3GPP TSG CN Plenary Meeting #20 **NP-030239**
04-06 June 2003. Hämeenlinna, FINLAND

Source: **CN5 (OSA)**

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

Agenda item: **8.2**

Document for: **APPROVAL**

Doc-1st-Level	Spec	CR	R	Ph	Subject	Cat	Ver-Curr	Doc-2nd-Level	WI
NP-030239	29.198-01	022	-	Rel-5	Removal of un-used references	F	5.1.1	N5-030300	OSA2
NP-030239	29.198-01	023	-	Rel-5	Correction to Java Realisation Annex	F	5.1.1	N5-030275	OSA2

CR-Form-v7

CHANGE REQUEST

⌘ **29.198-01** **CR** **023** ⌘ **rev** **-** ⌘ Current version: **5.1.1** ⌘

*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 ☒

Title:	⌘	Correction to Java Realisation Annex
Source:	⌘	AePONA, Ann-Marie Mulholland, IBM, Joe McIntyre
Work item code:	⌘	OSA2
Date:	⌘	08/05/2003
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:	⌘	Correction to Annex C of the current specification. The current Annex references Jain SPA. Jain SPA is no longer a supported activity or deliverable. Replace the current Annex with the Parlay Java Realisation as an informative Annex to the body of OSA API specfication deliverables.
Summary of change:	⌘	Replace the current Annex C that refers to Jain SPA as the informative Java Realisation with the Java Realisation rulebook produced by the Parlay Java Realisation Workgroup.
Consequences if not approved:	⌘	The API specification will reference a realisation that is no longer supported or valid.

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:
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.

Explanatory Notes

This submission is to take the form of an Annex to Open Service Access (OSA); Application Programming Interface (API); Part 1: Overview (3GPP TS 29.198-01 V5.1.1).

Sections C1 through C4 are taken from the Parlay UML to Java API Rulebook (Issue 1.1) produced by the Parlay Java Realisation Working Group. These rules comprise both common (applicable to both J2SE and J2EE APIs) and J2SE specific mappings to produce a J2SE Java Realisation of the Parlay API.

Due to time constraints on the delivery of the Parlay UML to Java API Rulebook, the production of relevant J2EE rules was not progressed within the Parlay workgroup. An additional section C5 not included in the Parlay UML to Java API Rulebook (Issue 1.1) details new submissions for J2EE specific mappings which, along with the common mappings, produce a J2EE Java Realisation of the Parlay API.

The J2SE Java Realisation API provides a Parlay to Java realisation for client application development. It is transport independent and provides an API which follows common Java coding patterns to aid application development.

The J2EE Java Realisation API provides a Parlay to Java realisation for a middleware API allowing clients and gateways to communicate using a common J2EE remote, J2EE local and RMI programming interface.

The following points covers areas where this contribution **differs** from the Parlay UML to Java API Rulebook:

- Package Namespace - the J2SE API will reside under the org.caspi.jr.se namespace and the J2EE local API will reside under the org.csapi.jr.ee namespace, and the J2EE/RMI remote interface will reside under the org.csapi.jr.ee.remote namespace.
- Package Naming Rule - framework packages have not been abbreviated; to keep the package naming consistent across the entire API.
- TpDate, TpTime and TpDateAndTime Rule - considered erroreous in that java.util.Calendar can only represent absolute times and not relative times, and therefore not included here.
- IllegalStateException added - this exception has been added as it is should be a checked exception.

~~Annex C (informative):~~
~~Java API~~

~~C.1 Tools and Languages~~

~~The Java language is used as a means to programmatically define the interfaces. Java files are either generated manually from class diagrams or by using a UML tool and editing scripts. Either way, the Java files are generated by the JAIN Community (http://www.java.sun.com/products/jain) in accordance with the Parlay UML to Java API Rulebook, which define a set of rules that are used to rapidly generate the Java APIs from the OSA/Parlay UML.~~

~~The generated Java files are verified using Java compilers such as javac. The Java API specifications are designed to be compatible with the Java 2 SDK, Standard Edition, version 1.4.0 (http://java.sun.com/j2se/1.4/docs/relnotes/features.html) or later. The Java API Realizations of the OSA/Parlay APIs are known as the JAIN Service Provider APIs (JAIN SPA).~~

~~C.2 JAIN SPA Overview~~

~~JAIN SPA is a local Java API realization of the OSA/Parlay specifications. The benefits of providing a local API (in addition to a distribution or remote API, such as the OSA/Parlay OMG-IDL or the OSA/Parlay W3C WSDL) is that the API is tailored to a particular programming language (in this case it's Java), which is distribution mechanism independent, meaning that, providing the necessary adapters are put in place, Java applications can be written to this local API that use any form of technology (e.g. CORBA, SOAP, RMI) for the purpose of distributing this API. With remote APIs, although the programmer may be free to write in multiple programming languages, he needs knowledge of, and is committed to, the particular distribution mechanism (e.g. CORBA, SOAP, RMI).~~

~~As the OSA/Parlay UML assumes a remote API, many optimizations have been made to the specifications, which, although acceptable to a "specialist" programmer taking distribution into account, would appear alien to the large community of "regular" Java programmers. As such, the JAIN SPA specifications are tailored to the Java language by following Java language naming conventions, design patterns and object oriented practices for a local Java API, while reusing as much Java codebase as possible. JAIN Service Provider APIs are developed by the JAIN Community under the Java Community Process (JCP) (http://jcp.org/). Within the JCP, each JAIN Service Provider API is developed by submitting a Java Specification Request (JSR) (http://jcp.org/jsr/overview/index.en.jsp). Each JAIN Service Provider API is assigned a JSR number, and an associated webpage, that can be used to identify it.~~

~~Each JSR webpage contains a table identifying the relationships between the different versions of the Parlay, ETSI/OSA, 3GPP/OSA and JAIN SPA specifications. In addition, each JAIN SPA specification version indicates to which Parlay, ETSI/OSA and 3GPP/OSA specification versions it corresponds to.~~

Annex C (informative):
Java Realisation API

C.1 Java Realisation Overview

The Parlay/OSA UML specifications are defined in a technology neutral manner. This annex aims to deliver for Java, a developer API, provided as a realisation, supporting a Java API that represents the UML specifications.

C.1.1 J2SE API

The J2SE API supports a J2SE development environment that

- provides an abstraction of the Parlay/OSA APIs that provides a local API for J2SE developers
- supports a listener based API for SCFs and a callback API for the Framework
- uses local object references as correlation mechanisms as Java developers are familiar with object correlation
- is a local API without visibility to the underlying transport

C.1.2 J2EE API

The J2EE API supports a development environment which allows the creation of J2EE and Java RMI interfaces for both the server and client, ensuring consistent interfaces for interoperability. These interfaces may be used for Java RMI on either JRMP or IIOP (RMI/IIOP), allowing use in J2EE environments. The interfaces may also be used as a thin layer on other transports, similar to other Java technologies that provide a RMI programming interface.

The J2EE API is a suitable base for Java across Java platforms, allowing creation of implementations that:

- may be a thin layer on transport protocols
- may support J2EE remote interfaces
- may support J2EE local interfaces

The Java files created with the realisation will be made available with the Parlay/OSA specifications.

The remaining sections of this annex deal with the following areas:

- section C.2 covers the tools and languages used to produce and define the Java Realisation
- section C.3 covers the mappings that are common across both Java Realisation APIs
- section C.4 covers the mappings specific to the J2SE API
- section C.5 covers the mappings specific to the J2EE API

C.2 Tools and languages

The Java language is used as a means to programmatically define the interfaces. Java source files are generated automatically from UML. The Java source files are created in accordance with the mappings defined within this annex.

The generated Java source files are verified syntactically using Java compilers such as javac. The Java API comprises

- J2SE API designed to be compatible with the Java 2 SDK, Standard Edition, version 1.3 (http://java.sun.com/j2se/1.3/) or later and a
- J2EE API compatible with the Java 2 Enterprise Edition (http://java.sun.com/j2ee/).

The J2SE API, developed in accordance to the conventions defined in section C.3 and C.4 will enable:

- portable Java applications, as far as the Java API is concerned
- independence of distribution mechanism technology (e.g. CORBA,SOAP,RMI)

C.3 Generic Mappings (Elements common to J2SE and J2EE)

Note: all Java code examples given in this section are taken from the J2SE Java Realisation API. See the appropriate Java files for examples for J2EE classes.

C.3.1 Namespace

The UML namespace org.csapi is represented by the Java package org.csapi.jr.

Packages under the org.csapi.jr package will contain "se" packages for J2SE specific Java artefacts and "ee" and “ee.remote” packages for J2EE specific Java artefacts.

For example, the User Location Camel Service package structure would appear as follows:

org.csapi.jr.se.mm.ulc containing J2SE API Java artefacts

org.csapi.jr.ee.mm.ulc containing J2EE local API Java artefacts

org.csapi.jr.ee.remote.mm.ulc containing the J2EE remote/RMI API Java artefacts

C.3.2 Package Naming Conventions

UML packages will be represented by Java packages. The sub-namespaces below the root namespaces described above will follow the naming used for the UML namespaces.

C.3.3 Object References

In Java there is no need to explicitly indicate a reference to an object as in Java objects are passed by value and not by reference. Where the specifications explicitly indicate a reference to an object by adding “Ref” to the object type, this addition is removed in the Java realisation.

Example 1:

UML	**Java Realisation**
IpUserLocationCamelRef	UserLocationCamel
IpCallRef	Call

C.3.4 Element Naming

The UML element names that begin with an uppercase will follow the Java naming conventions of with a leading lower case letter and mixed case names. The UML elements are equivalent to Java field names.

Example 2:

UML	**Java Realisation**
AddressPlan	addressPlan

C.3.5 Element Naming Collisions

If an element name collides with a Java keyword, the element name will be prefixed with an underscore.

Example 3:

UML	**Java Realisation**
Final	_final

C.3.5 Data Type Definitions

C.3.5.1 Basic Data Types

Java does not support type definitions (typedefs); therefore types are unwound to their basic data types e.g.:

Example 4:

UML	**Java Realisation**
TpCallAlertingMechanism	int
TpAccessType	java.lang.String

The following mappings apply to the basic data types:

UML	**Java Realisation**
TpBoolean	boolean
TpInt32	int
TpInt64	long
TpFloat	float
TpOctet	byte
TpString	java.lang.String
TpLongString	java.lang.String
TpAny	java.lang.Object

C.3.5.2 Constants

Constants are associated with a type definition or as a standalone entity. In both cases, the constant itself will be defined as a '`public final static`' field using its name and value.

When defined associated with a type definition, an interface using the name of the type definition will be defined enclosing all constants associated with the type definition.

Standalone constants within a package are defined within a Java interface with the name 'Constants' within that package.

Example 5:

```
package org.csapi.jr.se;
public interface Constants {
	public static final int METHOD_NOT_SUPPORTED = 22;
	public static final int NO_CALLBACK_ADDRESS_SET = 17;
	public static final int RESOURCES_UNAVAILABLE = 13;
	public static final int TASK_CANCELLED = 15;
	public static final int TASK_REFUSED = 14;
	public static final int INVALID_STATE = 744;
}
```

Example 6:

```
package org.csapi.jr.se.cc;
public interface CallSuperviseReport {
       public static final int CALL_SUPERVISE_TIMEOUT = 1;
       public static final int CALL_SUPERVISE_CALL_ENDED = 2;
       public static final int CALL_SUPERVISE_TONE_APPLIED = 4;
}
```

C.3.5.3 NumberedSetsOfDataElements (Collections)

In Java, Numbered Set and Numbered List are realised as an array of the data type.

Example 7:

UML	**Java Realisation**
TpAddressSet	Address[]

C.3.5.4 SequenceOfDataElements (Structures)

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

- each data element made available as a private variable in the class
- a default constructor and a constructor for all values are provided
- accessor and mutator methods are given for each variable
- the first letter of each sequence element name is changed to lower case
- an equals method is provided determining the equality of objects by their content
- a hashCode method is provided supporting the rules for hashCode relative to equals

Example 8:

```
package org.csapi.jr.se;
public final class Address implements java.io.Serializable {
       private AddressPlan plan;
       private String addrString = “”;
       private String name = “”;
       private AddressPresentation presentation;
       private AddressScreening screening;
       private String subAddressString = “”;

       public Address () {
       }

       public Address (AddressPlan plan, String addrString,
                    String name, AddressPresentation presentation,
                    AddressScreening screening, String subAddressString) {
          this.plan = plan;
          this.addrString = addrString;
          this.name = name;
          this.presentation = presentation;
          this.screening = screening;
          this.subAddressString = subAddressString;
       }

       public TpAddressPlan getPlan () {
          return (plan);
```

```
        }

        public void setPlan (TpAddressPlan plan) {
           this.plan = plan;
        }

        public String getAddrString () {
           return (addrString);
        }

        public void setAddrString (String addrString) {
           this.addrString = addrString;
        }

        … other get and set methods …

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

        public int hashcode () {
        // hash code calculation
        }
}
```

C.3.5.5 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);


	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 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.3.5.6 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

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;
        }


}
```

C.3.5.7 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.3.5.7.1 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;

        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.3.5.7.2 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;


	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.3.5.7.3 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;
	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 setExceptionType() {
            return exceptionType;
        }

        public String getExtraInformation() {
            return extraInformation;
        }

        public String setExtraInformation() {
            return extraInformation;
        }

}
```

C.3.5.7.4 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 {

        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.3.5.7.5 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;

	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.3.5.7.6 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 {

	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.3.5.7.7 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 {

	public InvalidEnumValueException () {
		super ();
	}
```

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

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

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

}
```

~~C.3.5.7.8 IllegalStateException~~

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

~~Example 18:~~

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

        private int _state;
        private 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;
        }
}
```

C.4 J2SE Specific Conventions

The UML interfaces are represented by Java public interfaces; those interfaces that inherit from other interfaces are represented in Java as extending that interface. The Java realisations of OSA/Parlay SCFs use an Event Listener design pattern while the Framework uses the Callback pattern.

This annex provides the information on realisation of the Java developer API including:

- How Java APIs are realised from Parlay UML
- Where the listener pattern is used, new classes to be generated from the UML
- Changes required to data types and methods to support correlation using object references
- Use of hierarchical exceptions

C.4.1 Removal of "Tp" Prefix

The UML data types labelled with the prefix “Tp” are represented in Java without this prefix.

Example 18:

UML	**Java Realisation**
TpCallAppInfo	CallAppInfo

In the case of name collisions between data types and interfaces as with IpTerminalCapabilities and IpService the UML data types labelled with the prefix “Tp” are represented in Java with an alternative prefix “Type”.

Example 19:

UML	**Java Realisation**
IpTerminalCapabilities	TerminalCapabilities
TpTerminalCapabilities	TypeTerminalCapabilities

The above example is based in conjunction with C.4.3 Removal of "Ip" Prefix.

C.4.2 Constants

The UML constants labelled with the prefix “P_” are represented in Java without this prefix.

Example 20:

UML Constant	**Java Constant**
P_NO_CALLBACK_ADDRESS_SET	NO_CALLBACK_ADDRESS_SET

C.4.3 Removal of "Ip" prefix

The "Ip" prefix is removed in the Java realisation of UML interfaces.

Example 21:

UML	**Java**
IpCallControlManager	CallControlManager

C.4.4 Mapping of IpInterface

IpInterface interface is represented by the CsapiInterface interface. This is a ‘marker’ interface, in that it contains no methods, but provides a common interface for related interfaces to inherit from. All interfaces to be serializable; this can be done by CsapiInterface extending Serializable.

Example 22:

```
package org.csapi.jr.se;
      public interface CsapiInterface extends Serializable{
      }
```

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 {
	void onOutOfService(OutOfServiceEvent event);
 }
```

Event class:

```
package org.csapi.jr.se;
public class OutOfServiceEvent extends EventObject {}
```

C.4.6 Mapping of UML Operations

The UML operations are represented in Java as methods.

Exceptions that can be raised by UML operations are represented in Java with the throws clause and the Java Realisation of the UML Exceptions.

UML "in" parameters, represented by "in " preceding the parameter type are represented in Java without this clause.

Example 24:

```
public void managerResumed ();

public CsapiInterface obtainInterface (InterfaceName interfaceName) throws
InvalidInterfaceNameException;

public Service createServiceManager (ClientAppID application, ServicePropertyList
serviceProperties, ServiceInstanceID serviceInstanceID);
```

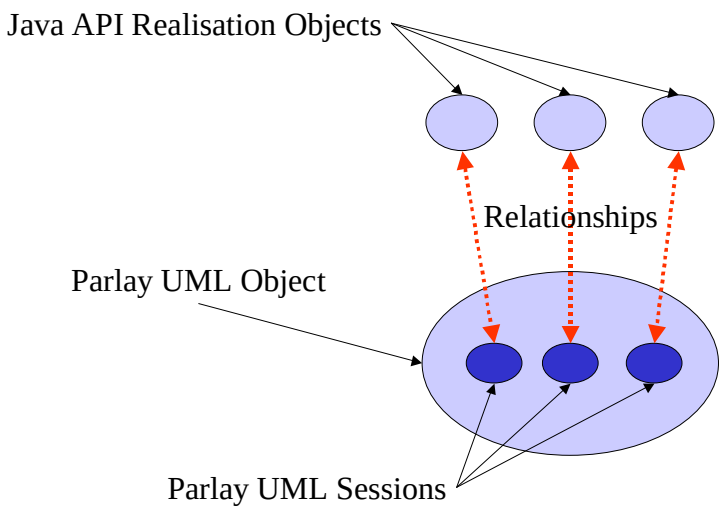
The above example method signatures are based on generic mapping of interfaces, exceptions and data types.

C.4.7 Mapping of TpSessionID

The UML TpSessionID data types will be hidden in the J2SE APIs (and optionally supported by the underlying Java implementation). Consequently, the TpSessionIDSet data type and IpService.setCallbackWithSessionID() method are superfluous. Also, structures with only TpSessionID and interface references (e.g. TpCallIdentifier) are no longer necessary and references to these structures should

be replaced by just the reference to the interface. For data types that contain TpSessionID the Java API Realisation object replaces theTpSessionID.

The following figure shows how Java API Realisation objects relate to Parlay UML objects and sessions. How this is realised in the adaptors is implementation dependent.



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 {
	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 {
	onInvalidState (InvalidActivityEvent event)
}

public class InvalidActivityEvent extends EventObject {
}
```

Parent interface:

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

Child Interface:

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

Listener Interface:

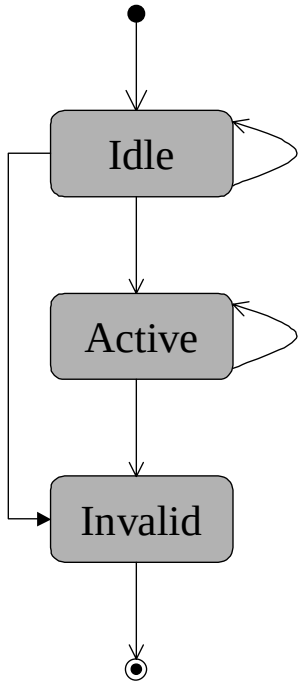
```
package org.csapi.jr.se.mm.ul;
public interface LocationReportListener extends CsapiInterface, java.util.EventListener {

	public void onLocationReportRes(LocationReportResEvent event);
	public void onLocationReportErr(LocationReportErrEvent event);
}
```

Event classes:

```
package org.csapi.jr.se.mmm.ul;
public class LocationReportResEvent extends java.util.EventObject{
```

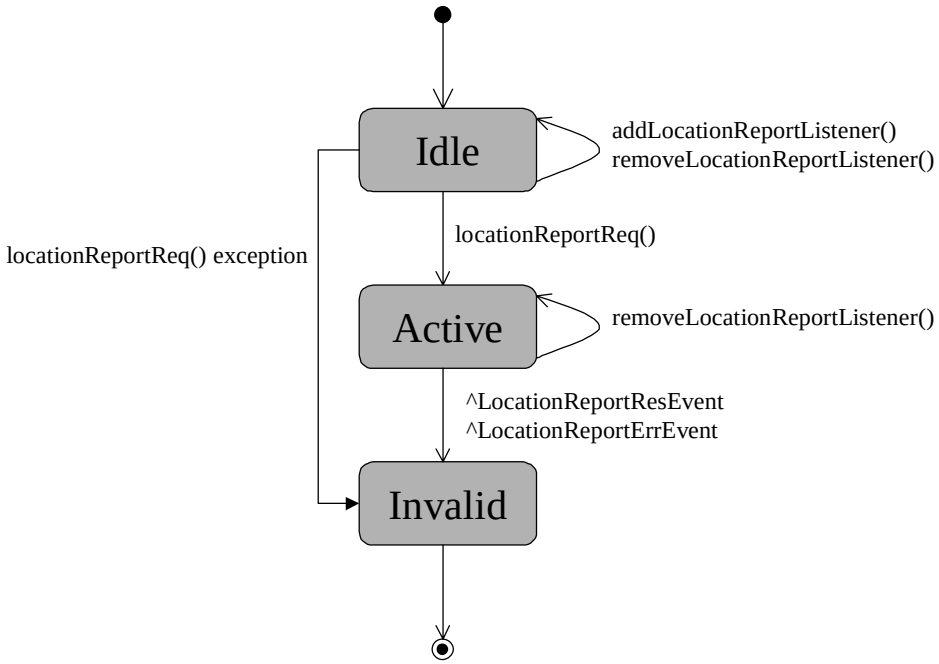
```
        // with a public UserLocation[] constructor and a public getter
        // method for the parameter of the event
}

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:

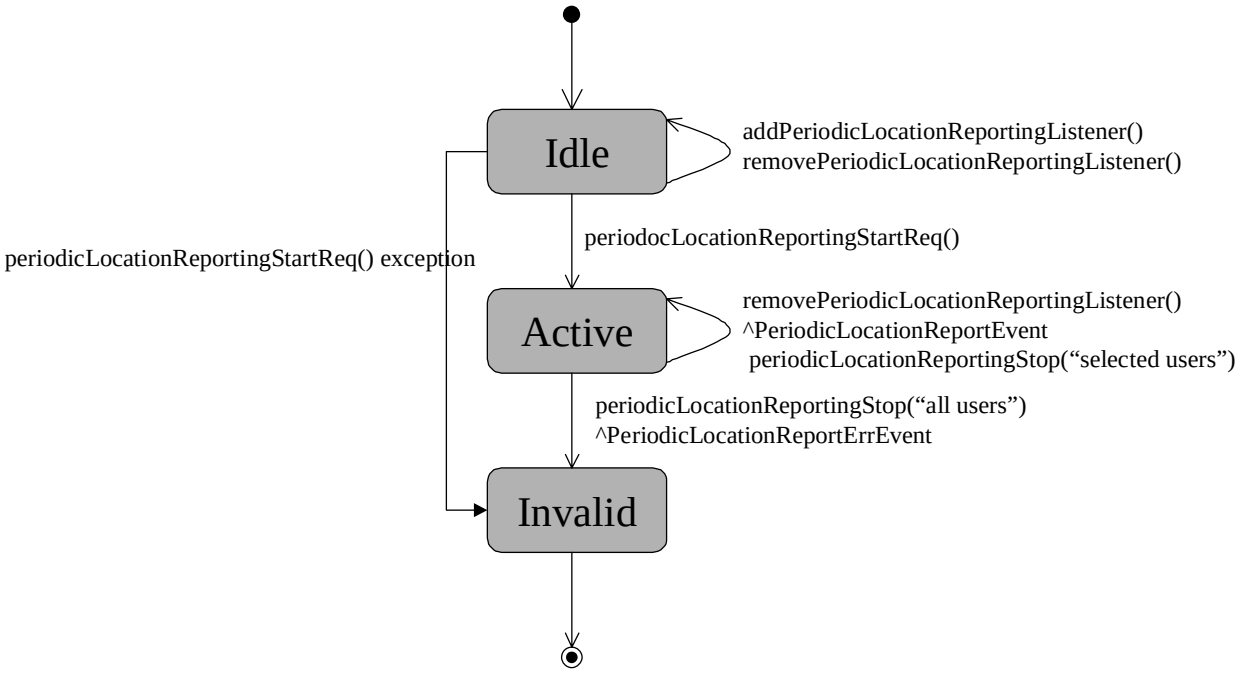


This interface specifies an activity, which might be provided by a service. An activity has three states: "idle", "active" and "invalid". The initial state is "idle" and here the listeners should be registered. It performs in the "active" state. It enters the "invalid" state when it has fulfilled its 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:



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



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 onGetInfoRes(GetInfoResEvent event)
public void onGetInfoErr(GetInfoErrEvent event)
public void onSuperviseRes(SuperviseResEvent event)
public void onSuperviseErr(SuperviseErrEvent event)
public void onCallEnded(CallEndedEvent event)
public void onCreateAndRouteCallLegErr(CreateAndRouteCallLegErrEvent event)
}
```

MuliPartyCall Interface additional methods:

```
public void addMultiPartyCallListener(MultiPartyCallListener multiPartyCallListener);

public void removeMultiPartyCallListener(MultiPartyCallListener multiPartyCallListener);
```

C.4.10 Factory Rule

The following Factory class allows applications to obtain proprietary peer API objects. The term "peer" is Java nomenclature for a particular platform-specific implementation of a Java interface.

Example 27:

```
package org.csapi.jr.se.fw;
import org.csapi.jr.se.PeerUnavailableException;
import org.csapi.jr.se.InvalidArgumentException;
import org.csapi.jr.se.ResourcesUnavailableException;
import org.csapi.jr.se.fw.access.tsm.Initial;
import java.util.*;

public class InitialFactory {
      private static InitialFactory myFactory;
      private static String className = null;
      private static String lang     = "en";
      private static String cntry    = "US";

      private InitialFactory() {
      }

      public synchronized Initial createInitial(String initialPeerReference) throws
PeerUnavailableException, ResourcesUnavailableException , InvalidArgumentException {
```

```
        Locale currentLocale;
        ResourceBundle messages;
        String tryMessage;

        try {
              currentLocale = new Locale(lang, cntry);
              messages = ResourceBundle.getBundle("InitialFactoryBundle",
              currentLocale);

              // Validate all used values before using them later
              // avoiding error text exception to hide the real exception

              tryMessage = messages.getString("InitialPeerReferenceNull");
              tryMessage = messages.getString("InitialInstFailure");
              tryMessage = message.getString("DestroyInitialFailure");
        }
        catch (Exception e) {
              throw new ResourcesUnavailableException ("Localisation failed to be
              initialized");
        }

        if (initialPeerReference == null) {
              String errmsg = messages.getString("InitialPeerReferenceNull");
              throw new InvalidArgumentException (errmsg);
        }

        try {
              Class c = Class.forName (getImplementationClassName ());
              if(initialPeerReference.equals("")){
                    // Creates a new instance of the Object class
                    // using default constructor
                    return (Initial)c.newInstance ();
              }

              Class[] paramTypes = {initialPeerReference.getClass()};
              java.lang.reflect.Constructor ctor =
              c.getConstructor(paramTypes);
              Object[] params = {initialPeerReference};
              return (Initial) ctor.newInstance(params);
        } catch (Exception e) {
              String errmsg = messages.getString("InitialInstFailure");
              throw new PeerUnavailableException (errmsg);
        }
}

public synchronized static InitialFactory getInstance() {
        if (myFactory == null) {
              myFactory = new InitialFactory ();
        }
        return myFactory;
}

public String getImplementationClassName () {
        return className;
}

public static void setImplementationClassName (String className) {
        this.className = className;
}

public synchronized static void setLocale(String language, String country) {
        if (langauage == null) {
              lang = "en";
        }
        else {
              lang = language;
        }

        if (country == null) {
              cntry = "US";
```

```
            }
            else {
                  cntry = country;
            }
      }

      public void destroyInitial(Initial initialInstance) {
            if (initialInstance == null) {
                  return;
            }

            try {
                  delete initialInstance;
            } catch (Exception e) {
                  String errmsg = messages.getString("DestroyInitialFailure");
                  throw new RuntimeException(errmsg);
            }
      }
}
```

C.4.11 J2SE Specific Exceptions

Exceptions in this section are only applicable within a J2SE environment.

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;
      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 Exception {

      private int _state;
      private 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;
    }
}
```

C.4.12 User Interaction Specific Rules

C.4.12.1 Interfaces representing UML IpUI and IpUICall Rule

The following mappings take account of the fact that when the TpAssignmentID rule is applied the Java interfaces representing UML IpUICall does not extend the Java interfaces representing UML IpUI.

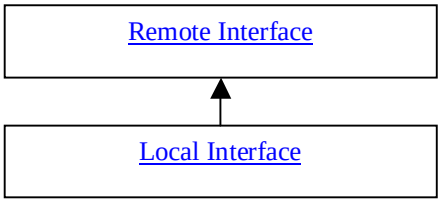
Java UIGeneric replaces the UML IpUI. Methods common to both the Java UIGeneric and Java UICall are pulled up into a super-interface called UI. UML IpAppUI and IpAppUiCall interfaces are replaced by a UIListener interface.

C.4.12.2 Naming Collisions of GUI and CUI Activities Rule

Naming collisions that arise through GUI and CUI activities e.g. XXX, having the same name will be dealt with by prefixing the Call Related UI activity by “CallRelated”. Methods to create the activity will become createCallRelatedXXX().

C.5 J2EE Specific Conventions

J2EE supports both remote and local interfaces. To support one set of data type definitions that will work with both remote and local interfaces, an inheritance approach is used where the remote interface is a super interface to the local interface, supporting Java language rules and accomplishing this goal. This is transparent to the application writer.



C.5.1 Interface Fields in Data Types

A data type as a field must reference the interface as defined in the org.csapi.jr.ee package. This requirement addresses the Java language rule that allows exceptions to be removed in a sub-interface, but does not allow an exception to be added to a sub-interface.

Example ~~27~~30:

```
package org.csapi.jr.ee;

public class TpMultiPartyCallIdentifier {
	org.csapi.jr.ee.remote.IpMultiPartyCall callReference;
	int callSessionID;
}
```

C.5.2 Serialization UID

All data types will have a serialVersionUID defined within its definition, as a static final long value.

Example ~~28~~31:

```
package org.csapi.jr.ee;
public final class TpAddress implements java.io.Serializable {
	static final long serialVersionUID = 989898989898L;

	private TpAddressPlan plan;
	… remainder of class …
}
```

C.5.3 Remote Interface Definitions

C.5.3.1 IpInterface

This interface implements java.io.Serializable. Since it is the root interface for all other interfaces, this makes all defined interfaces serializable.

Example ~~229~~32:

```
public interface IpCall extends IpService
```

C.5.3.3 Methods for Remote Interfaces

A public method is defined within a remote interface for each method defined in the specification, with zero or one output specified as the return value, and all other parameters listed without any input marker. Each method will return java.rmi.RemoteException in addition to other exceptions, if any.

Example ~~30~~33:

```
public void deassignCall (int callSessionID) throws java.rmi.RemoteException,
org.csapi.jr.ee.TpCommonException, org.csapi.jr.ee.InvalidSessionIdException;
```

C.5.4 Local Interface Definitions

C.5.4.1 Parent Interface

Each local interface extends its corresponding remote interface.

Example ~~31~~34:

```
public interface IpCall extends org.csapi.jr.ee.remote.IpCall
```

C.5.4.2 Interface Inheritance

Interfaces in Java may extend each other using the 'extends' keyword. Where an interface is defined as inheriting from one or more other interfaces, it will declare the list of interfaces it inherits from in its interface declaration.

Example ~~31~~35:

```
public interface IpCall extends org.csapi.jr.ee.remote.IpCall, IpService
```

C.5.4.3 Methods for Local Interfaces

A public method is defined within a local interface for each method defined in the specification, with zero or one output specified as the return value, and all other parameters listed without any input marker.

Example ~~32~~36:

```
public void deassignCall (int callSessionID) throws org.csapi.jr.ee.TpCommonExceptions,
org.csapi.jr.ee.InvalidSessionIdException;
```

End of Submission

CR-Form-v7

CHANGE REQUEST

⌘ 29.198-01 CR 022 ⌘rev - ⌘ Current version: 5.1.1 ⌘

*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 ☒

Title: ⌘	Removal of un-used references	
Source: ⌘	N5	
Work item code: ⌘	OSA2	**Date:** ⌘ 26/05/2003
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: ⌘	Most of the references are not used in the body text and hence should be removed.
Summary of change: ⌘	Remove un-used references
Consequences if not approved: ⌘	May generate confusion.

Clauses affected: ⌘	2 References

Other specs affected: ⌘

Y	N	
	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.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document.*

[1] 3GPP TR 21.905: "Vocabulary for 3GPP Specifications".

[2] 3GPP TS 22.127: "Service Requirement for the Open Services Access (OSA)".

[3] 3GPP TS 23.127: "Virtual Home Environment / Open Service Access (OSA)".

~~[4] 3GPP TS 23.078: "Customised Applications for Mobile network Enhanced Logic (CAMEL); Stage 2".~~ NOT used in body text (should be removed)

[5] 3GPP TS 22.101: "Service Aspects; Service Principles".

~~[6] World Wide Web Consortium "Composite Capability/Preference Profiles (CC/PP): A user side framework for content negotiation" (http://www.w3.org/TR/NOTE-CCPP/).~~ NOT used in body text (should be removed)

[7] 3GPP TS 29.002: "Mobile Application Part (MAP) specification".

[8] 3GPP TS 29.078: "Customised Applications for Mobile network Enhanced Logic (CAMEL); CAMEL Application Part (CAP) specification".

~~[9] Wireless Application Protocol (WAP), Version 2.0: "User Agent Profiling Specification" (WAP-248) (http://www.wapforum.org/what/technical.htm).~~ NOT used in body text (should be removed)

~~[10] Wireless Application Protocol (WAP), Version 2.0: "WAP Service Indication Specification" (WAP-167) (http://www.wapforum.org/what/technical.htm).~~ NOT used in body text (should be removed)

~~[11] Wireless Application Protocol (WAP), Version 2.0: "Push Architectural Overview" (WAP-250) (http://www.wapforum.org/what/technical.htm).~~ NOT used in body text (should be removed)

~~[12] Wireless Application Protocol (WAP), Version 2.0: "Wireless Application Protocol Architecture Specification" (WAP-210) (http://www.wapforum.org/what/technical.htm).~~ NOT used in body text (should be removed)

[13] Void.

[14] Void.

[15] Void.

~~[16] 3GPP TS 22.002: "Circuit Bearer Services (BS) supported by a Public Land Mobile Network (PLMN)".~~ NOT used in body text (should be removed)

~~[17] 3GPP TS 22.003: "Circuit Teleservices supported by a Public Land Mobile Network (PLMN)".~~ NOT used in body text (should be removed)

~~[18] 3GPP TS 24.002: "GSM - UMTS Public Land Mobile Network (PLMN) Access Reference Configuration".~~ NOT used in body text (should be removed)

[19] ITU-T Q.763: "Signalling System No. 7 - ISDN user part formats and codes". - NOT used in body text (should be removed)

[20] ITU-T Q.931: "ISDN user-network interface layer 3 specification for basic call control". - NOT used in body text (should be removed)

[21] ISO 8601: "Data elements and interchange formats -- Information interchange -- Representation of dates and times". - NOT used in body text (should be removed)

[22] ISO 4217: "Codes for the representation of currencies and funds". - NOT used in body text (should be removed)

[23] 3GPP TS 22.121: "Service aspects; The Virtual Home Environment; Stage 1". - NOT used in body text (should be removed)

[24] void

[25] void

[26] 3GPP TS 23.057: "Mobile Execution Environment (MExE); Functional Description; Stage 2". - NOT used in body text (should be removed)

[27] void

[28] void

[29] void

[30] void

[31] 3GPP TS 23.271 "Functional stage 2 description of location services (3GPP TS 23.271)". - NOT used in body text (should be removed)

[32] ISO 639: "Code for the representation of names of languages". - NOT used in body text (should be removed)

[33] IETF RFC 822: "Standard for the format of ARPA Internet text messages". NOT used in body text (should be removed)

[34] IETF RFC 1738: "Uniform Resource Locators (URL)". - NOT used in body text (should be removed)

[35] ANSI T1.113: "Signalling System No. 7 (SS7) - Integrated Services Digital Network (ISDN) User Part". - NOT used in body text (should be removed)

[36] IETF RFC 3261: "SIP: Session Initiation Protocol". - NOT used in body text (should be removed)

[37] ITU-T Recommendation Q.932: "Digital subscriber signalling system No. 1 - Generic procedures for the control of ISDN supplementary services". - NOT used in body text (should be removed)

[38] ITU-T Recommendation H.221: "Frame structure for a 64 to 1920 kbit/s channel in audiovisual teleservices". - NOT used in body text (should be removed)

[39] ITU-T Recommendation H.323: "Packet-based multimedia communications systems". - NOT used in body text (should be removed)

[40] IETF RFC 1994: "PPP Challenge Handshake Authentication Protocol (CHAP)". - NOT used in body text (should be removed)

[41] IETF RFC 2630: "Cryptographic Message Syntax". - NOT used in body text (should be removed)

[42] IETF RFC 2313: "PKCS #1: RSA Encryption Version 1.5". - NOT used in body text (should be removed)

[43] IETF RFC 2459: "Internet X.509 Public Key Infrastructure Certificate and CRL Profile". - NOT used in body text (should be removed)

[44] IETF RFC 2437: "PKCS #1: RSA Cryptography Specifications Version 2.0". NOT used in body text (should be removed)

[45] IETF RFC 1321: "The MD5 Message-Digest Algorithm". NOT used in body text (should be removed)

[46] IETF RFC 2404: "The Use of HMAC-SHA-1-96 within ESP and AH". NOT used in body text (should be removed)

[47] IETF RFC 2403: "The Use of HMAC-MD5-96 within ESP and AH". NOT used in body text (should be removed)

[48] ITU-T Recommendation G.722: "7 kHz audio-coding within 64 kbit/s". NOT used in body text (should be removed)

[49] ITU-T Recommendation G.711: "Pulse code modulation (PCM) of voice frequencies". NOT used in body text (should be removed)

[50] ITU-T Recommendation G.723.1: "Speech coders : Dual rate speech coder for multimedia communications transmitting at 5.3 and 6.3 kbit/s". NOT used in body text (should be removed)

[51] ITU-T Recommendation G.728: "Coding of speech at 16 kbit/s using low-delay code excited linear prediction". NOT used in body text (should be removed)

[52] ITU-T Recommendation G.729: "Coding of speech at 8 kbit/s using conjugate-structure algebraic-code-excited linear-prediction (CS-ACELP)". NOT used in body text (should be removed)

[53] ITU-T Recommendation H.261: "Video codec for audiovisual services at p x 64 kbit/s". NOT used in body text (should be removed)

[54] ITU-T Recommendation H.263: "Video coding for low bit rate communication". NOT used in body text (should be removed)

[55] ITU-T Recommendation H.262: "Information technology - Generic coding of moving pictures and associated audio information: Video". NOT used in body text (should be removed)

[56] World Geodetic System 1984 (WGS 84). (http://www.wgs84.com/files/wgsman24.pdf). NOT used in body text (should be removed)

[57] ITU-T Recommendation X.400: "Message handling services: Message handling system and service overview". NOT used in body text (should be removed)

[58] ITU-T Recommendation E.164: " The international public telecommunication numbering plan". NOT used in body text (should be removed)

[59] IETF RFC 2445: "Internet Calendaring and Scheduling Core Object Specification (iCalendar)". NOT used in body text (should be removed)

[60] IETF RFC 2778: "A Model for Presence and Instant Messaging". NOT used in body text (should be removed)