3GPP TSG CN Plenary Meeting #22 **NP-030547**
10 - 12 December 2003, Maui, Hawaii, USA

Source: **CN5 (OSA)**

Title: **Rel-5 CRs 29.198-01 OSA API Part 1: Overview**

Agenda item: **8.2**

Document for: **APPROVAL**

Doc-1st-Level	Spec	CR	R	Ph	Subject	Cat	Version-Current	Doc-2nd-Lev	WI
NP-030547	29.198-01	025	-	Rel-5	Add Java Realization rules to solve MPCC name conflicts	F	5.3.0	N5-030547	OSA2
NP-030547	29.198-01	026	-	Rel-5	Correction to Java Realisation Rulebook	F	5.3.0	N5-030634	OSA2

joint-API-group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-030547**
Meeting #25, Bangkok, THAILAND, 27 - 31 October 2003

CR-Form-v7

CHANGE REQUEST

⌘ **29.198-01 CR 025** ⌘ **rev - ⌘** Current version: **5.3.0** ⌘

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

Proposed change affects: UICC apps⌘ ☐ ME ☐ Radio Access Network ☐ Core Network [X]

Title:	⌘	Add Java Realization rules to solve MPCC name conflicts		
Source:	⌘	CN5 (AePONA & IBM)		
Work item code:	⌘	OSA2	***Date:*** ⌘	31/10/2003
Category:	⌘	**F**	***Release:*** ⌘	*REL-5*
		Use one of the following categories: ***F*** *(correction)* ***A*** *(corresponds to a correction in an earlier release)* ***B*** *(addition of feature),* ***C*** *(functional modification of feature)* ***D*** *(editorial modification)* Detailed explanations of the above categories can be found in 3GPP TR 21.900.		*Use one of the following releases:* *2 (GSM Phase 2)* *R96 (Release 1996)* *R97 (Release 1997)* *R98 (Release 1998)* *R99 (Release 1999)* *Rel-4 (Release 4)* *Rel-5 (Release 5)* *Rel-6 (Release 6)*

Reason for change:	⌘	There is a problem that has existed for a while that should be addressed with the Java Realization of the Multi Party Call Control Manager section. The problem primarily relates to the IpAppMultiPartyCall and IpAppCallLeg interfaces of the Multi Party Call Control SCF. These interfaces contain 4 methods that have the same names. In practice, it is usually desireable to implement an object that implements both of these interfaces. While Java in general allows an object to implement two interfaces that have a method by the same name (and signature), RMI/IIOP and CORBA do not allow methods from two interfaces with the same method names. The conflicting methods in IpAppCallLeg are: getInfoErr, getInfoRes, superviseErr, superviseRes. The associated request methods in IpCallLeg are: getInfoReq, superviseReq In order to fix the problem, we recommend modifying the method names in the IpAppCallLeg interface to include "CallLeg" as part of the method name, such as "getCallLegInfoRes", thereby removing the name conflict, and also the IpCallLeg interface for consistency.
Summary of change:	⌘	Provide a rule in the Multi Party Call Control section that provides renaming of these problematic methods, in order to avoid the name collisions. The method renaming is done in IpCallLeg and IpAppCallLeg for consistency.
Consequences if not approved:	⌘	There are other solutions to address this problem, however it complicates the application design, and each of these other solutions also have pitfalls and create various other design issues within the J2EE environment. The problems and complexities can be avoided most easily if there was no naming conflict in the interfaces.

Clauses affected:	⌘	Annex C

		Y	N			
Other specs affected:	⌘		X	Other core specifications	⌘	
			X	Test specifications		
			X	O&M Specifications		
Other comments:	⌘					

How to create CRs using this form:

C.5.5 Multi Party Call Control Specific Rules

The Multi Party Call Control Manager interface has specific Java Realisation considerations.

C.5.5.1 IpCallLeg and IpAppCallLeg method name conflicts

Some method names within the IpAppCallLeg interface have the same names as methods in the IpAppMultiPartyCall interface. These method names conflict when both interfaces are implemented on the same object within an RMI/IIOP or CORBA environment.

For the method names that are the same in both IpMultiPartyCall and IpCallLeg interfaces or IpAppMultiPartyCall and IpAppCallLeg, the call leg related method names are modified to include "CallLeg" as part of the method name to avoid name conflicts. The following method names result:

IpCallLeg Method Name	**Realisation Method Name**
getInfoReq	getCallLegInfoReq
superviseReq	superviseCallLegReq

Figure 1: IpCallLeg method name modifications

IpAppCallLeg Method Name	**Realisation Method Name**
getInfoRes	getCallLegInfoRes
getInfoErr	getCallLegInfoErr
superviseRes	superviseCallLegRes
superviseErr	superviseCallLegErr

Figure 2: IpAppCallLeg method name modifications

Annex D (informative): Change history

Change history							
Date	**TSG #**	**TSG Doc.**	**CR**	**Rev**	**Subject/Comment**	**Old**	**New**
Mar 2001	CN_11	NP-010134	047	--	CR 29.198: for moving TS 29.198 from R99 to Rel-4 (N5-010158)	3.2.0	4.0.0
Jun 2001	CN_12	NP-010330	001	--	Corrections to OSA API Rel4 (Correction to IDL namespace to align with that of ETSI and Parlay equivalent APIs: Change org.open_service_access root namespace to org.csapi) (N5-010267)	4.0.0	4.1.0
Sep 2001	CN_13	NP-010464	002	--	Changing references to JAIN	4.1.0	4.2.0
Dec 2001	CN_14	NP-010594	003	--	Replace Out Parameters with Return Types	4.2.0	4.3.0
Dec 2001	CN_14	NP-010594	004	--	Remove the perception that the OSA API only uses CORBA for its transport mechanism	4.2.0	4.3.0
Mar 2002	--	--	--	--	Editorial update (no CR) following Hong Kong CN5#16	4.3.0	4.3.1
Jun 2002	CN_16	NP-020181	005	--	Addition of support for Java API technology realisation	4.3.1	5.0.0
Jun 2002	CN_16	NP-020182	006	--	Addition of support for WSDL realisation	4.3.1	5.0.0
Jun 2002	CN_16	NP-020184	007	--	Adding the full naming convention for exceptions	4.3.1	5.0.0
Jun 2002	CN_16	NP-020184	008	--	Correction of References in OSA specifications	4.3.1	5.0.0
Jun 2002	CN_16	NP-020184	009	--	Addition of text describing the technology realisations of the Parlay/OSA specification	4.3.1	5.0.0
Sep 2002	CN_17	NP-020427	010	--	Addition to ObjectRef description in WSDL Mapping Rules	5.0.0	5.1.0
Sep 2002	CN_17	NP-020427	011	--	Addition of sequence tag to Choice types	5.0.0	5.1.0
Sep 2002	CN_17	NP-020427	012	--	Replace all occurrences of the xsd:anyURI type to xsd:string	5.0.0	5.1.0
Sep 2002	CN_17	NP-020427	013	--	Correction to Namespace mapping in WSDL Mapping Rules	5.0.0	5.1.0
Sep 2002	CN_17	NP-020427	014	--	Correction to xmlns:wsdl Namespace	5.0.0	5.1.0
Sep 2002	CN_17	NP-020427	015	--	Prepend class name to <message> name.	5.0.0	5.1.0
Sep 2002	CN_17	NP-020427	016	--	Correction to void return types in WSDL Mapping Rules	5.0.0	5.1.0
Sep 2002	CN_17	NP-020427	017	--	Add missing CORBA realization rules in Part 1	5.0.0	5.1.0
Sep 2002	CN_17	NP-020427	018	--	Add general introduction to the OSA APIs in Part 1	5.0.0	5.1.0
Sep 2002	CN_17	NP-020395	020	--	Add text to clarify relationship between 3GPP and ETSI/Parlay OSA specifications	5.0.0	5.1.0
Mar 2003	CN_19	--	--	--	Editorial update (no CR) following Bangkok CN5#22 (Introduction, Reference Titles)	5.1.0	5.1.1
Jun 2003	CN_20	NP-030298	022	1	Removal of un-used references	5.1.1	5.2.0
Jun 2003	CN_20	NP-030239	023	--	Correction to Java Realisation Annex	5.1.1	5.2.0
Sep 2003	CN_21	NP-030352	024	--	Correction to Java Realisation Annex	5.2.0	5.3.0
v							

joint-API-group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-030634**
Meeting #25, Bangkok, Thailand, 27 – 31 October 2003

CR-Form-v7

CHANGE REQUEST

⌘	**29.198-01**	**CR**	**026**	⌘ **rev**	**-**	⌘	Current version:	**5.3.0**	⌘

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

Proposed change affects: UICC apps⌘ ☐ ME ☐ Radio Access Network ☐ Core Network [✓ X]

Title:	⌘	Correction to Java Realisation Rulebook		
Source:	⌘	CN5 (AePONA – Eamonn Murray)		
Work item code:	⌘	OSA2	***Date:*** ⌘	31/10/2003
Category:	⌘	**F**	***Release:*** ⌘	*REL-5*

Use one of the following categories:
- ***F*** *(correction)*
- ***A*** *(corresponds to a correction in an earlier release)*
- ***B*** *(addition of feature),*
- ***C*** *(functional modification of feature)*
- ***D*** *(editorial modification)*

Detailed explanations of the above categories can be found in 3GPP TR 21.900.

Use one of the following releases:

2	*(GSM Phase 2)*
R96	*(Release 1996)*
R97	*(Release 1997)*
R98	*(Release 1998)*
R99	*(Release 1999)*
Rel-4	*(Release 4)*
Rel-5	*(Release 5)*
Rel-6	*(Release 6)*

Reason for change:	⌘	Application of the Java realisation rulebook to all parts of the OSA specification to produce the Java code has highlighted a number of minor errors in the rules themselves.
Summary of change:	⌘	The rules have been corrected based on feedback and validation from code generation and compilation
Consequences if not approved:	⌘	The existing Java rules are wrong, and the Java produced is inconsistent with the documented rules

Clauses affected:	⌘	C 3.6.5, C 3.6.6, C 3.6.7, C.4.5, C.4.8, C.4.9, C.4.11.1, C.4.11.2				
		Y	N			
Other specs affected:	⌘		X	Other core specifications	⌘	
			X	Test specifications		
			X	O&M Specifications		
Other comments:	⌘					

How to create CRs using this form:
Comprehensive information and tips about how to create CRs can be found at http://www.3gpp.org/specs/CR.htm. Below is a brief summary:

1) Fill out the above form. The symbols above marked ⌘ contain pop-up help information about the field that they are closest to.

2) Obtain the latest version for the release of the specification to which the change is proposed. Use the MS Word "revision marks" feature (also known as "track changes") when making the changes. All 3GPP specifications can be downloaded from the 3GPP server under ftp://ftp.3gpp.org/specs/ For the latest version, look for the directory name with the latest date e.g. 2001-03 contains the specifications resulting from the March 2001 TSG meetings.

3) With "track changes" disabled, paste the entire CR form (use CTRL-A to select it) into the specification just in front of the clause containing the first piece of changed text. Delete those parts of the specification which are not relevant to the change request.

************** Start of Change # 1 ***********************

C.3.6.5 NameValuePair (Enumerations)

NameValuePair data types are represented in Java as public final classes that implement java.io.Serializable, and have:

- two static final data members per name-value pair
- a value returning method, named getValue()
- a name returning method, named getValueText()
- an integer conversion method, named getObject()
- a private constructor
- hashCode and equals implementations

No default constructor is provided. One of the data members per name-value pair has the same name as the name-value pair name. The other has an underscore "_" prepended and is intended for use in switch statements. Values are assigned sequentially, starting with 0.

The getObject() method returns the name-value pair class with the specified value if the specified value corresponds to an element of the name-value pair data type. If the specified value is out of range, an InvalidEnumValueException exception is raised

Example 9:

```
package org.csapi.jr.se;
public final class AddressScreening implements java.io.Serializable {
        private int _value;
        private static int _size = 5;
        private static AddressScreening[] _array = new AddressScreening[_size];


        public static final int _ADDRESS_SCREENING_UNDEFINED = 0;
        public static final AddressScreening ADDRESS_SCREENING_UNDEFINED = new
AddressScreening(_ADDRESS_SCREENING_UNDEFINED);

        public static final int _ADDRESS_SCREENING_USER_VERIFIED_PASSED = 1;
        public static final AddressScreening ADDRESS_SCREENING_USER_VERIFIED_PASSED = new
AddressScreening(_ADDRESS_SCREENING_USER_VERIFIED_PASSED);

        public static final int _ADDRESS_SCREENING_USER_NOT_VERIFIED = 2;
        public static final AddressScreening ADDRESS_SCREENING_USER_NOT_VERIFIED = new
AddressScreening(_ADDRESS_SCREENING_USER_NOT_VERIFIED);

        public static final int _ADDRESS_SCREENING_USER_VERIFIED_FAILED = 3;
        public static final AddressScreening ADDRESS_SCREENING_USER_VERIFIED_FAILED = new
AddressScreening(_ADDRESS_SCREENING_USER_VERIFIED_FAILED);

        public static final int _ADDRESS_SCREENING_NETWORK =  4;
        public static final AddressScreening ADDRESS_SCREENING_NETWORK = new
AddressScreening(_ADDRESS_SCREENING_NETWORK);

        private AddressScreening(int value) {
            this._value = value;
            this._array[this._value] = this;
        }


        public int getValue() {
            return _value;
        }

        public String getValueText() {
            switch (_value) {
            case _ADDRESS_SCREENING_UNDEFINED:
                return "ADDRESS_SCREENING_UNDEFINED";
            case _ADDRESS_SCREENING_USER_VERIFIED_PASSED:
                return "ADDRESS_SCREENING_USER_VERIFIED_PASSED";
            case _ADDRESS_SCREENING_USER_NOT_VERIFIED:
                return "ADDRESS_SCREENING_USER_NOT_VERIFIED";
            case _ADDRESS_SCREENING_USER_VERIFIED_FAILED:
                return "ADDRESS_SCREENING_USER_VERIFIED_FAILED";
            case _ADDRESS_SCREENING_NETWORK:
                return "ADDRESS_SCREENING_NETWORK";
            default:
                return "ERROR";
            }
        }

        public static AddressScreening getObject(int value) throws
org.csapi.jr.se.InvalidEnumValueException {
            if(value >= 0 && value < _size) {
                return _array[value];
            } else {
                throw new org.csapi.jr.se.InvalidEnumValueException();
            }
        }

        public boolean equals(Object o) {
        //equality logic
        }

        public int hashCode() {
        //hash code calculation
            ~~return _value;~~
        }

        ~~public static AddressScreening getObject(int value) throws~~
~~org.csapi.jr.se.InvalidEnumValueException {~~
            ~~if(value >= 0 && value < _size) {~~
                ~~return _array[value];~~
            ~~} else {~~
                ~~throw new org.csapi.jr.se.InvalidEnumValueException();~~
            ~~}~~
        ~~}~~

        ~~private AddressScreening(int value) {~~
            ~~this._value = value;~~
            ~~this._array[this._value] = this;~~
        ~~}~~

}
```

C.3.6.6 TaggedChoiceOfDataElements (Unions)

Union data types are represented in Java as public final classes that implement java.io.Serializable, and have:

- a default constructor
- a discriminator field
- a discriminator accessor method, named getDiscriminator()
- an accessor and modifier method for each data element, the names of which are derived from choice element name
- hashCode and equals implementations

Conflicting names should be resolved by prefixing the field name with an underscore for getDiscriminator if there is a name clash with the mapped data type name or any of the data element names.

Where choice element type and choice element name are "NULL" and "Undefined", respectively, a Java Object set as null replaces the NULL. If multiple NULL/Undefined combinations occur in the tagged choice of data elements, the method, setUndefined, will receive the discriminator as a parameter and set _object to null.

Accessor methods shall raise an InvalidUnionAccessorException exception if the expected data element has not been set.

Example 10:

```
package org.csapi.jr.se;
public final class AoCOrder implements java.io.Serializable {
        private CallAoCOrderCategory _discriminator = null;
        private java.lang.Object _object;

        public AoCOrder() {
        }

        public CallAoCOrderCategory getDiscriminator() throws
org.csapi.jr.se.InvalidUnionAccessorException {
            if(_discriminator == null) {
                throw new org.csapi.jr.se.InvalidUnionAccessorException();
            }
            return _discriminator;
        }

        public org.csapi.jr.se.ChargeAdviceInfo getChargeAdviceInfo() throws
org.csapi.jr.se.InvalidUnionAccessorException {
            if (~~!(~~_discriminator.~~equals((CallAoCOrderCategory)~~ !=
CallAoCOrderCategory.CHARGE_ADVICE_INFO~~))~~) {
                throw new org.csapi.jr.se.InvalidUnionAccessorException();
            }
            return ((org.csapi.jr.se.ChargeAdviceInfo) _object);
        }

        public void setChargeAdviceInfo(org.csapi.jr.se.ChargeAdviceInfo value) {
            _discriminator = ~~(CallAoCOrderCategory) ~~CallAoCOrderCategory.CHARGE_ADVICE_INFO;
            _object = value;
        }

        public org.csapi.jr.se.ChargePerTime getChargePerTime() throws
org.csapi.jr.se.InvalidUnionAccessorException {
            if (~~!(~~_discriminator.~~equals((CallAoCOrderCategory)~~!=
CallAoCOrderCategory.CHARGE_PER_TIME~~))~~) {
                throw new org.csapi.jr.se.InvalidUnionAccessorException();
            }
            return ((org.csapi.jr.se.ChargePerTime) _object);
        }

        public void setChargePerTime(org.csapi.jr.se.ChargePerTime value) {
            _discriminator = ~~(CallAoCOrderCategory) ~~CallAoCOrderCategory.CHARGE_PER_TIME;
            _object = value;
        }

        public java.lang.String getNetworkCharge() throws
org.csapi.jr.se.InvalidUnionAccessorException {
            if (~~!(~~_discriminator.~~equals((CallAoCOrderCategory)~~!=
CallAoCOrderCategory.CHARGE_NETWORK~~))~~) {
                throw new org.csapi.jr.se.InvalidUnionAccessorException();
            }
            return ((java.lang.String) _object);
        }

        public void setNetworkCharge(java.lang.String value) {
            _discriminator = ~~(CallAoCOrderCategory) ~~CallAoCOrderCategory.CHARGE_NETWORK;
            _object = value;
        }

        public void setUndefined(CallAocOrderCategory discriminator) {
            _discriminator = discriminator;
            _object = null;
        }

        public boolean equals(Object o) {
        //equality logic
        }

        public int hashCode() {
        //hash code calculation
        }

}
```

C.3.6.7 Exceptions

An exception maps to a constructed exception, providing appropriate constructors and accessor methods for the data contained within the exception. Each exception is defined as a public class extending java.lang.Exception, and containing a private field for each information element contained within the exception.

A default constructor is provided, along with a constructor containing only an embedded exception, a constructor containing a list of the fields in the exception and a constructor that contains the fields plus an embedded exception.

An accessor method is provided for each field, and for the embedded exception.

The following Java Realisations apply to mapping of exceptions:

- PlatformException
- P_XXX_XXX Exceptions
- TpCommonExceptions
- TpCommonExceptions' associated exceptions
- Additional abstract exceptions
- InvalidUnionAccessorException
- InvalidEnumValueException

C.3.6.7.1 PlatformException

PlatformException exception handles local platform and communication problem exceptions.

Example 11:

```
package org.csapi.jr.se;
public class PlatformException extends java.lang.RuntimeException {
        private Throwable _cause = null;

        public PlatformException () {
            super();
        }

        public PlatformException (String message) {
            super(message);
        }

        public PlatformException (String message, Throwable cause) {
            super(message);
            _cause = cause;
        }

        public PlatformException (Throwable cause) {
            _cause = cause;
        }

        public Throwable getCause() {
            return _cause;
        }
}
```

C.3.6.7.2 P_XXX_XXX Exceptions

P_XXX_XXX exceptions follow the XxxXxxException naming pattern, and inherit from java.lang.Exception.

Example 12:

```
package org.csapi.jr.se;
public class InvalidInterfaceTypeException extends java.lang.Exception {
        private Throwable _cause = null;

        public InvalidInterfaceTypeException() {
            super();
        }

        public InvalidInterfaceTypeException(String message) {
            super(message);
        }

        public InvalidInterfaceTypeException(String message,Throwable cause) {
            super(message);
            _cause = cause;
        }

        public InvalidInterfaceTypeException(Throwable cause) {
            _cause = cause;
        }

        public Throwable getCause() {
            return _cause;
        }
}
```

## C.3.6.7.3	TpCommonExceptions

The name for TpCommonExceptions exception is made singular, i.e. CommonException, and inherits from java.lang.Exception.

Example 13:

```
package org.csapi.jr.se;
public class CommonException extends java.lang.Exception {
        private Throwable _cause = null;
        private int _exceptionType;
        private String _extraInformation;


        public CommonException () {
            super();
        }

        public CommonException (String message) {
            super(message);
        }
        public CommonException (String message, Throwable cause) {
            super(message);
            _cause = cause;
        }

        public CommonException (Throwable cause) {
            _cause = cause;
        }


        public Throwable getCause() {
            return _cause;
        }

        public int getExceptionType() {
            return _exceptionType;
        }

        public ~~int ~~void setExceptionType(int exceptionType) {
            ~~return ~~_exceptionType = exceptionType;
        }

        public String getExtraInformation() {
            return _extraInformation;
        }

        public ~~String ~~void setExtraInformation(String extraInformation) {
            ~~return ~~_extraInformation = extraInformation;
        }
}
```

C.3.6.7.4 TpCommonException's associated exceptions

P_XXX_XXX exception types (constants) associated with TpCommonExceptions follow the XxxXxxException naming pattern and inherit from CommonException.

Example 14:

```
package org.csapi.jr.se;
public class ResourcesUnavailableException extends org.csapi.jr.se.CommonException {
        private Throwable _cause = null;

        public ResourcesUnavailableException () {
            super();
        }

        public ResourcesUnavailableException (String message) {
            super(message);
        }

        public ResourcesUnavailableException (String message, Throwable cause) {
            super(message, cause);
        }

        public ResourcesUnavailableException (Throwable cause) {
            _cause = cause;
        }


}
```

C.3.6.7.5 Additional abstract exceptions

Additional abstract exceptions (See ETSI ES 202 915-2, Annex D) have been defined which are TpInvalidArgumentException, TpFrameworkException, TpMobilityException, TpDataSessionException, TpMessagingException, TpConnectivityException, TpAccountException, TpPAMException and TpPolicyException and are mapped as follows:

Example 15:

```
package org.csapi.jr.se;
public class InvalidArgumentException extends java.lang.Exception {
        private Throwable _cause = null;

        public InvalidArgumentException () {
            super();
        }

        public InvalidArgumentException (String message) {
            super(message);
        }

        public InvalidArgumentException (String message, Throwable cause) {
            super(message);
            _cause = cause;
        }

        public InvalidArgumentException (Throwable cause) {
            _cause = cause;
        }

        public Throwable getCause() {
            return _cause;
        }
}
```

C.3.6.7.6 InvalidUnionAccessorException

An additional exception, InvalidUnionAccessorException, is defined which indicates that the expected data element has not been set.

Example 16:

```
package org.csapi.jr.se;
public class InvalidUnionAccessorException extends org.csapi.jr.se.InvalidArgumentException {
        private Throwable _cause = null;

        public InvalidUnionAccessorException (){
            super ();
        }

        public InvalidUnionAccessorException (String message){
            super (message);
        }

        public InvalidUnionAccessorException (String message, Throwable cause){
            super (message,cause);
        }

        public InvalidUnionAccessorException (Throwable cause) {
            _cause = cause;
        }

}
```

C.3.6.7.7 InvalidEnumValueException

An additional exception, InvalidEnumValueException, is defined which indicates that an enum data type was accessed with an invalid request value.

Example 17:

```
package org.csapi.jr.se;
public class InvalidEnumValueException extends org.csapi.jr.se.InvalidArgumentException {
        private Throwable _cause = null;

        public InvalidEnumValueException () {
            super ();
        }

        public InvalidEnumValueExceptions (String message) {
            super (message);
        }

        public InvalidEnumValueException (String message, Throwable cause) {
            super (message,cause);
        }

        public InvalidEnumValueException (Throwable cause) {
            _cause = cause;
        }

}
```

************** End of Change # 1 ************************

************** Start of Change # 2 **********************

C.4.5 Mapping of IpService

IpService interface is represented by the Java Service interface. This provides a common interface for related interfaces to inherit from.

Example 23:

Service Interface:

```
package org.csapi.jr.se;
public interface Service extends CsapiInterface {
        public final static int IN_SERVICE_STATE=0 ;
        public final static int OUT_OF_SERVICE_STATE=1;

        void addServiceStateChangeListener(ServiceStateChangeListener listener)
        int getServiceState();
        void removeServiceStateChangeListener( ServiceStateChangeListener listener) ;
}
```

Listener interface:

```
package org.csapi.jr.se;
public interface ServiceStateChangeListener extends java.util.EventListener {
        void onOutOfService(OutOfServiceEvent event);
}
```

Event class:

```
package org.csapi.jr.se;
public class OutOfServiceEvent extends  jav.util.EventObject {
    public OutOfServiceEvent(java.lang.Object source){
        super(source)
    }
}
```

C.4.8 Mapping of TpAssignmentID to the creation of an Activity object.

The UML TpAssignmentID data types, which differentiate between multiple parallel asynchronous method invocations (activities) on the same (“parent”) interface, are deleted and replaced with createXxx methods (one for each parallel asynchronous activity) that create (“child”) activity interfaces. Where this would result in method names of the pattern createCreateXxx, this should be changed to method names with the pattern createXxx. Associated listeners would then remove the Create prefix from their name. These activity interfaces, in addition to possibly supporting other methods, will support one of the previously mentioned multiple parallel asynchronous method invocations. Hence, the Java API realisation creates multiple (activity) objects and invokes a single request per object rather than creating a single object and invoking multiple requests on that object, each request being differentiated using the TpAssignmentID value. The results of the asynchronous method invocation will be handled by the activity interface’s listener interface. To create the activity interface, the original IpXxx interface (to be named Xxx) will replace its parallel supporting asynchronous method invocations, yyyYyyReq, with createYyyYyy methods that take no parameters but returns the activity interface, YyyYyy. Where this would result in method names of the pattern createCreateXxx, this should be changed to method names with the pattern createXxx. Associated listeners would then remove the Create prefix from their name. The activity interface will extend Activity interface (see next rule), have a simple FSM, the addYyyYyyListener, removeYyyYyyListener and the asynchronous method that previously supported a parallel capability (typically named yyyYyyReq, but also yyyYyyStop).

An Activity interface, packaged in org.csapi.jr.se, is added as a parent to all activity interfaces. An application may add listeners of type ActivityStateChangeListener to an Activity if it wishes be explicitly informed when the activity becomes invalid.

The YyyYyyListener activity listener interfaces will extend java.util.EventListener. The asynchronous methods of previously named IpAppXxx, typically labelled yyyYyyRes and yyyYyyErr but also yyyYyy, will be renamed onYyyYyyRes and onYyyYyyErr but also onYyyYyy. Each method will have an event parameter, typically labelled YyyYyyResEvent and YyyYyyErrEvent, but also YyyYyyEvent. Events will be classes that extend java.util.EventObject and contain a public constructor (with multiple parameters – one per class carried by the event) and a number of public getter methods (one per “gettable” class carried by the event). As a result of adding activity listener interfaces, this may cause the requirement for the original IpAppXxx to disappear, since the yyyYyyRes and yyyYyyErr methods will effectively be ported to the activity listener interfaces.

For data types that contain TpAssignmentID the activity object replaces the TpAssignmentID.

Example 25:

Activity Interface:

```
package org.csapi.jr.se;
public interface Activity  extends CsapiInterface {
        public final static int IDLE_STATE = 0;
        public final static int ACTIVE_STATE = 1;
        public final static int INVALID_STATE = 2;
        public int getState();
        public void addActivityStateChangeListener(ActivityStateChangeListener listener);
        public void removeActivityStateChageListener(ActivityStateChangeListener listener);
}
```

Activity Listener Interface and Event class:

```
package org.csapi.jr.se;
public interface ActivityStateChangeListener {
        onInvalidStateEvent  (InvalidActivityEvent event)
}

public class InvalidActivityEvent extends java.util.EventObject {
    public InvalidActivityEvent(java.lang.Object source){
        super(source)
    }

}
```

Parent interface:

```
package org.csapi.jr.se.mmm.ul;
public interface UserLocation extends org.csapi.jr.se.Service {
        public LocationReport createLocationReport();
        public ExtendedLocationReport createExtendedLocationReport();
        public PeriodicLocationReporting createPeriodicLocationReporting();
}
```

Child Interface:

```
package org.csapi.jr.se.mm.ul;
public interface LocationReport extends org.csapi.jr.se.Activity {
        public void addLocationReportListener(LocationReportListener listener)
        public void removeLocationReportListener(LocationReportListener listener)
        public void locationReportReq(Address[] users) throws …
}
```

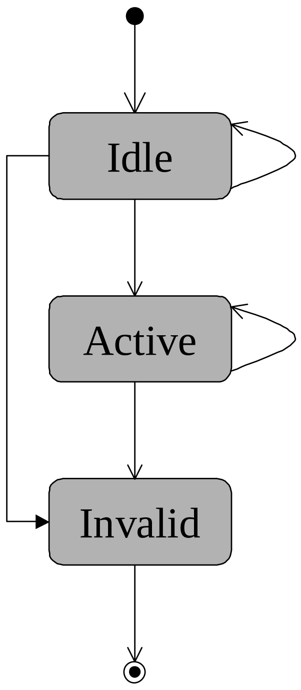
Listener Interface:

```
package org.csapi.jr.se.mm.ul;
public interface LocationReportListener extends ~~CsapiInterface, ~~java.util.EventListener {

        public void onLocationReportResEvent(LocationReportResEvent event);
        public void onLocationReportErrEvent(LocationReportErrEvent event);
}
```

Event classes:

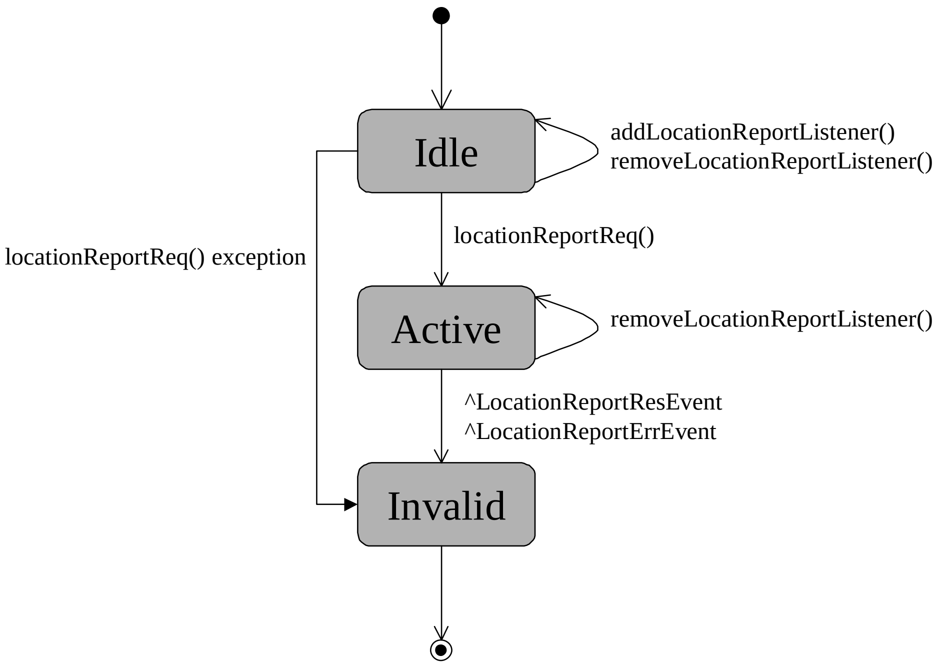
```
package org.csapi.jr.se.mmm.ul;
public class LocationReportResEvent extends java.util.EventObject{
        // with a public UserLocation[] constructor and a public getter
        // method for the parameter of the event
}

public classLocationReportErrEvent extends java.util.EventObject {
        // with a public MobilityError and MobilityDiagnostic constructor
        // and two public getter methods, one for each of the parameters
        // of the event
}
```

The Finite State Model for the Activity interface is given below:

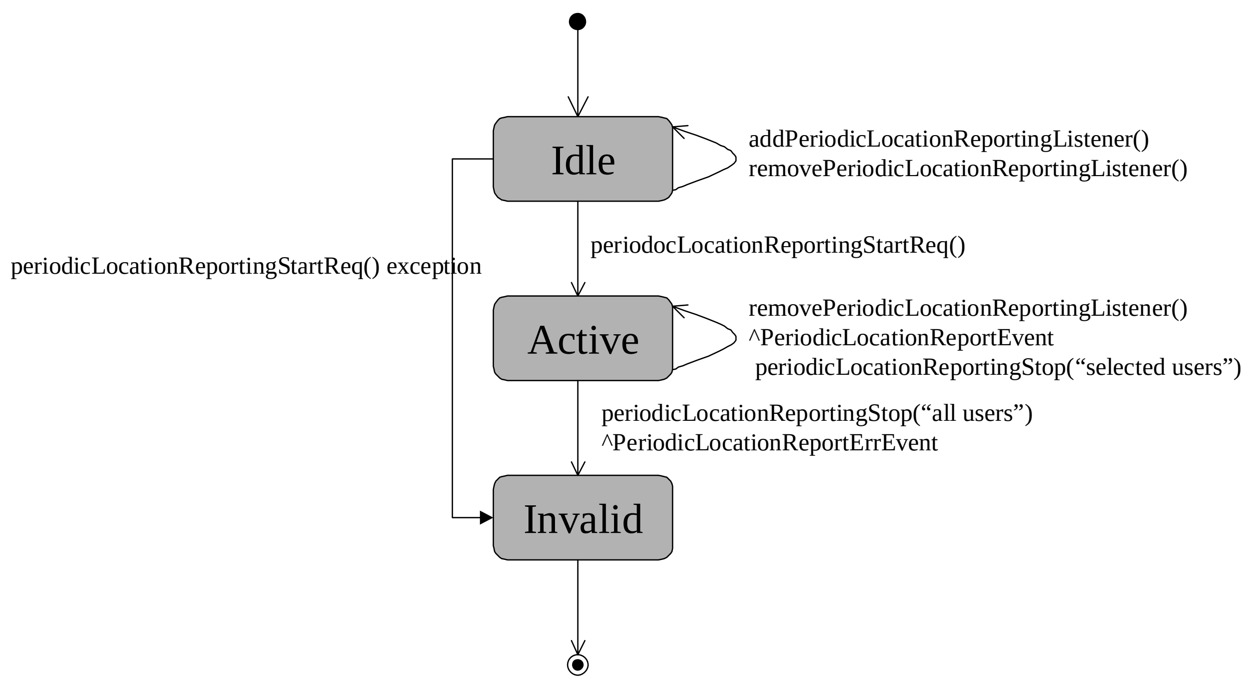


This interface specifies an activity, which might be provided by a service. An activity has three states: "idle", "active" and "invalid". The initial state is "idle" and here the listeners should be registered. It performs in the "active" state. It enters the "invalid" state when it has fulfilled its task or a fatal error occurred. In special cases state transition from "idle" to "invalid" is possible.

An example activity interface FSM is given below for a single activity request with a single response:


Idle
addLocationReportListener()
removeLocationReportListener()
locationReportReq()
locationReportReq() exception
Active
removeLocationReportListener()
^LocationReportResEvent
^LocationReportErrEvent
Invalid

An example activity interface FSM is given below for a single activity request with repeating responses:


Idle
addPeriodicLocationReportingListener()
removePeriodicLocationReportingListener()
periodocLocationReportingStartReq()
periodicLocationReportingStartReq() exception
Active
removePeriodicLocationReportingListener()
^PeriodicLocationReportEvent
periodicLocationReportingStop("selected users")
periodicLocationReportingStop("all users")
^PeriodicLocationReportErrEvent
Invalid

C.4.9 Callback Rule

The UML callback design pattern for non client-to-service interfaces (Parlay interface numbers 1, 3, 4, 5 and 6 [Fig 1]) is represented in Java with the callback design pattern. The UML callback design pattern for client-to-service interfaces (Parlay interface number 2 [Fig 1]) is represented in Java with the event listener design pattern.

The UML client-to-service interfaces (Parlay interface number 2) with the IpAppXxxx naming convention are represented in Java with the XxxxListener naming convention.

The IpService.setCallback method can be deleted; the interfaces that inherited the setCallback method now have associated addXxxxListener and removeXxxxListener methods. According to the *TpSessionID* mapping, IpService.setCallbackWithSessionID() method is deleted.

The XxxxListener listener interfaces will extend java.util.EventListener. The asynchronous methods of previously named IpAppXxxx, typically labelled yyyyYyyyRes and yyyyYyyyyErr but also yyyyYyyy, will be renamed onYyyyYyyyRes and onYyyyYyyyErr but also onYyyyYyyy. Each method will have an event parameter, typically labelled YyyyYyyyResEvent and YyyyYyyyErrEvent, but also YyyyYyyyEvent. Events will be classes that extend java.util.EventObject and contain a private constructor (with multiple parameters – one per class carried by the event) and a number of public getter methods (one per "gettable" class carried by the event). Events are read-only and serializable.

Example 26:

Listener Interface:

```
package org.csapi.jr.se.cc.mpccs;

MultiPartyCallListener extends ~~CsapiInterface, ~~java.util.EventListener{

public void onGetInfoResEvent(GetInfoResEvent event)
public void onGetInfoErrEvent(GetInfoErrEvent event)
public void onSuperviseResEvent(SuperviseResEvent event)
public void onSuperviseErrEvent(SuperviseErrEvent event)
public void onCallEndedEvent(CallEndedEvent event)
public void onCreateAndRouteCallLegErrEvent(CreateAndRouteCallLegErrEvent event)
}
```

MuliPartyCall Interface additional methods:

```
public void addMultiPartyCallListener(MultiPartyCallListener multiPartyCallListener);
public void removeMultiPartyCallListener(MultiPartyCallListener multiPartyCallListener);
```

************** End of Change # 2 ************************

************** Start of Change # 3 ***********************

C.4.11.1 PeerUnavailableException

PeerUnavailableException indicates failure to access an implementation of the Initial interface.

Example 28:

```
public class PeerUnavailableException extends java.lang.Exception {
        private Throwable _cause = null;
        public PeerUnavailableException () {
            super();
        }

        public PeerUnavailableException (String message) {
            super(message);
        }

        public PeerUnavailableException (String message, Throwable cause) {
            super(message);
            _cause = cause;
        }

        public PeerUnavailableException (Throwable cause) {
            _cause = cause;
        }

        public Throwable getCause() {
            return _cause;
        }
}
```

C.4.11.2 IllegalStateException

IllegalStateException exception signals that a method has been invoked at an illegal or inappropriate time.

Example 29:

```
package org.csapi.jr.se;
public class IllegalStateException extends java.lang.Exception {

        private int _state;
        private. java.lang.Object _object;

        public IllegalStateException(Object object, int state) {
            super();
            _object = object;
            _state  = state;
        }

        public Illegal StateException(Object object, int state, String s) {
            super(s);
            _object = object;
            _state  = state;
        }

        public Object getObject() {
            return _object;
        }
        public int getState() {
            return _state;
        }
}
```

************** End of Change # 3 ***********************

Annex D (informative): Change history

Change history							
Date	**TSG #**	**TSG Doc.**	**CR**	**Rev**	**Subject/Comment**	**Old**	**New**
Mar 2001	CN_11	NP-010134	047	--	CR 29.198: for moving TS 29.198 from R99 to Rel-4 (N5-010158)	3.2.0	4.0.0
Jun 2001	CN_12	NP-010330	001	--	Corrections to OSA API Rel4 (Correction to IDL namespace to align with that of ETSI and Parlay equivalent APIs: Change org.open_service_access root namespace to org.csapi) (N5-010267)	4.0.0	4.1.0
Sep 2001	CN_13	NP-010464	002	--	Changing references to JAIN	4.1.0	4.2.0
Dec 2001	CN_14	NP-010594	003	--	Replace Out Parameters with Return Types	4.2.0	4.3.0
Dec 2001	CN_14	NP-010594	004	--	Remove the perception that the OSA API only uses CORBA for its transport mechanism	4.2.0	4.3.0
Mar 2002	--	--	--	--	Editorial update (no CR) following Hong Kong CN5#16	4.3.0	4.3.1
Jun 2002	CN_16	NP-020181	005	--	Addition of support for Java API technology realisation	4.3.1	5.0.0
Jun 2002	CN_16	NP-020182	006	--	Addition of support for WSDL realisation	4.3.1	5.0.0
Jun 2002	CN_16	NP-020184	007	--	Adding the full naming convention for exceptions	4.3.1	5.0.0
Jun 2002	CN_16	NP-020184	008	--	Correction of References in OSA specifications	4.3.1	5.0.0
Jun 2002	CN_16	NP-020184	009	--	Addition of text describing the technology realisations of the Parlay/OSA specification	4.3.1	5.0.0
Sep 2002	CN_17	NP-020427	010	--	Addition to ObjectRef description in WSDL Mapping Rules	5.0.0	5.1.0
Sep 2002	CN_17	NP-020427	011	--	Addition of sequence tag to Choice types	5.0.0	5.1.0
Sep 2002	CN_17	NP-020427	012	--	Replace all occurrences of the xsd:anyURI type to xsd:string	5.0.0	5.1.0
Sep 2002	CN_17	NP-020427	013	--	Correction to Namespace mapping in WSDL Mapping Rules	5.0.0	5.1.0
Sep 2002	CN_17	NP-020427	014	--	Correction to xmlns:wsdl Namespace	5.0.0	5.1.0
Sep 2002	CN_17	NP-020427	015	--	Prepend class name to <message> name.	5.0.0	5.1.0
Sep 2002	CN_17	NP-020427	016	--	Correction to void return types in WSDL Mapping Rules	5.0.0	5.1.0
Sep 2002	CN_17	NP-020427	017	--	Add missing CORBA realization rules in Part 1	5.0.0	5.1.0
Sep 2002	CN_17	NP-020427	018	--	Add general introduction to the OSA APIs in Part 1	5.0.0	5.1.0
Sep 2002	CN_17	NP-020395	020	--	Add text to clarify relationship between 3GPP and ETSI/Parlay OSA specifications	5.0.0	5.1.0
Mar 2003	CN_19	--	--	--	Editorial update (no CR) following Bangkok CN5#22 (Introduction, Reference Titles)	5.1.0	5.1.1
Jun 2003	CN_20	NP-030298	022	1	Removal of un-used references	5.1.1	5.2.0
Jun 2003	CN_20	NP-030239	023	--	Correction to Java Realisation Annex	5.1.1	5.2.0
Sep 2003	CN_21	NP-030352	024	--	Correction to Java Realisation Annex	5.2.0	5.3.0
v							