
public static final	P_KEY
public static final	P_MOBILE
public static final	P_PERSISTENT
protected	persistent
protected	serviceComponent

Constructors

public	ServiceContainer(String, String) Loads a service component specification and creates a new container for a service component with the given key.

Methods

public ServiceComponent	createServiceComponent(ClassLoader) Creates a new service component based on the currently cached specification (the JAR file).
public Date	getCacheDate() Returns the date of the current version of the JAR file.
public Date	getChangeDate() Returns the date of the latest change of the service component in this container.
public Date	getCreationDate() Returns the creation date of the service component in this container.

ServiceContainer se.sics.sview.core

description

Member Summary

public byte	`getJar()` Returns the specification of the service component as a byte array.
public InputStream	`getJarStream()` Returns the specification of the service component in an input stream.
public String	`getJarUrl()` Returns the URL of the original specification of the service component in this container (the JAR file).
public String	`getKey()` Returns the key of the service component.
public Properties	`getProperties()` Returns the properties of this service container.
public byte	`getServiceComponent()` Returns the service component of this container.
public boolean	`isMobile()` Returns `true` if the service component has been declared as mobile (see `Mobile`).
public boolean	`isPersistent()` Returns `true` if the service component has been declared as persistent (see `Persistent`).
public static ServiceContainer	`load(InputStream, String)` Loads a serialized service container from a given input stream.
public ServiceComponent	`loadServiceComponent(ClassLoader)` Loads a previously saved copy of the service component in this container.
public void	`merge(ServiceContainer)` Merges the content of the given service container to this service container.
public void	`removeJar()` Empties the JAR cache.
public static void	`save(ServiceContainer, OutputStream)` Saves a serialized service container to a given output stream.
public void	`saveServiceComponent(ServiceComponent)` Saves the service component of this container.
protected void	`setChangeDate(Date)` Sets the change date of service component in this container.
protected void	`setCreationDate(Date)` Sets the creation date of service component in this container.
public void	`setMonitor(Monitor)` Sets the monitor of this service container (see `Monitor`).
public void	`setServiceComponent(byte[])` Sets the service component of this container.
public ServiceContainer	`stripJar()` Returns a clone of this container, without the JAR file.
public ServiceContainer	`stripServiceComponent()` Returns a clone of this container, without the service component.
public String	`toString()` Returns a string representation of this container.
public void	`validateJar()` Validates the JAR file of this service component.

Inherited Member Summary

Methods inherited from class java.lang.Object

Inherited Member Summary
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Fields

activator

protected java.lang.String **activator**

changeDate

protected java.util.Date **changeDate**

creationDate

protected java.util.Date **creationDate**

jarCache

protected se.sics.sview.core.ServiceContainer.JarCache **jarCache**

key

protected java.lang.String **key**

mobile

protected boolean **mobile**

monitor

protected transient Monitor **monitor**

P_CACHEDATE

public static final java.lang.String **P_CACHEDATE**

P_CHANGEDATE

public static final java.lang.String **P_CHANGEDATE**

P_CREATIONDATE

public static final java.lang.String **P_CREATIONDATE**

P_JARURL

public static final java.lang.String **P_JARURL**

P_KEY

public static final java.lang.String **P_KEY**

P_MOBILE

P_MOBILE

```
public static final java.lang.String P_MOBILE
```

P_PERSISTENT

```
public static final java.lang.String P_PERSISTENT
```

persistent

```
protected boolean persistent
```

serviceComponent

```
protected byte[] serviceComponent
```

Constructors

ServiceContainer(String, String)

```
public ServiceContainer(java.lang.String jarUrl, java.lang.String key)
            throws MalformedURLException, IOException
```

Loads a service component specification and creates a new container for a service component with the given key. This method does not *create* the service component per se.

Parameters:

`jarUrl` - a URL to the JAR file of the service component

`key` - the service component key

Throws:

`IOException`, `MalformedURLException`

Methods

createServiceComponent(ClassLoader)

```
public ServiceComponent createServiceComponent(java.lang.ClassLoader loader)
            throws IOException, ClassNotFoundException, InstantiationException, Illegal
            AccessException
```

Creates a new service component based on the currently cached specification (the JAR file).

Parameters:

`loader` - the class loader to use when creating the service component

Returns: the newly created service component

Throws:

`IllegalAccessException`, `InstantiationException`, `ClassNotFoundException`, `IOException`

getCacheDate()

```
public java.util.Date getCacheDate()
```

Returns the date of the current version of the JAR file.

Returns: the latest cache date

getChangeDate()

```
public java.util.Date getChangeDate()
```

Returns the date of the latest change of the service component in this container.

Returns: the change date

getCreationDate()

```
public java.util.Date getCreationDate()
```

Returns the creation date of the service component in this container.

Returns: the creation date

getJar()

```
public byte[] getJar()
```

Returns the specification of the service component as a byte array.

Returns: a byte array with the JAR file of the service component

getJarStream()

```
public java.io.InputStream getJarStream()
```

Returns the specification of the service component in an input stream.

Returns: an input stream with the JAR file of the service component

getJarUrl()

```
public java.lang.String getJarUrl()
```

Returns the URL of the original specification of the service component in this container (the JAR file).

Returns: the URL of the JAR file of the service component

getKey()

```
public java.lang.String getKey()
```

Returns the key of the service component.

Returns: the key of the service component

getProperties()

```
public java.util.Properties getProperties()
```

Returns the properties of this service container. The properties are currently value of the service component's key, creation date, and change date. The JAR URL of the original specification of the service component and the date of the latest caching of the JAR file. The properties also includes values that show whether the service component is persistent and/or mobile.

Returns: the properties of this service container

getServiceComponent()

```
public byte[] getServiceComponent()
```

Returns the service component of this container.

Returns: the service component of this container as an array of bytes

isMobile()

```
public boolean isMobile()
```

Returns `true` if the service component has been declared as mobile (see `Mobile`).

Returns: `true` if the service component has been declared as mobile, `false` otherwise

isPersistent()

```
public boolean isPersistent()
```

Returns `true` if the service component has been declared as persistent (see `Persistent`).

Returns: `true` if the service component has been declared as persistent, `false` otherwise

load(InputStream, String)

```
public static ServiceContainer load(java.io.InputStream is, java.lang.String key)
            throws IOException, ClassNotFoundException
```

Loads a serialized service container from a given input stream.

NOTE! Service containers, in order to be loaded properly, must be loaded with this method.

Parameters:

`is` - an input stream from which a serialized service briefcase should be read

`key` - the new key of the loaded service component

Returns: the loaded service briefcase

Throws:

`ClassNotFoundException`, `IOException`

loadServiceComponent(ClassLoader)

```
public ServiceComponent loadServiceComponent(java.lang.ClassLoader loader)
            throws IOException, ClassNotFoundException
```

Loads a previously saved copy of the service component in this container.

Parameters:

`loader` - the class loader to use when loading the service component

Returns: the newly loaded service component

Throws:

`ClassNotFoundException`, `IOException`

merge(ServiceContainer)

```
public void merge(ServiceContainer sc)
```

Merges the content of the given service container to this service container.

Parameters:
`sc` - the service container to merge to this container

removeJar()

```
public void removeJar()
```

Empties the JAR cache.

save(ServiceContainer, OutputStream)

```
public static void save(ServiceContainer sc, java.io.OutputStream os)
            throws IOException
```

Saves a serialized service container to a given output stream.

NOTE! Service containers, in order to be saved properly, must be saved with this method.

Parameters:
`sb` - the service container to save

`os` - the output stream to save the container to save

Throws:
`IOException`

saveServiceComponent(ServiceComponent)

```
public void saveServiceComponent(ServiceComponent s)
            throws IOException
```

Saves the service component of this container.

Parameters:
`s` - the service component to save

Throws:
`IOException`

setChangeDate(Date)

```
protected void setChangeDate(java.util.Date changeDate)
```

Sets the change date of service component in this container.

Parameters:
`changeDate` - the new change date

setCreationDate(Date)

```
protected void setCreationDate(java.util.Date creationDate)
```

Sets the creation date of service component in this container.

Parameters:
`creationDate` - the new creation date

setMonitor(Monitor)

```
public void setMonitor(Monitor monitor)
```

Sets the monitor of this service container (see `Monitor`).

setServiceComponent(byte[])

Parameters:
`monitor` - the new monitor

setServiceComponent(byte[])

```
public void setServiceComponent(byte[] serviceComponent)
```

Sets the service component of this container.

Parameters:
`serviceComponent` - the new service component

stripJar()

```
public ServiceContainer stripJar()
```

Returns a clone of this container, without the JAR file.

Returns: the stripped service container

stripServiceComponent()

```
public ServiceContainer stripServiceComponent()
```

Returns a clone of this container, without the service component.

Returns: the stripped service container

toString()

```
public java.lang.String toString()
```

Returns a string representation of this container.

Overrides: java.lang.Object.toString() in class java.lang.Object

Returns: a string representation of this container

validateJar()

```
public void validateJar()
```

Validates the JAR file of this service component. If it turns out that it is old, it will be updated.

se.sics.sview.core
ServiceContext

Declaration

```
public interface ServiceContext extends se.sics.sview.core.Constants
```

All Superinterfaces: Constants

Description

This interface specifies an API to the runtime environment of a service briefcase. It specifies methods for handling service components (creation, maintenance, removal, etc.), the runtime environment (save, synchronize, reload, and shutdown), and the state of the service component.

A service component has access to three types of properties via its service context: local, stationary, and mobile. *Local* properties are controlled by the administrator of the server on which the service environment executes. These properties can be read but not be set or modified by service components. *Stationary* properties can both be read, set, and modified by service components. Stationary properties are local to a specific server. *Mobile* properties can both be read, set, and modified by service components. Mobile properties follow the service briefcase as it migrates from server to server.

Member Summary

Methods

public void	addServiceComponentListener(String, ServiceComponentListener) Adds a listener to service component events of a specified service component.
public void	addServiceContextListener(ServiceContextListener) Adds a listener to service context events.
public void	createServiceComponent(String) Creates and adds a service component based on a JAR file containing a specification of a service component.
public String	getJarAttribute(String) Gets an attribute from the JAR file of the service component.
public byte	getJarEntry(String) Gets a JAR entry from the JAR file of the service component.
public String	getLocalProperty(String) Searches for the property with the specified key in local property list.
public String	getLocalProperty(String, String) Searches for the property with the specified key in the local property list.
public String	getMobileProperty(String) Searches for the property with the specified key in the mobile property list.
public String	getMobileProperty(String, String) Searches for the property with the specified key in the mobile property list.
public ServiceProxy	getServiceProxy(String) Acquire a proxy to a service.
public int	getState() Returns the state of the service.
public String	getStationaryProperty(String) Searches for the property with the specified key in the stationary property list.

Member Summary

public String	`getStationaryProperty(String, String)` Searches for the property with the specified key in the stationary property list.
public void	`loadServiceComponent(InputStream)` Loads and adds a saved service component from an input stream.
public void	`registerService(String, ServiceInterfaceFactory)` Registers a service.
public void	`reload()` Resets the service environment to the last saved state.
public void	`remove()` Schedules the service component for removal.
public void	`removeServiceComponent(String)` Removes a service component from the service environment.
public void	`removeServiceComponentListener(String, ServiceComponentListener)` Removes a listener to service component events of a specified service component.
public void	`removeServiceContextListener(ServiceContextListener)` Removes a listener to service context events.
public void	`resumeServiceComponent(String, ServiceContextEvent)` Resumes a service component.
public void	`save()` Saves the state of the service environment in a service briefcase.
public void	`setMobileProperty(String, String)` Sets the property with the specified key in the mobile property list.
public void	`setState(int)` Sets the state of the service.
public void	`setStationaryProperty(String, String)` Sets the property with the specified key in the stationary property list.
public void	`shutdown()` Performs a shutdown of the service environment without saving.
public void	`stop()` Schedules the service component for termination.
public void	`stopServiceComponent(String, ServiceContextEvent)` Stops a service component.
public void	`suspend()` Schedules the service component for suspension.
public void	`suspendServiceComponent(String, ServiceContextEvent)` Suspends a service component.
public void	`synchronize()` Synchronizes the service briefcase with the default service briefcase servers.
public void	`unregisterService(String)` Unregisters a service.

Inherited Member Summary

Fields inherited from interface `Constants`

`ACTIVE`, `INACTIVE`, `INITIALIZED`, `INITIALIZING`, `JAR_ACTIVATOR`, `JAR_CLASSPATH`, `JAR_DEPEND`, `JAR_EXPORT`, `JAR_IMPORT`, `JAR_NAME`, `JAR_PERMISSION`, `RESUMED`, `RESUMING`, `SP_HOSTS`, `STARTING`, `STOPPED`, `STOPPING`, `SUSPENDED`, `SUSPENDING`, `stateNames`

Methods

addServiceComponentListener(String, ServiceComponentListener)

```
public void addServiceComponentListener(java.lang.String key,
            ServiceComponentListener listener)
            throws ServiceContextException
```

Adds a listener to service component events of a specified service component.

Parameters:
`key` - the key of the service component to listen to

`listener` - the listener to add

Throws:
`ServiceContextException` - if the service component specified does not exist

addServiceContextListener(ServiceContextListener)

```
public void addServiceContextListener(ServiceContextListener listener)
```

Adds a listener to service context events.

Parameters:
`listener` - the listener to add

createServiceComponent(String)

```
public void createServiceComponent(java.lang.String jarName)
            throws ServiceContextException
```

Creates and adds a service component based on a JAR file containing a specification of a service component.

Parameters:
`jarName` - a URL to the JAR file

Throws:
`ServiceContextException` - if the service could not be created

getJarAttribute(String)

```
public java.lang.String getJarAttribute(java.lang.String name)
```

Gets an attribute from the JAR file of the service component.

Parameters:
`name` - the name of the attribute

Returns: the JAR attribute with the specified name

getJarEntry(String)

```
public byte[] getJarEntry(java.lang.String name)
```

Gets a JAR entry from the JAR file of the service component.

Parameters:
`name` - the name JAR entry name

Returns: a JAR entry with the specified entry name

getLocalProperty(String)

`public java.lang.String` **`getLocalProperty`**`(java.lang.String key)`

Searches for the property with the specified key in local property list. The method returns `null` if the property is not found.

Parameters:
`key` - the property key

Returns: the value in the local property list with the specified key value

getLocalProperty(String, String)

`public java.lang.String` **`getLocalProperty`**`(java.lang.String key, java.lang.String def)`

Searches for the property with the specified key in the local property list. The method returns the default value argument if the property is not found.

Parameters:
`key` - the property key

`def` - the default value

Returns: the value in the local property list with the specified key value

getMobileProperty(String)

`public java.lang.String` **`getMobileProperty`**`(java.lang.String key)`

Searches for the property with the specified key in the mobile property list. The method returns `null` if the property is not found.

Parameters:
`key` - the property key

Returns: the value in the mobile property list with the specified key value

getMobileProperty(String, String)

`public java.lang.String` **`getMobileProperty`**`(java.lang.String key, java.lang.String def)`

Searches for the property with the specified key in the mobile property list. The method returns the default value argument if the property is not found.

Parameters:
`key` - the property key

`def` - the default value

Returns: the value in the mobile property list with the specified key value

getServiceProxy(String)

`public ServiceProxy` **`getServiceProxy`**`(java.lang.String name)`

Acquire a proxy to a service.

Parameters:
`name` - the registered name of the service

Returns: a proxy to the service

getState()

```
public int getState()
```

Returns the state of the service.

Returns: the state of the service

getStationaryProperty(String)

```
public java.lang.String getStationaryProperty(java.lang.String key)
```

Searches for the property with the specified key in the stationary property list. The method returns `null` if the property is not found.

Parameters:
`key` - the property key

Returns: the value in the stationary property list with the specified key value

getStationaryProperty(String, String)

```
public java.lang.String getStationaryProperty(java.lang.String key,
            java.lang.String def)
```

Searches for the property with the specified key in the stationary property list. The method returns the default value argument if the property is not found.

Parameters:
`key` - the property key

`def` - the default value

Returns: the value in the stationary property list with the specified key value

loadServiceComponent(InputStream)

```
public void loadServiceComponent(java.io.InputStream is)
            throws ServiceContextException
```

Loads and adds a saved service component from an input stream.

Parameters:
`is` - the input stream from which the service should be loaded

Throws:
`ServiceContextException` - if the service could not be loaded

registerService(String, ServiceInterfaceFactory)

```
public void registerService(java.lang.String name,
            ServiceInterfaceFactory interfaceFactory)
```

Registers a service.

Parameters:
`name` - the name of the service to register

`factory` - a factory for creating interfaces to the service when acquired by another service component

reload()

```
public void reload()
            throws ServiceContextException
```

Resets the service environment to the last saved state. This method will cause the service environment to shutdown temorarily. Unsaved data and modifications will be lost.

Throws:
`ServiceContextException` - if reload failed

remove()

```
public void remove()
```

Schedules the service component for removal.

removeServiceComponent(String)

```
public void removeServiceComponent(java.lang.String key)
            throws ServiceContextException
```

Removes a service component from the service environment.

Parameters:
`key` - the key of service component to remove

Throws:
`ServiceContextException` - if the service component specified does not exist

removeServiceComponentListener(String, ServiceComponentListener)

```
public void removeServiceComponentListener(java.lang.String key,
            ServiceComponentListener listener)
            throws ServiceContextException
```

Removes a listener to service component events of a specified service component.

Parameters:
`key` - the key of the service component from which the listener should be removed

`listener` - the listener to remove

Throws:
`ServiceContextException` - if the service component specified does not exist

removeServiceContextListener(ServiceContextListener)

```
public void removeServiceContextListener(ServiceContextListener listener)
```

Removes a listener to service context events.

Parameters:
`listener` - the listener to remove

resumeServiceComponent(String, ServiceContextEvent)

```
public void resumeServiceComponent(java.lang.String key, ServiceContextEvent evt)
            throws ServiceContextException
```

Resumes a service component.

Parameters:
`key` - the key of service to resume

`evt` - the event containing the reason for the resumption

Throws:
`ServiceContextException` - if the service component specified does not exist

save()

```
public void save()
            throws ServiceContextException
```

Saves the state of the service environment in a service briefcase. This method will cause the service environment to shutdown temporarily.

Throws:
`ServiceContextException` - if save failed

setMobileProperty(String, String)

```
public void setMobileProperty(java.lang.String key, java.lang.String value)
```

Sets the property with the specified key in the mobile property list.

Parameters:
`key` - the property key

`value` - theh value of the property

setState(int)

```
public void setState(int state)
```

Sets the state of the service. This method is only effective if the service is currently engaged in a state change (i.e. the ServiceContext has called one of the state modifying methods, to which the service has returned a negative value to indicate that the state modification is ongoing).

Parameters:
`state` - the new state of the service

setStationaryProperty(String, String)

```
public void setStationaryProperty(java.lang.String key, java.lang.String value)
```

Sets the property with the specified key in the stationary property list.

Parameters:
`key` - the property key

`value` - theh value of the property

shutdown()

```
public void shutdown()
            throws ServiceContextException
```

Performs a shutdown of the service environment without saving. Unsaved data and modifications will be lost.

stop()

Throws:
ServiceContextException - if shutdown failed

stop()

```
public void stop()
```

Schedules the service component for termination.

stopServiceComponent(String, ServiceContextEvent)

```
public void stopServiceComponent(java.lang.String key, ServiceContextEvent evt)
            throws ServiceContextException
```

Stops a service component.

Parameters:
key - the key of service to stop.

evt - the event containing the reason for the stop

Throws:
ServiceContextException - if the service component specified does not exist

suspend()

```
public void suspend()
```

Schedules the service component for suspension.

suspendServiceComponent(String, ServiceContextEvent)

```
public void suspendServiceComponent(java.lang.String key, ServiceContextEvent evt)
            throws ServiceContextException
```

Suspends a service component.

Parameters:
key - the key of service to suspend

evt - the event containing the reason for the suspenstion

Throws:
ServiceContextException - if the service component specified does not exist

synchronize()

```
public void synchronize()
            throws ServiceContextException
```

Synchronizes the service briefcase with the default service briefcase servers. This method will cause the service environment to shutdown temporarily.

Throws:
ServiceContextException - if synchronize failed

unregisterService(String)

```
public void unregisterService(java.lang.String name)
```

Unregisters a service.

Parameters:

`name` - the name of the service to unregister

se.sics.sview.core
ServiceContextEvent

Declaration

```
public class ServiceContextEvent

java.lang.Object
  |
  +--se.sics.sview.core.ServiceContextEvent
```

Direct Known Subclasses: se.sics.sview.core.event.StartEvent, se.sics.sview.core.event.StopEvent, se.sics.sview.core.event.SuspendEvent

Description

Superclass of all service context events. See the classes in package se.sics.sview.core.event for a full listing of predefined service context events.

Member Summary	
Constructors	
	public ServiceContextEvent()

Inherited Member Summary
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

ServiceContextEvent()

```
public ServiceContextEvent()
```

se.sics.sview.core
ServiceContextException

Declaration

```
public class ServiceContextException extends java.lang.Exception

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--se.sics.sview.core.ServiceContextException
```

All Implemented Interfaces: java.io.Serializable

Description

This exception is thrown from the service context whenever a request experiences fatal errors.

Member Summary

Fields

public	`detail` Nested Exception to hold wrapped exception.

Constructors

public	`ServiceContextException()` Constructs a `ServiceContextException` with no specified detail message.
public	`ServiceContextException(String)` Constructs a `ServiceContextException` with the specified detail message.
public	`ServiceContextException(String, Throwable)` Constructs a `ServiceContextException` with the specified detail message and nested exception.

Methods

public String	`getMessage()` Returns the detail message, including the message from the nested exception if there is one.
public void	`printStackTrace()` Prints the composite message to `System.err`.
public void	`printStackTrace(PrintStream)` Prints the composite message and the embedded stack trace to the specified stream `ps`.
public void	`printStackTrace(PrintWriter)` Prints the composite message and the embedded stack trace to the specified print writer `pw`

detail

Inherited Member Summary
Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`
Methods inherited from class java.lang.Throwable
`fillInStackTrace, getLocalizedMessage, toString`

Fields

detail

```
public java.lang.Throwable detail
```

Nested Exception to hold wrapped exception.

Constructors

ServiceContextException()

```
public ServiceContextException()
```

Constructs a `ServiceContextException` with no specified detail message.

ServiceContextException(String)

```
public ServiceContextException(java.lang.String s)
```

Constructs a `ServiceContextException` with the specified detail message.

Parameters:

`s` - the detail message

ServiceContextException(String, Throwable)

```
public ServiceContextException(java.lang.String s, java.lang.Throwable ex)
```

Constructs a `ServiceContextException` with the specified detail message and nested exception.

Parameters:

`s` - the detail message

`ex` - the nested exception

Methods

getMessage()

```
public java.lang.String getMessage()
```

Returns the detail message, including the message from the nested exception if there is one.

Overrides: java.lang.Throwable.getMessage() in class java.lang.Throwable

printStackTrace()

```
public void printStackTrace()
```

Prints the composite message to `System.err`.

Overrides: java.lang.Throwable.printStackTrace() in class java.lang.Throwable

printStackTrace(PrintStream)

```
public void printStackTrace(java.io.PrintStream ps)
```

Prints the composite message and the embedded stack trace to the specified stream `ps`.

Overrides: java.lang.Throwable.printStackTrace(java.io.PrintStream) in class java.lang.Throwable

Parameters:
`ps` - the print stream

printStackTrace(PrintWriter)

```
public void printStackTrace(java.io.PrintWriter pw)
```

Prints the composite message and the embedded stack trace to the specified print writer `pw`

Overrides: java.lang.Throwable.printStackTrace(java.io.PrintWriter) in class java.lang.Throwable

Parameters:
`pw` - the print writer

se.sics.sview.core
ServiceContextListener

Declaration
`public interface` **`ServiceContextListener`**

Description
A listener to events from the service context. See also
`ServiceContext.addServiceContextListener(ServiceContextListener)` .

Member Summary	
Methods	
`public void`	`serviceComponentAdded(ServiceComponentEvent)` Is invoked whenever a service component is added to the service environment.
`public void`	`serviceComponentRemoved(ServiceComponentEvent)` Is invoked whenever a service component is removed from the service environment.

Methods

serviceComponentAdded(ServiceComponentEvent)

`public void` **`serviceComponentAdded`**`(ServiceComponentEvent evt)`

Is invoked whenever a service component is added to the service environment.

Parameters:
`evt` - the first `ServiceComponentEvent` that is created by the newly added service component

serviceComponentRemoved(ServiceComponentEvent)

`public void` **`serviceComponentRemoved`**`(ServiceComponentEvent evt)`

Is invoked whenever a service component is removed from the service environment.

Parameters:
`evt` - the last `ServiceComponentEvent` that was created by the service component before it was removed

se.sics.sview.core
ServiceInterfaceFactory

Declaration
```
public interface ServiceInterfaceFactory
```

Description
Service components that wish to register services for other service components to use must come with an implementation of this interface. An instantiation of the class should be sent to the service context during service registration, and is used to create interfaces to the service when other service components request subscriptions.

Member Summary	
Methods	
public Object	createServiceInterface(String) Invoked to create a service interface to a service of a service component.

Methods

createServiceInterface(String)

```
public java.lang.Object createServiceInterface(java.lang.String name)
```

Invoked to create a service interface to a service of a service component.

Parameters:
`name` - the key of the service component to create an interface for

Returns: an interface to the service

se.sics.sview.core
ServiceListener

Declaration
`public interface` **`ServiceListener`**

Description
The listener to service events. See also
`ServiceProxy.addServiceListener(ServiceListener)` .

Member Summary	
Methods	
`public void`	`serviceRegistered(String)` Invoked when the service registers.
`public void`	`serviceUnregistered(String)` Invoked when the service unregisters.

Methods

serviceRegistered(String)

`public void` **`serviceRegistered`**`(java.lang.String name)`

Invoked when the service registers.

Parameters:
`name` - the name of the service

serviceUnregistered(String)

`public void` **`serviceUnregistered`**`(java.lang.String name)`

Invoked when the service unregisters.

Parameters:
`name` - the name of the service

se.sics.sview.core
ServiceProxy

Declaration
```
public interface ServiceProxy
```

Description
A service component that wish to subscribe to a service requests a service proxy to the service from its service context. Via the service proxy, the service component can (un)subscribe to the service, and register for notifications of when the service (un)registers (a service need not be registered in order for a service component to acquire a service proxy to it).

Member Summary		
Methods		
	`public void`	`addServiceListener(ServiceListener)` Adds a service listener to this proxy.
	`public void`	`removeServiceListener(ServiceListener)` Removes a service listener from this service proxy.
	`public Object`	`subscribe()` Subscribe to the service represented by this proxy.
	`public Object`	`subscribe(long)` Subscribe to the service represented by this proxy.
	`public void`	`unsubscribe()` Unsubscribe to the service represented by this proxy.

Methods

addServiceListener(ServiceListener)

```
public void addServiceListener(ServiceListener listener)
```

Adds a service listener to this proxy. The listener will be notified when the service represented by this proxy (un)registers.

Parameters:
`listener` - the object to notify when the service (un)registers

removeServiceListener(ServiceListener)

```
public void removeServiceListener(ServiceListener listener)
```

Removes a service listener from this service proxy.

Parameters:
`listener` - the listener object to remove

subscribe()

```
public java.lang.Object subscribe()
```

Subscribe to the service represented by this proxy.

Returns: an interface to the service, `null` if the service is not registered

subscribe(long)

```
public java.lang.Object subscribe(long timeout)
            throws InterruptedException
```

Subscribe to the service represented by this proxy. If the service is not registered yet, wait `timeout` milliseconds for it to registered. If the service is not registered within that time, return `null`.

Parameters:
`number` - of milliseconds to wait for the service to register

Returns: an interface to the service, `null` if the service is not registered

Throws:
`InterruptedException` - if interrupted while waiting

unsubscribe()

```
public void unsubscribe()
```

Unsubscribe to the service represented by this proxy.

se.sics.sview.core
TransactionCoordinator

Declaration
`public interface` **`TransactionCoordinator`**

Description
A transaction wraps the steps in service briefcase synchronization in order to make it atomic, and to provide exception handling.

A transaction coordinator shold implement this interface. The coordinator of a transaction can be, but need not be, the initiator if the synchronization.

Member Summary	
Methods	
`public void`	`abort()` Aborts the transaction.
`public void`	`acknowledge(TransactionParticipant)` The participants of a transaction calls this method in order to acknowledge that the transaction has completed successfully.

Methods

abort()
`public void` **`abort`**`()`

Aborts the transaction.

acknowledge(TransactionParticipant)
`public void` **`acknowledge`**`(TransactionParticipant tp)`

The participants of a transaction calls this method in order to acknowledge that the transaction has completed successfully.

Parameters:
`tp` - the transaction participant that acknowledges

se.sics.sview.core
TransactionInitiator

Declaration
`public interface` **`TransactionInitiator`**

Description
A transaction wraps the steps in service briefcase synchronization in order to make it atomic, and to provide exception handling.

A transaction initiator shold implement this interface.

Member Summary	
Methods	
`public void`	`globalAcknowledge()` The coordinator of the transaction calls this method when all participants have acknowledged that the transaction has completed successfully.
`public void`	`globalCommit(TransactionParticipant[])` The coordinator of the transaction calls this method when all participants have voted for participation.

Methods

globalAcknowledge()

`public void` **`globalAcknowledge`**`()`

The coordinator of the transaction calls this method when all participants have acknowledged that the transaction has completed successfully.

globalCommit(TransactionParticipant[])

`public void` **`globalCommit`**`(TransactionParticipant[] tps)`

The coordinator of the transaction calls this method when all participants have voted for participation. The participants that voted in favor of the transaction are representd in the given array of participants.

Parameters:

`tps` - an array with the transaction participants that has voted in favor of the transaction

se.sics.sview.core
TransactionParticipant

Declaration
```
public interface TransactionParticipant
```

Description
A transaction wraps the steps in service briefcase synchronization in order to make it atomic, and to provide exception handling.

A transaction participant shold implement this interface.

Member Summary	
Methods	
public void	globalCommit(TransactionCoordinator) Called by the coordinator to signal that the transaction is ready to start.
public boolean	vote() Called by the coordinator to vote for participation in a transaction.

Methods

globalCommit(TransactionCoordinator)
```
public void globalCommit(TransactionCoordinator tc)
            throws Exception
```
Called by the coordinator to signal that the transaction is ready to start.

Parameters:
`tc` - a reference to transaction coordinator

Throws:
`Exception`

vote()
```
public boolean vote()
            throws Exception
```
Called by the coordinator to vote for participation in a transaction.

Returns: commit `true` or abort `false`

Throws:
`Exception`

TransactionParticipant se.sics.sview.core

vote()

Package
se.sics.sview.core.event

Class Summary	
Classes	
CreateEvent	This event is used to start service components for the first time, immediately after it has been created.
LoadEvent	This event is used to start a service enevironment that has been loaded from persistent media.
MoveEvent	This event is used to suspend service components before the service environment is moved (or synchronized) to another server.
ReloadEvent	This event is used to stop the service components in a service environment before reloading the environment.
RemoveEvent	This stop event is used to stop a service component before it is removed from the service environment (and the service briefcase).
ResetEvent	This stop event is used to stop service components before reseting the server.
SaveEvent	This event is used to suspend service components before saving the service briefcase.
ShutdownEvent	This event is used to stop service components before shutting down the service environment.
StartEvent	This is the super class of all start events.
StopEvent	This is the super class of all stop events.
SuspendEvent	This is the super class of all suspend events.
UpdateEvent	This event is used to stop a service component that is about to get updated.

se.sics.sview.core.event
CreateEvent

Declaration

```
public class CreateEvent extends se.sics.sview.core.event.StartEvent

java.lang.Object
  |
  +--se.sics.sview.core.se.sics.sview.core.ServiceContextEvent
        |
        +--se.sics.sview.core.event.se.sics.sview.core.event.StartEvent
              |
              +--se.sics.sview.core.event.CreateEvent
```

Description

This event is used to start service components for the first time, immediately after it has been created.

Member Summary
Constructors
public CreateEvent()

Inherited Member Summary
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

CreateEvent()

```
public CreateEvent()
```

se.sics.sview.core.event
LoadEvent

Declaration

```
public class LoadEvent extends se.sics.sview.core.event.StartEvent

java.lang.Object
  |
  +--se.sics.sview.core.se.sics.sview.core.ServiceContextEvent
        |
        +--se.sics.sview.core.event.se.sics.sview.core.event.StartEvent
              |
              +--se.sics.sview.core.event.LoadEvent
```

Description

This event is used to start a service enevironment that has been loaded from persistent media.

Member Summary
Constructors
public LoadEvent()

Inherited Member Summary
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

LoadEvent()

```
public LoadEvent()
```

se.sics.sview.core.event
MoveEvent

Declaration

```
public class MoveEvent extends se.sics.sview.core.event.SuspendEvent

java.lang.Object
  |
  +--se.sics.sview.core.se.sics.sview.core.ServiceContextEvent
        |
        +--se.sics.sview.core.event.se.sics.sview.core.event.SuspendEvent
              |
              +--se.sics.sview.core.event.MoveEvent
```

Description

This event is used to suspend service components before the service environment is moved (or synchronized) to another server.

Member Summary
Constructors
public MoveEvent()

Inherited Member Summary
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

MoveEvent()

```
public MoveEvent()
```

se.sics.sview.core.event
ReloadEvent

Declaration

```
public class ReloadEvent extends se.sics.sview.core.event.StopEvent

java.lang.Object
  |
  +--se.sics.sview.core.se.sics.sview.core.ServiceContextEvent
        |
        +--se.sics.sview.core.event.se.sics.sview.core.event.StopEvent
              |
              +--se.sics.sview.core.event.ReloadEvent
```

Description

This event is used to stop the service components in a service environment before reloading the environment.

Member Summary
Constructors
public ReloadEvent()

Inherited Member Summary
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

ReloadEvent()

```
public ReloadEvent()
```

RemoveEvent()

se.sics.sview.core.event
RemoveEvent

Declaration

```
public class RemoveEvent extends se.sics.sview.core.event.StopEvent

java.lang.Object
  |
  +--se.sics.sview.core.se.sics.sview.core.ServiceContextEvent
        |
        +--se.sics.sview.core.event.se.sics.sview.core.event.StopEvent
              |
              +--se.sics.sview.core.event.RemoveEvent
```

Description

This stop event is used to stop a service component before it is removed from the service environment (and the service briefcase).

Member Summary	
Constructors	
	public RemoveEvent()

Inherited Member Summary
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

RemoveEvent()

```
public RemoveEvent()
```

se.sics.sview.core.event
ResetEvent

Declaration

```
public class ResetEvent extends se.sics.sview.core.event.StopEvent

java.lang.Object
  |
  +--se.sics.sview.core.se.sics.sview.core.ServiceContextEvent
        |
        +--se.sics.sview.core.event.se.sics.sview.core.event.StopEvent
              |
              +--se.sics.sview.core.event.ResetEvent
```

Description

This stop event is used to stop service components before reseting the server.

Member Summary
Constructors
public ResetEvent()

Inherited Member Summary
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

ResetEvent()

```
public ResetEvent()
```

se.sics.sview.core.event
SaveEvent

Declaration

```
public class SaveEvent extends se.sics.sview.core.event.SuspendEvent

java.lang.Object
  |
  +--se.sics.sview.core.se.sics.sview.core.ServiceContextEvent
        |
        +--se.sics.sview.core.event.se.sics.sview.core.event.SuspendEvent
              |
              +--se.sics.sview.core.event.SaveEvent
```

Description

This event is used to suspend service components before saving the service briefcase.

Member Summary	
Constructors	
	public SaveEvent()

Inherited Member Summary
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

SaveEvent()

```
public SaveEvent()
```

se.sics.sview.core.event
ShutdownEvent

Declaration

```
public class ShutdownEvent extends se.sics.sview.core.event.StopEvent

java.lang.Object
  |
  +--se.sics.sview.core.se.sics.sview.core.ServiceContextEvent
        |
        +--se.sics.sview.core.event.se.sics.sview.core.event.StopEvent
              |
              +--se.sics.sview.core.event.ShutdownEvent
```

Description

This event is used to stop service components before shutting down the service environment.

Member Summary	
Constructors	
	public ShutdownEvent()

Inherited Member Summary
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

ShutdownEvent()

```
public ShutdownEvent()
```

se.sics.sview.core.event
StartEvent

Declaration

```
public class StartEvent extends se.sics.sview.core.ServiceContextEvent

java.lang.Object
  |
  +--se.sics.sview.core.se.sics.sview.core.ServiceContextEvent
        |
        +--se.sics.sview.core.event.StartEvent
```

Direct Known Subclasses: CreateEvent, LoadEvent

Description

This is the super class of all start events.

Member Summary
Constructors
public StartEvent()

Inherited Member Summary
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

StartEvent()

```
public StartEvent()
```

se.sics.sview.core.event
StopEvent

Declaration

```
public class StopEvent extends se.sics.sview.core.ServiceContextEvent

java.lang.Object
  |
  +--se.sics.sview.core.se.sics.sview.core.ServiceContextEvent
        |
        +--se.sics.sview.core.event.StopEvent
```

Direct Known Subclasses: ReloadEvent, RemoveEvent, ResetEvent, ShutdownEvent, UpdateEvent

Description

This is the super class of all stop events.

Member Summary
Constructors
public StopEvent()

Inherited Member Summary
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

StopEvent()

```
public StopEvent()
```

SuspendEvent()

se.sics.sview.core.event
SuspendEvent

Declaration

```
public class SuspendEvent extends se.sics.sview.core.ServiceContextEvent

java.lang.Object
  |
  +--se.sics.sview.core.se.sics.sview.core.ServiceContextEvent
        |
        +--se.sics.sview.core.event.SuspendEvent
```

Direct Known Subclasses: MoveEvent, SaveEvent

Description

This is the super class of all suspend events.

Member Summary
Constructors
public SuspendEvent()

Inherited Member Summary
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

SuspendEvent()

```
public SuspendEvent()
```

se.sics.sview.core.event
UpdateEvent

Declaration

```
public class UpdateEvent extends se.sics.sview.core.event.StopEvent

java.lang.Object
  |
  +--se.sics.sview.core.se.sics.sview.core.ServiceContextEvent
        |
        +--se.sics.sview.core.event.se.sics.sview.core.event.StopEvent
              |
              +--se.sics.sview.core.event.UpdateEvent
```

Description

This event is used to stop a service component that is about to get updated.

Member Summary
Constructors
public UpdateEvent()

Inherited Member Summary
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

UpdateEvent()

```
public UpdateEvent()
```

UpdateEvent se.sics.sview.core.event

UpdateEvent()

Package
se.sics.sview.core.permission

Class Summary	
Interfaces	
AllPermissions	Allows handling of all protected actions.
ComponentHandling	Allows creation, loading, and removal of service components.
CreateComponent	Allows creation service components.
LoadComponent	Allows loading of service components.
OtherPermissionHandling	Allows handling of other service components' permissions.
OwnPermissionHandling	Allows handling of own service component permissions.
PermissionHandling	Allows handling of service component permissions.
ReloadEnvironment	Allows reloading of the service environment.
RemoveComponent	Allows removal of service components.
ResumeComponent	Allows resumption of service components.
RuntimeHandling	Allows managing (suspend, resume, and stop) of service components.
SaveComponent	Allows saving of service components.
SaveEnvironment	Allows saving of the service environment.
ServiceComponentHandling	Allows creation and removal of service components, as well as managing (suspend, resume, and stop) of allready existing service components.
ServiceEnvironmentHandling	Allows handling of the service environment (reload, synchronize, save, and shutdown).
ShutdownEnvironment	Allows shutdown of the service environment.
StopComponent	Allows stopping of service components.
SuspendComponent	Allows suspension of service components.
SynchronizeEnvironment	Allows synchronization of the service environment.

se.sics.sview.core.permission
AllPermissions

Declaration
`public interface` **`AllPermissions extends`** **`se.sics.sview.core.ServiceComponentPermission`**

All Superinterfaces: `se.sics.sview.core.ServiceComponentPermission`

All Known Subinterfaces: `ComponentHandling`, `CreateComponent`, `LoadComponent`, `OtherPermissionHandling`, `OwnPermissionHandling`, `PermissionHandling`, `ReloadEnvironment`, `RemoveComponent`, `ResumeComponent`, `RuntimeHandling`, `SaveComponent`, `SaveEnvironment`, `ServiceComponentHandling`, `ServiceEnvironmentHandling`, `ShutdownEnvironment`, `StopComponent`, `SuspendComponent`, `SynchronizeEnvironment`

Description
Allows handling of all protected actions.

Member Summary	
Fields	
`public static final`	`description` A textual description of the permission.

Fields

description
`public static final java.lang.String` **`description`**

A textual description of the permission. Override this field to describe what the permission grants access to.

se.sics.sview.core.permission
ComponentHandling

Declaration

```
public interface ComponentHandling extends
    se.sics.sview.core.permission.ServiceComponentHandling
```

All Superinterfaces: AllPermissions, ServiceComponentHandling, se.sics.sview.core.ServiceComponentPermission

All Known Subinterfaces: CreateComponent, LoadComponent, RemoveComponent, SaveComponent

Description

Allows creation, loading, and removal of service components.

Member Summary
Fields
public static final description

Fields

description

```
public static final java.lang.String description
```

se.sics.sview.core.permission
CreateComponent

Declaration
`public interface` **`CreateComponent extends`** **`se.sics.sview.core.permission.ComponentHandling`**

All Superinterfaces: `AllPermissions`, `ComponentHandling`, `ServiceComponentHandling`, `se.sics.sview.core.ServiceComponentPermission`

Description
Allows creation service components.

Member Summary
Fields
`public static final` `description`

Fields

description
`public static final java.lang.String` **`description`**

se.sics.sview.core.permission
LoadComponent

Declaration
```
public interface LoadComponent extends se.sics.sview.core.permission.ComponentHandling
```

All Superinterfaces: AllPermissions, ComponentHandling, ServiceComponentHandling, se.sics.sview.core.ServiceComponentPermission

Description
Allows loading of service components.

Member Summary
Fields
public static final description

Fields

description
```
public static final java.lang.String description
```

se.sics.sview.core.permission
OtherPermissionHandling

Declaration

```
public interface OtherPermissionHandling extends
    se.sics.sview.core.permission.PermissionHandling
```

All Superinterfaces: `AllPermissions`, `PermissionHandling`, `se.sics.sview.core.ServiceComponentPermission`

Description

Allows handling of other service components' permissions.

Member Summary	
Fields	
`public static final`	`description` A textual description of the permission.

Fields

description

```
public static final java.lang.String description
```

A textual description of the permission. Override this field to describe what the permission grants access to.

se.sics.sview.core.permission
OwnPermissionHandling

Declaration

```
public interface OwnPermissionHandling extends
    se.sics.sview.core.permission.PermissionHandling
```

All Superinterfaces: AllPermissions, PermissionHandling, se.sics.sview.core.ServiceComponentPermission

Description

Allows handling of own service component permissions.

Member Summary	
Fields	
public static final	description A textual description of the permission.

Fields

description

```
public static final java.lang.String description
```

A textual description of the permission. Override this field to describe what the permission grants access to.

se.sics.sview.core.permission
PermissionHandling

Declaration
`public interface` **`PermissionHandling extends`** **`se.sics.sview.core.permission.AllPermissions`**

All Superinterfaces: `AllPermissions`, `se.sics.sview.core.ServiceComponentPermission`

All Known Subinterfaces: `OtherPermissionHandling`, `OwnPermissionHandling`

Description
Allows handling of service component permissions.

Member Summary	
Fields	
`public static final`	`description` A textual description of the permission.

Fields

description
`public static final java.lang.String` **`description`**

A textual description of the permission. Override this field to describe what the permission grants access to.

se.sics.sview.core.permission
ReloadEnvironment

Declaration

```
public interface ReloadEnvironment extends
    se.sics.sview.core.permission.ServiceEnvironmentHandling
```

All Superinterfaces: AllPermissions, se.sics.sview.core.ServiceComponentPermission, ServiceEnvironmentHandling

Description

Allows reloading of the service environment.

Member Summary	
Fields	
public static final	description A textual description of the permission.

Fields

description

```
public static final java.lang.String description
```

A textual description of the permission. Override this field to describe what the permission grants access to.

se.sics.sview.core.permission
RemoveComponent

Declaration
`public interface` **`RemoveComponent extends`** **`se.sics.sview.core.permission.ComponentHandling`**

All Superinterfaces: `AllPermissions`, `ComponentHandling`, `ServiceComponentHandling`, `se.sics.sview.core.ServiceComponentPermission`

Description
Allows removal of service components.

Member Summary
Fields
`public static final` `description`

Fields

description
`public static final java.lang.String` **`description`**

se.sics.sview.core.permission
ResumeComponent

Declaration
`public interface` **`ResumeComponent extends`** **`se.sics.sview.core.permission.RuntimeHandling`**

All Superinterfaces: `AllPermissions`, `RuntimeHandling`, `ServiceComponentHandling`, `se.sics.sview.core.ServiceComponentPermission`

Description
Allows resumption of service components.

Member Summary
Fields
`public static final` `description`

Fields

description
`public static final java.lang.String` **`description`**

se.sics.sview.core.permission
RuntimeHandling

Declaration

```
public interface RuntimeHandling extends
    se.sics.sview.core.permission.ServiceComponentHandling
```

All Superinterfaces: `AllPermissions`, `ServiceComponentHandling`, `se.sics.sview.core.ServiceComponentPermission`

All Known Subinterfaces: `ResumeComponent`, `StopComponent`, `SuspendComponent`

Description

Allows managing (suspend, resume, and stop) of service components.

Member Summary
Fields
`public static final description`

Fields

description

```
public static final java.lang.String description
```

se.sics.sview.core.permission
SaveComponent

Declaration
`public interface` **`SaveComponent extends`** **`se.sics.sview.core.permission.ComponentHandling`**

All Superinterfaces: `AllPermissions`, `ComponentHandling`, `ServiceComponentHandling`, `se.sics.sview.core.ServiceComponentPermission`

Description
Allows saving of service components.

Member Summary
Fields
`public static final` `description`

Fields

description
`public static final java.lang.String` **`description`**

se.sics.sview.core.permission
SaveEnvironment

Declaration

```
public interface SaveEnvironment extends
    se.sics.sview.core.permission.ServiceEnvironmentHandling
```

All Superinterfaces: `AllPermissions`, `se.sics.sview.core.ServiceComponentPermission`, `ServiceEnvironmentHandling`

Description

Allows saving of the service environment.

Member Summary	
Fields	
`public static final`	`description` A textual description of the permission.

Fields

description

```
public static final java.lang.String description
```

A textual description of the permission. Override this field to describe what the permission grants access to.

se.sics.sview.core.permission
ServiceComponentHandling

Declaration

```
public interface ServiceComponentHandling extends
    se.sics.sview.core.permission.AllPermissions
```

All Superinterfaces: AllPermissions, se.sics.sview.core.ServiceComponentPermission

All Known Subinterfaces: ComponentHandling, CreateComponent, LoadComponent, RemoveComponent, ResumeComponent, RuntimeHandling, SaveComponent, StopComponent, SuspendComponent

Description

Allows creation and removal of service components, as well as managing (suspend, resume, and stop) of allready existing service components.

Member Summary
Fields
public static final description

Fields

description

```
public static final java.lang.String description
```

se.sics.sview.core.permission
ServiceEnvironmentHandling

Declaration

```
public interface ServiceEnvironmentHandling extends
    se.sics.sview.core.permission.AllPermissions
```

All Superinterfaces: `AllPermissions`, `se.sics.sview.core.ServiceComponentPermission`

All Known Subinterfaces: `ReloadEnvironment`, `SaveEnvironment`, `ShutdownEnvironment`, `SynchronizeEnvironment`

Description

Allows handling of the service environment (reload, synchronize, save, and shutdown).

Member Summary	
Fields	
`public static final`	`description` A textual description of the permission.

Fields

description

```
public static final java.lang.String description
```

A textual description of the permission. Override this field to describe what the permission grants access to.

se.sics.sview.core.permission
ShutdownEnvironment

Declaration

```
public interface ShutdownEnvironment extends
    se.sics.sview.core.permission.ServiceEnvironmentHandling
```

All Superinterfaces: `AllPermissions`, `se.sics.sview.core.ServiceComponentPermission`, `ServiceEnvironmentHandling`

Description

Allows shutdown of the service environment.

Member Summary	
Fields	
`public static final`	`description` A textual description of the permission.

Fields

description

```
public static final java.lang.String description
```

A textual description of the permission. Override this field to describe what the permission grants access to.

se.sics.sview.core.permission
StopComponent

Declaration
`public interface` **`StopComponent extends`** **`se.sics.sview.core.permission.RuntimeHandling`**

All Superinterfaces: `AllPermissions`, `RuntimeHandling`, `ServiceComponentHandling`, `se.sics.sview.core.ServiceComponentPermission`

Description
Allows stopping of service components.

Member Summary
Fields
`public static final` `description`

Fields

description
`public static final java.lang.String` **`description`**

se.sics.sview.core.permission

SuspendComponent

Declaration

```
public interface SuspendComponent extends se.sics.sview.core.permission.RuntimeHandling
```

All Superinterfaces: AllPermissions, RuntimeHandling, ServiceComponentHandling, se.sics.sview.core.ServiceComponentPermission

Description

Allows suspension of service components.

Member Summary
Fields
public static final description

Fields

description

```
public static final java.lang.String description
```

se.sics.sview.core.permission
SynchronizeEnvironment

Declaration

```
public interface SynchronizeEnvironment extends
    se.sics.sview.core.permission.ServiceEnvironmentHandling
```

All Superinterfaces: `AllPermissions`, `se.sics.sview.core.ServiceComponentPermission`, `ServiceEnvironmentHandling`

Description

Allows synchronization of the service environment.

Member Summary	
Fields	
`public static final`	`description` A textual description of the permission.

Fields

description

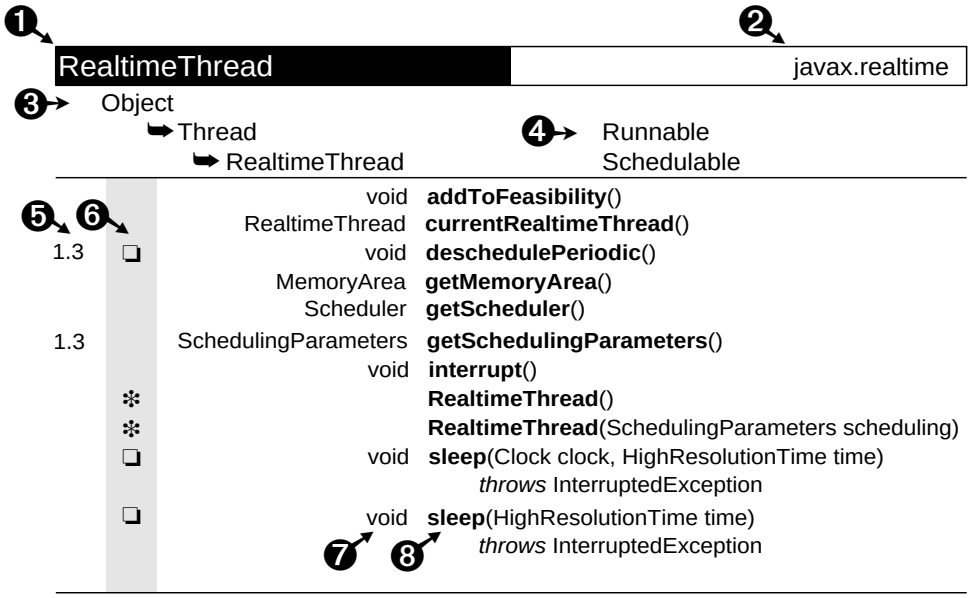
```
public static final java.lang.String description
```

A textual description of the permission. Override this field to describe what the permission grants access to.

ALMANAC LEGEND

The almanac presents classes and intefaces in alphabetic order, regardless of their package. Each class displays a list of its members in alphabetic order – fields, methods and constructors are sorted together.

This almanac is modeled after the style introduced by Patrick Chan in his excellent book *Java Developers Almanac*.



1. The name of the class, interface, nested class or nested interface. All interfaces are shown in italic.
2. The name of the package containing the class.
3. The inheritance chain of superclasses. In this example, `RealtimeThread` extends `Thread`, which extends `Object`.
4. Implemented interfaces. The class and the interface it implements are on the same line. In this example, `Thread` implements `Runnable`, and `RealtimeThread` implements `Schedulable`.
5. The first column is for the value of the `@since` comment, which indicates the version in which the item was introduced.
6. The second column is for the following icons that indicate modifiers, constructors and fields. If the "protected" symbol does not appear, the member is public. (Private and package-private modifiers have no symbols.)

Modifiers		**Constructors and Fields**	
❍	abstract	❇	constructor
●	final	✍	field
❑	static		
■	static final		
♦	protected		

7. The return type of a method or the declared type of a field. It is blank for constructors.
8. The name of the constructor, field or method. Sorted alphabetically. Nested classes are not listed as members.

Almanac

AllPermissions	**se.sics.sview.core.permission**
AllPermissions	se.sics.sview.core.ServiceComponentPermission
✍■	String **description**

Callback	**se.sics.sview.core**
Object	
➥Callback	Runnable
❇	**Callback(String type, Object[] args, CallbackListener listener)**
	void **run()**

CallbackListener	**se.sics.sview.core**
CallbackListener	
	void **callback(String type, Object[] args)**

ComponentHandling	**se.sics.sview.core.permission**
ComponentHandling	ServiceComponentHandling
✍■	String **description**

Constants	**se.sics.sview.core**
Constants	
✍■	int **ACTIVE**
✍■	int **INACTIVE**
✍■	int **INITIALIZED**
✍■	int **INITIALIZING**
✍■	String **JAR_ACTIVATOR**
✍■	String **JAR_CLASSPATH**
✍■	String **JAR_DEPEND**
✍■	String **JAR_EXPORT**
✍■	String **JAR_IMPORT**
✍■	String **JAR_NAME**
✍■	String **JAR_PERMISSION**
✍■	int **RESUMED**
✍■	int **RESUMING**
✍■	String **SP_HOSTS**

✍■	int	**STARTING**
✍■	String[]	**stateNames**
✍■	int	**STOPPED**
✍■	int	**STOPPING**
✍■	int	**SUSPENDED**
✍■	int	**SUSPENDING**

CreateComponent	**se.sics.sview.core.permission**

CreateComponent ComponentHandling

✍■	String	**description**

CreateEvent	**se.sics.sview.core.event**

Object
➥se.sics.sview.core.ServiceContextEvent
➥StartEvent
➥CreateEvent

❊	**CreateEvent()**

LoadComponent	**se.sics.sview.core.permission**

LoadComponent ComponentHandling

✍■	String	**description**

LoadEvent	**se.sics.sview.core.event**

Object
➥se.sics.sview.core.ServiceContextEvent
➥StartEvent
➥LoadEvent

❊	**LoadEvent()**

Mobile	**se.sics.sview.core**

Mobile

Monitor	**se.sics.sview.core**

Object
➥Monitor

	void	**arrogate(Object ref)** ***throws*** **InterruptedException**
	void	**enter()**
	void	**exit()**
❊		**Monitor()**
	void	**renounce()**

MoveEvent — se.sics.sview.core.event

Object
➥se.sics.sview.core.ServiceContextEvent
➥SuspendEvent
➥MoveEvent

❊		**MoveEvent()**

ObjectInputStreamLoader — se.sics.sview.core

Object
➥java.io.InputStream
➥java.io.ObjectInputStream — java.io.ObjectInput, java.io.ObjectStreamConstants
➥ObjectInputStreamLoader

	ClassLoader	**getClassLoader()**
❊		**ObjectInputStreamLoader(java.io.InputStream in, ClassLoader loader)** *throws* **java.io.IOException, java.io.StreamCorruptedException**
♦	Class	**resolveClass(java.io.ObjectStreamClass classDesc)** *throws* **java.io.IOException, ClassNotFoundException**

OtherPermissionHandling — se.sics.sview.core.permission

OtherPermissionHandling — PermissionHandling

✍■	String	**description**

OwnPermissionHandling — se.sics.sview.core.permission

OwnPermissionHandling — PermissionHandling

✍■	String	**description**

PermissionDeniedException — se.sics.sview.core

Object
➥Throwable — java.io.Serializable
➥Exception
➥RuntimeException
➥PermissionDeniedException

❊		**PermissionDeniedException()**
❊		**PermissionDeniedException(String s)**

PermissionHandling — se.sics.sview.core.permission

PermissionHandling — AllPermissions

✍■	String	**description**

Persistent — se.sics.sview.core

Persistent — java.io.Serializable

	boolean	**freeze()**
	void	**thaw()**

ReloadEnvironment	**se.sics.sview.core.permission**
ReloadEnvironment	ServiceEnvironmentHandling
✍■	String **description**

ReloadEvent	**se.sics.sview.core.event**
Object ➥se.sics.sview.core.ServiceContextEvent ➥StopEvent ➥ReloadEvent	
❇	**ReloadEvent()**

RemoveComponent	**se.sics.sview.core.permission**
RemoveComponent	ComponentHandling
✍■	String **description**

RemoveEvent	**se.sics.sview.core.event**
Object ➥se.sics.sview.core.ServiceContextEvent ➥StopEvent ➥RemoveEvent	
❇	**RemoveEvent()**

ResetEvent	**se.sics.sview.core.event**
Object ➥se.sics.sview.core.ServiceContextEvent ➥StopEvent ➥ResetEvent	
❇	**ResetEvent()**

ResumeComponent	**se.sics.sview.core.permission**
ResumeComponent	RuntimeHandling
✍■	String **description**

RuntimeHandling	**se.sics.sview.core.permission**
RuntimeHandling	ServiceComponentHandling
✍■	String **description**

SaveComponent	**se.sics.sview.core.permission**
SaveComponent	ComponentHandling
✍■	String **description**

SaveEnvironment — se.sics.sview.core.permission

SaveEnvironment — ServiceEnvironmentHandling

✍■	String	**description**

SaveEvent — se.sics.sview.core.event

Object
➥se.sics.sview.core.ServiceContextEvent
➥SuspendEvent
➥SaveEvent

❄		**SaveEvent()**

ServerProxy — se.sics.sview.core

ServerProxy

	String	**getProtocol()**
	ServiceBriefcaseServer	**getServiceBriefcaseServerProxy(String uri) *throws* Exception**
	TransactionParticipant	**getTransactionParticipantProxy(String uri, String id) *throws* Exception**
	void	**initialize(ServiceBriefcaseServer localServer, String[] args) *throws* Exception**

ServiceBriefcase — se.sics.sview.core

Object
➥ServiceBriefcase — java.io.Serializable

	void	**arrogateMonitor(Object ref) *throws* InterruptedException**
●	boolean	**authenticate(String uid, String pwd)**
●	void	**changePassword(String uid, String oldPwd, String newPwd)**
	void	**enterMonitor()**
	void	**exitMonitor()**
	java.util.Properties	**getMobileProperties(String uid, String pwd)**
	ServiceContainer[]	**getServiceComponents(String[] keys, String uid, String pwd)**
	ServiceContainer	**getServiceContainer(String key, String uid, String pwd)**
	String[]	**getServiceKeys(String uid, String pwd)**
	java.util.Properties[]	**getState(String uid, String pwd)**
	java.util.Properties	**getStationaryProperties(String uid, String pwd)**
❑	ServiceBriefcase	**load(java.io.InputStream is) *throws* java.io.IOException, ClassNotFoundException**
	void	**putServiceContainer(ServiceContainer service, String uid, String pwd)**
	void	**removeServiceContainer(String key, String uid, String pwd)**
	void	**renounceMonitor()**
❑	void	**save(ServiceBriefcase sb, java.io.OutputStream os) *throws* java.io.IOException**
❄		**ServiceBriefcase(java.util.Properties mobileProps, java.util.Properties stationaryProps, String uid, String pwd)**
❄		**ServiceBriefcase(String uid, String pwd)**
	void	**setMobileProperties(java.util.Properties props, String uid, String pwd)**
	void	**setMonitor(Monitor monitor)**

void	**setStationaryProperties(java.util.Properties props, String uid, String pwd)**
ServiceBriefcase	**toMobile(String uid, String pwd)**
void	**updateServiceBriefcase(ServiceContainer[] serviceContainers, java.util.Properties mobileProperties, String uid, String pwd)**

ServiceBriefcaseServer — se.sics.sview.core

ServiceBriefcaseServer

java.util.Properties	**getMobileProperties(String uid, String pwd, String transactionId) *throws* ServiceBriefcaseServerException**
String[]	**getRegisteredUsers() *throws* ServiceBriefcaseServerException**
ServiceBriefcase	**getServiceBriefcase(String uid, String pwd) *throws* ServiceBriefcaseServerException**
ServiceBriefcase	**getServiceBriefcase(String uid, String pwd, java.util.Date date) *throws* ServiceBriefcaseServerException**
java.util.Properties[]	**getServiceBriefcaseState(String uid, String pwd, String transactionId) *throws* ServiceBriefcaseServerException**
ServiceContainer[]	**getServiceComponents(String uid, String pwd, String[] keys, String transactionId) *throws* ServiceBriefcaseServerException**
void	**newServiceBriefcase(String uid, String pwd) *throws* ServiceBriefcaseServerException**
void	**removeServiceBriefcase(String uid, String pwd) *throws* ServiceBriefcaseServerException**
void	**startPse(String uid, String pwd) *throws* ServiceBriefcaseServerException**
void	**stopPse(String uid, String pwd) *throws* ServiceBriefcaseServerException**
void	**updateServiceBriefcase(String uid, String pwd, ServiceContainer[] serviceComponents, java.util.Properties mobileProperties, String transactionId) *throws* ServiceBriefcaseServerException**

ServiceBriefcaseServerException — se.sics.sview.core

Object
➥Throwable — java.io.Serializable
➥Exception
➥ServiceBriefcaseServerException

✍	Throwable	**detail**
	String	**getMessage()**
	void	**printStackTrace()**
	void	**printStackTrace(java.io.PrintStream ps)**
	void	**printStackTrace(java.io.PrintWriter pw)**
❄		**ServiceBriefcaseServerException()**
❄		**ServiceBriefcaseServerException(String s)**
❄		**ServiceBriefcaseServerException(String s, Throwable ex)**

ServiceComponent — se.sics.sview.core

ServiceComponent

int	**initialize(ServiceContext context, ServiceContextEvent evt)**
int	**resume(ServiceContext context, ServiceContextEvent evt)**
int	**start(ServiceContext context, ServiceContextEvent evt)**
int	**stop(ServiceContext context, ServiceContextEvent evt)**
int	**suspend(ServiceContext context, ServiceContextEvent evt)**

ServiceComponentEvent — se.sics.sview.core

Object
➥ServiceComponentEvent

	String	**getKey()**
	String	**getName()**
	int	**getState()**
❇		**ServiceComponentEvent(String key, String name, int state)**

ServiceComponentHandling — se.sics.sview.core.permission

ServiceComponentHandling — AllPermissions

✍■	String	**description**

ServiceComponentListener — se.sics.sview.core

ServiceComponentListener

	void	**stateChanged(ServiceComponentEvent evt)**

ServiceComponentPermission — se.sics.sview.core

ServiceComponentPermission

✍■	String	**description**

ServiceContainer — se.sics.sview.core

Object
➥ServiceContainer — java.io.Serializable, Cloneable

✍♦	String	**activator**
✍♦	java.util.Date	**changeDate**
	ServiceComponent	**createServiceComponent(ClassLoader loader)** *throws* **java.io.IOException, ClassNotFoundException, InstantiationException, IllegalAccessException**
✍♦	java.util.Date	**creationDate**
	java.util.Date	**getCacheDate()**
	java.util.Date	**getChangeDate()**
	java.util.Date	**getCreationDate()**
	byte[]	**getJar()**
	java.io.InputStream	**getJarStream()**
	String	**getJarUrl()**
	String	**getKey()**
	java.util.Properties	**getProperties()**
	byte[]	**getServiceComponent()**
	boolean	**isMobile()**
	boolean	**isPersistent()**
✍♦	ServiceContainer.JarCache	**jarCache**
✍♦	String	**key**
❑	ServiceContainer	**load(java.io.InputStream is, String key)** *throws* **java.io.IOException, ClassNotFoundException**

	ServiceComponent	**loadServiceComponent(ClassLoader loader)** ***throws*** **java.io.IOException, ClassNotFoundException**
	void	**merge(ServiceContainer sc)**
✍♦	boolean	**mobile**
✍♦	Monitor	**monitor**
✍■	String	**P_CACHEDATE**
✍■	String	**P_CHANGEDATE**
✍■	String	**P_CREATIONDATE**
✍■	String	**P_JARURL**
✍■	String	**P_KEY**
✍■	String	**P_MOBILE**
✍■	String	**P_PERSISTENT**
✍♦	boolean	**persistent**
	void	**removeJar()**
❑	void	**save(ServiceContainer sc, java.io.OutputStream os)** ***throws*** **java.io.IOException**
	void	**saveServiceComponent(ServiceComponent s)** ***throws*** **java.io.IOException**
✍♦	byte[]	**serviceComponent**
❊		**ServiceContainer(String jarUrl, String key)** ***throws*** **java.net.MalformedURLException, java.io.IOException**
♦	void	**setChangeDate(java.util.Date changeDate)**
♦	void	**setCreationDate(java.util.Date creationDate)**
	void	**setMonitor(Monitor monitor)**
	void	**setServiceComponent(byte[] serviceComponent)**
	ServiceContainer	**stripJar()**
	ServiceContainer	**stripServiceComponent()**
	String	**toString()**
	void	**validateJar()**

ServiceContext — se.sics.sview.core

ServiceContext — Constants

	void	**addServiceComponentListener(String key, ServiceComponentListener listener)** ***throws*** **ServiceContextException**
	void	**addServiceContextListener(ServiceContextListener listener)**
	void	**createServiceComponent(String jarName)** ***throws*** **ServiceContextException**
	String	**getJarAttribute(String name)**
	byte[]	**getJarEntry(String name)**
	String	**getLocalProperty(String key)**
	String	**getLocalProperty(String key, String def)**
	String	**getMobileProperty(String key)**
	String	**getMobileProperty(String key, String def)**
	ServiceProxy	**getServiceProxy(String name)**
	int	**getState()**
	String	**getStationaryProperty(String key)**
	String	**getStationaryProperty(String key, String def)**

	void	**loadServiceComponent(java.io.InputStream is) *throws* ServiceContextException**
	void	**registerService(String name, ServiceInterfaceFactory interfaceFactory)**
	void	**reload() *throws* ServiceContextException**
	void	**remove()**
	void	**removeServiceComponent(String key) *throws* ServiceContextException**
	void	**removeServiceComponentListener(String key, ServiceComponentListener listener) *throws* ServiceContextException**
	void	**removeServiceContextListener(ServiceContextListener listener)**
	void	**resumeServiceComponent(String key, ServiceContextEvent evt) *throws* ServiceContextException**
	void	**save() *throws* ServiceContextException**
	void	**setMobileProperty(String key, String value)**
	void	**setState(int state)**
	void	**setStationaryProperty(String key, String value)**
	void	**shutdown() *throws* ServiceContextException**
	void	**stop()**
	void	**stopServiceComponent(String key, ServiceContextEvent evt) *throws* ServiceContextException**
	void	**suspend()**
	void	**suspendServiceComponent(String key, ServiceContextEvent evt) *throws* ServiceContextException**
	void	**synchronize() *throws* ServiceContextException**
	void	**unregisterService(String name)**

ServiceContextEvent — se.sics.sview.core

Object
➥ServiceContextEvent

❇		**ServiceContextEvent()**

ServiceContextException — se.sics.sview.core

Object
➥Throwable — java.io.Serializable
➥Exception
➥ServiceContextException

✍	Throwable	**detail**
	String	**getMessage()**
	void	**printStackTrace()**
	void	**printStackTrace(java.io.PrintStream ps)**
	void	**printStackTrace(java.io.PrintWriter pw)**
❇		**ServiceContextException()**
❇		**ServiceContextException(String s)**
❇		**ServiceContextException(String s, Throwable ex)**

ServiceContextListener	**se.sics.sview.core**
ServiceContextListener	
	void **serviceComponentAdded(ServiceComponentEvent evt)**
	void **serviceComponentRemoved(ServiceComponentEvent evt)**

ServiceEnvironmentHandling	**se.sics.sview.core.permission**
ServiceEnvironmentHandling	AllPermissions
✍■	String **description**

ServiceInterfaceFactory	**se.sics.sview.core**
ServiceInterfaceFactory	
	Object **createServiceInterface(String name)**

ServiceListener	**se.sics.sview.core**
ServiceListener	
	void **serviceRegistered(String name)**
	void **serviceUnregistered(String name)**

ServiceProxy	**se.sics.sview.core**
ServiceProxy	
	void **addServiceListener(ServiceListener listener)**
	void **removeServiceListener(ServiceListener listener)**
	Object **subscribe()**
	Object **subscribe(long timeout)** ***throws*** **InterruptedException**
	void **unsubscribe()**

ShutdownEnvironment	**se.sics.sview.core.permission**
ShutdownEnvironment	ServiceEnvironmentHandling
✍■	String **description**

ShutdownEvent	**se.sics.sview.core.event**
Object	
➥se.sics.sview.core.ServiceContextEvent	
➥StopEvent	
➥ShutdownEvent	
❄	**ShutdownEvent()**

StartEvent	**se.sics.sview.core.event**
Object	
➥se.sics.sview.core.ServiceContextEvent	
➥StartEvent	
❄	**StartEvent()**

StopComponent	**se.sics.sview.core.permission**
StopComponent	RuntimeHandling
✍■	String **description**

StopEvent	**se.sics.sview.core.event**
Object ➥se.sics.sview.core.ServiceContextEvent ➥StopEvent	
❄	**StopEvent()**

SuspendComponent	**se.sics.sview.core.permission**
SuspendComponent	RuntimeHandling
✍■	String **description**

SuspendEvent	**se.sics.sview.core.event**
Object ➥se.sics.sview.core.ServiceContextEvent ➥SuspendEvent	
❄	**SuspendEvent()**

SynchronizeEnvironment	**se.sics.sview.core.permission**
SynchronizeEnvironment	ServiceEnvironmentHandling
✍■	String **description**

TransactionCoordinator	**se.sics.sview.core**
TransactionCoordinator	
	void **abort()**
	void **acknowledge(TransactionParticipant tp)**

TransactionInitiator	**se.sics.sview.core**
TransactionInitiator	
	void **globalAcknowledge()**
	void **globalCommit(TransactionParticipant[] tps)**

TransactionParticipant	**se.sics.sview.core**
TransactionParticipant	
	void **globalCommit(TransactionCoordinator tc)** ***throws*** **Exception**
	boolean **vote()** ***throws*** **Exception**

UpdateEvent	se.sics.sview.core.event

Object
➥se.sics.sview.core.ServiceContextEvent
➥StopEvent
➥UpdateEvent

❄	**UpdateEvent()**