Annex A: Conventions for Reference Points

This annex describes the module naming conventions for IDL modules, and implicitly the module structure. Also, the complete list with defined modules and contained interfaces is presented for reference. Last, some recorded problems with IDL compilation are listed.

A.1 Module Naming Conventions

This part described the naming and structuring conventions used in the definition of the IDL for Ret RP. These conventions do not only apply to Ret, but can also be a guideline for specifications for other reference points.

The naming for the modules, reflecting the separation described in this document, is as follows:

<modulename> := TINA<common> |
TINA<sessionrole><part> |
TINA<reference point><domain><part>

The first set of <modulename> applies to the common parts. The second and third set apply to business role parts. Essentially the third set inherits from the second set and refers to reference point specific business roles.

<common> := CommonTypes |
AccessCommonTypes |
UsageCommonTypes |
StreamCommonTypes
<feature set>Types

<reference point> := Ret |
<other acronyms, e.g. Tcon, Cons, RtR>

<sessionrole> := User | (for access part)
Party | (for usage part)
Provider

<domain> := Consumer |
Retailer

<part> := Initial |
Access |
<usage part>

<usage part>:= <feature set>Usage

<feature set> := Basic
BasicExt
Multiparty
MultipartyInd
Voting
ControlSR
PaSB
PaSBInd
CommS

A.2 Defined Modules and Interfaces

Table 8-1 lists the modules currently defined for Ret, together with the interfaces they contain:

Table 8-1. Modules, Interfaces and dependencies

Module	Contained interfaces	Dependencies
TINACommonTypes	none	none
TINAAccessCommonTypes	none	TINACommonTypes
TINAStreamTypes	none	TINACommonTypes
TINAUserInitial	i_UserInitial	TINAAccessCommonTypes
TINAUserAccess	i_UserAccessGetInterfaces i_UserAccess i_UserInvite i_UserTerminal i_UserAccessSessionInfo i_UserSessionInfo	TINAAccessCommonTypes
TINAProviderInitial	i_ProviderInitial i_ProviderAuthenticate	TINAAccessCommonTypes
TINAProviderAccess	i_ProviderAccessGetInterfaces i_ProviderAccessRegisterInterfaces i_ProviderAccessInterfaces i_ProviderAccess i_ProviderNamedAccess i_ProviderAnonAccess	TINAAccessCommonTypes TINAStreamTypes
TINARetConsumerInitial	i_ConsumerInitial	TINAUserInitial
TINARetConsumerAccess	i_ConsumerAccess i_ConsumerInvite i_ConsumerTerminal i_ConsumerAccessSessionInfo i_ConsumerSessionInfo	TINAUserAccess
TINARetRetailerInitial	i_RetailerInitial i_RetailerAuthenticate	TINAProviderInitial
TINARetRetailerAccess	i_RetailerAccess i_RetailerNamedAccess i_RetailerAnonAccess i_DiscoverServicesIterator	TINAProviderAccess
TINAUsageCommonTypes	none	TINACommonTypes
TINAProviderBasicUsage	i_ProviderBasicReq	TINAUsageCommonTypes TINASessionModel
TINAPartyBasicExtUsage	i_PartyBasicExtReq	TINAUsageCommonTypes TINASessionModel
TINAPartyMultipartyUsage	i_PartyMultipartyExe i_PartyMultipartyInfo (optional)	TINAUsageCommonTypes
TINAProviderMultipartyUsage	i_ProviderMultipartyReq	TINAUsageCommonTypes

Table 8-1. Modules, Interfaces and dependencies

Module	Contained interfaces	Dependencies
TINAPartyMultipartyIndUsage	i_PartyMultipartyInd	TINAUsageCommonTypes
TINAPartyVotingUsage	i_PartyVotingInfo	TINAUsageCommonTypes
TINAProviderVotingUsage	i_ProviderVotingReq	TINAUsageCommonTypes
TINAControlSRTypes	none	TINAUsageCommonTypes
TINAPartyControlSRUsage	i_PartyControlSRInd i_PartyControlSRInfo	TINAUsageCommonTypes TINAControlSRTypes
TINAProviderControlSRUsage	i_ProviderControlSRReq	TINAUsageCommonTypes TINAControlSRTypes
TINASBCommSCommonTypes	none	none
TINAStreamCommonTypes	none	TINACommonTypes TINASBCommSCommonTypes
TINAPaSBTypes	none	TINAUsageCommonTypes TINAStreamCommonTypes
TINAPartyPaSBUsage	i_PartyPaSBExe i_GeneralStreamInfo i_PartyGeneralStreamInfo i_PartyPaSBInfo	TINAPaSBTypes
TINAProviderPaSBUsage	i_ProviderPaSBReq	TINAPaSBTypes
TINAPartyPaSBIndUsage	i_PartyPaSBInd	TINAPaSBTypes
TINACommSCommonTypes	none	TINASBCommSCommonTypes TINAConSCommSCommonTypes
TINAConSCommSCommonTypes	none	TINASBCommSCommonTypes
TINAPartyCommSUsage	i_BasicTerminalFlowControl i_TerminalFlowControl	TINASBCommSCommonTypes TINAConSCommSCommonTypes TINACommSCommonTypes

A.3 Recorded problems with IDL

In the process of defining, writing and compiling the IDLs, some problems have been recorded that have had their impact on how some of the IDL is defined:

1. In order to work with OrbixWeb (current mapping) all IDL interfaces or types should be scoped within a MODULE.
2. In order to work with NEO, all IDL files should NOT start with a comment on the first line
3. In order to work with all ORBs, any types or interfaces can only be forward declared with the file in which they are later defined. i.e.forward declaration of an interface which is properly defined in a different file is not acceptable.
4. Typedefs of Object (i.e. CORBA::Object) should not be used.
5. Preprocessor commands (e.g. #define #ifndef etc) should not be terminated with comments. e.g.

```
#define i_retailerInitial_idl // Retailer Initial
```

6. Structures or exceptions should not contain submembers called type. e.g

```
struct aStruct {
       short type;
       boolean another;
};
```

7. All IDL files should end with an endline. i.e. after the last comment or preprocessor command, there should be an empty line.

8. The code for arrays of octets is not properly generated using Orbix. It's a known bug in Orbix. workaround: sequences

9. The typedefs of atomic types are not generated using the java mapping. This explains why in the application code some TINA types cannot be used.

10. A sequence definition using a scoped element compiles without problems with the Orbix IDL compiler. However, the generated C++ code is not compilable. Work-around: Always define sequences in the same module where the element type is defined. Example:

```
// Generated C++ does not compile:

// File A.idl
module A {
      typedef string t_string;
};

// File B.idl
#include "A.idl"

module B {
      typedef sequence<A::t_string> t_stringList;
};
```

11. In order to work with idldoc (an IDL to HTML document processor), unions must have a tag associated with each case body. Two or more tags cannot be associated with a single body. (All other IDL compilers, and the CORBA specification allow this. However Orbix doesn't allow the discriminator (switch tag) to be set. It can only be 'inferred' by setting the case body variable, and so Orbix doesn't work with multiple tags either.)

```
// Does not compile with idldoc:
union t_union switch (t_discriminator) {
      case okayTag: short okayCase;
      case firstCase:
      case secondCase: octet bodyForBoth;
};

// Does compile with idldoc:
union t_union switch (t_discriminator) {
      case okayTag: short okayCase;
      case firstCase: octet bodyForFirstCase;
      case secondCase: octet bodyForSecondCase;
};
```

Annex B: Common IDL

This module defines all common types over access, usage and stream idls.

B.1 FILE: README .edits

This file contains an indexed list of all the edits to the IDL files since the IDL first became ‘stable’ in May 1997.

The edits are listed in Last-in First-out order. i.e. the latest edits are at the top of the file. The edits are indexed by 000 - 999 (053) at time of writing. These edits can be found in the IDL files by searching for: $$000$$. The actual edit is sometime also described at the place of editting, where it’s easier to understand what was done, with the IDL present.

The edits have also been divided in sections to show which changes have been made with each version of the document. This has only been done since v0.7. So if you are migrating from a previous version, then it going to be more difficult to determine the changes that have been made. Sorry.

```
// FILE: README.edits
//
//  Log of edits made to idl-stable directory and files, after issue of
//   idl on 28 March 97.
//
// To find edits in files look for $$editNum$$
//    e.g. to find edit 001 in file access/i_ConsumerAccessSessionInfo.idl
//             search for $$001$$
//
// major change - change to semantics of operation/structure
//                      (includes adding/deleting an operation/parameter)
//
// minor change - anything that does not affect semantics of any operation
//                      typographical error,
//                      move of type from one module to another, where a typedef
//                                is placed in original module.

// Add latest edits to top of file

// changes incorporated in v1.0 below here

057   97/1/15   Patrick Farley     major change
      File TINAPartyCommSUsage.idl
      Found some inconsistencies between the document and IDL.
      Have updated the IDL to be consistent with the document.
      (changes have been recorded at the edit marker where significant)

056   97/1/15   Patrick Farley     major change
      file: TINAPartyPaSBUsage.idl
      changed i_PartyPaSBInfo to inherit from i_PartyGeneralStreamInfo
      instead of i_GeneralStreamInfo (as the documentation for i_PartyPaSBInfo
      includes extra ops in i_PartyGeneralStreamInfo)

055   97/1/15   Patrick Farley     minor change
      File: TINAProviderPaSBUsage.idl TINAProviderPaSBUsage.idl
```

```
                TINAPartyPaSBIndUsage.idl
      renamed modifyCritieriaProviderPaSBReq to modifyCriteriaProviderPaSBReq
      (to remove extra i in Critieria) Similar spelling mistakes also changed
      all marked with edit tag. eg. InsuffisicentBandwidth => Insufficient

054   97/1/15   Patrick Farley     minor change
      FILE TINAPartyCommSUsage.idl or TINAPartyCommS.idl
      The file TINAPartyCommS.idl was documented as ebing called
      TINAPartyCommSUsage.idl, and contained the module TINAPartyCommSUsage
      So renamed the file TINAPartyCommS.idl to TINAPartyCommSUsage.idl
      (TINAPartyCommS.idl no longer exists.)

053   97/1/15   Patrick Farley     minor change
      File: TINASBCommSCommonTypes.idl
      deleted from top of file. Unneccessary.
      #ifdef defined(PLATyTools) && defined(debug)
      #include "PLATyToolsFix.idl"
      #endif

052   97/1/15   Patrick Farley     minor change
      File TINAAccessCommonTypes.idl
      changed t_SpecifiedAccessSession to include 2 dummy cases, with
      2 dummy variable, instead of just 1. (All IDL compilers were fine with
      previous version, but idldoc (IDL to HTML converter didn't like it.)
      (CORBA spec allows cases to 'run through' several case headers to one
      case body, but I thought it would be better to cope with idldoc's
      problems.

051   97/1/7    Patrick Farley     major change
      File TINAAccessCommonTypes.idl, TINAProviderAccess.idl
      moved t_ApplicationInfo, t_StartServiceUAProperties,
      t_StartServiceSSProperties to TINAAccessCommonTypes
      and created typedefs for them in TINAProviderAccess.idl


// Changes incorporated in v0.9 below.

050   97/12/15  Patrick Farley     major change
      File TINACommonTypes.idl,
      changed t_SessionModelName name: defined names "TINASessionModel"
      to "TINAServiceSessionModel"
      and added a new name "TINACommSessionModel"
      Both session models can have feature sets as properties.

049   97/12/02  Richard Westergaminor change
      Two new sequences added in TINAAccessCommonTypes.idl
      t_ServiceIdList and t_NAPIdList, both needed in
      TINASubCommonTypes.idl of the service components idl
      Defining these sequences in TINASubCommonTypes.idl as
      typedef sequence<TINAAccessCommonTypes::t_ServiceId> t_ServiceIdList
      causes the idl compiler to generate uncompilable c++ code. Note: the
      idl compiler does *not* complain.
```

```
048   97/12/02  Richard Westergamajor change
      Removed nested module CosTrading from TINACommonTypes.idl
      Changed all CosTrading:: scopes to basic types

// Changes incorporated in v0.8 below.

047   97/09/25  Patrick Farley.    major change
      FILE:     TINAPartyMultipartyUsage.idl
      added userDetails to joinSessionInfo() so can tell 'who' has joined.

046   97/09/25  Patrick Farley.    major change
      FILE:     TINAPartyMultipartyIndUsage.idl
      removed reqPartyId, and e_PartyError from joinSessionInd
      (When joining a session, the user joining is not a party until
      the Info is sent. So only the user details are sent from now on.

045   97/09/10  Patrick Farley.    minor change
      File TINAUserAccess.idl
      moved t_CancelAccessSessionProperties from inside interface i_UserAccess
      to just inside module TINAUserAccess

044   97/09/08  Patrick Farley.    major change
      FILE: TINACommonTypes.idl & TINASessionModels.idl
      moved t_FeatureSetList to TINACommonTypes.idl, and deleted all
      types from TINASessionModels.idl

043   97/09/08  Patrick Farley.    major change
      FILE: TINACommonTypes.idl & TINAUsageCommonTypes.idl
      moved t_UserDetailsErrorCode & e_UserDetailsError
      to TINACommonTypes.idl

042   97/09/05  Patrick Farley.    major change
      FILE: TINAProviderAccess.idl
      moved all types defined inside interfaces to just inside the module.

// changes to draft0.8 above this
// Changes incorporated in v0.7 below.

041   97/08/13  Patrick Farley.    major change
      TINACommonTypes.idl added some new types for usage part

040   97/08/13  Patrick Farley.    major change
      changed typedef TINACommonTypes::t_ElementId t_SIIdList
      to typedef sequence<t_SIId> t_SIIdList
      which is basically the same thing, but allows t_SIId to be changed
      which having to remember to change t_SIIdList

// Started doing the usage IDL after this, but none of the usage changes are
//    documented until they are stable.

039   97/08/13  Patrick Farley.    major change
```

```
      FILE: TINAProviderAccess.idl TINACommonTypes.idl
               TINAAccessCommonTypes.idl
      added new ops getUserCtxt() and getUserCtxts()  and
      getUserCtxtsAccessSessions() to i_ProviderAccess
      (and some bits to e_UserCtxtErrorCodes)

038   97/08/04  Richard Westergamajor change
      FILE TINAProviderInitial.idl
      changed e_AuthenticationError { t_Reference } to be
      e_AutenthicationError {TINACommonTypes::Istring }
      Removed t_Reference union type from TINACommonTypes.idl

037   97/08/01  Patrick Farley.    minor change
      FILE: TINACommonTypes.idl
      added typedef sequence<t_SessionId> t_SessionIdList
      FILE: from TINAProviderAccess.idl
      changed e_EndAccessSessionError t_SessionList to t_SessionIdList

036   97/08/01  Patrick Farley.    minor change
      FILE: from TINAAccessCommonTypes.idl
      to union t_SpecifiedAccessSession, added new case types for all the
      t_WhichAccessSession discriminator enums, with a dummy variable
      so the idl works.

035   97/07/26  Patrick Farley.    minor change
      FILE:  TINAProviderAccess.idl
      added desiredProperties parameter and e_PropertyError exception
      to getInterfaces()

034   97/07/24  Patrick Farley.    minor change
      moved all the types to outside the interface, just inside the module
      otherwise in implementation have to have 2 levels of scoping.

033   97/07/24  Patrick Farley.    major change
      FILE: from TINAProviderAccess.idl to TINACommonTypes.idl
      moved t_InvitationReplyErrorCode and e_InvitationReplyError


032   97/07/24  Patrick Farley.    minor change
      FILE:  TINAProviderAccess.idl
      added InvalidApplication error code to t_ApplicationInfoErrorCode
      and SessionDoesNotExist error code to t_SessionErrorCode

031   97/07/24  Patrick Farley.    major change
      FILE: TINACommonTypes.idl
      new t_Reference union for object ref and StringIOR
      and changed e_AuthenticationError { Object authIR; };
      to be e_AuthenticationError { t_Reference authIR; };

030   97/07/24  Patrick Farley.    major change
      FILE:  TINAProviderAccess.idl
      added some new error codes to t_ServiceErrorCode
```

```
      and replace e_PropertyError with 2 exceptions
      e_StartServiceUAPropertyError, e_StartServiceSSPropertyError
      to match the parameters.

029   97/07/24  Patrick Farley.    major change
      FILE: from TINARetRetailerAccess.idl to TINAProviderAccess.idl
      moved definition of i_DiscoverServicesIterator as it will be
      reused by all refpoints implementing TINAProviderAccess.idl

028   97/07/24  Patrick Farley.    minor change
      FILE: TINAProviderAccess.idl
      removed TINACommonTypes::e_ListError from getSessionInterface()
      as no list is returned!!

027   97/07/24  Patrick Farley.    minor change
      FILE: TINAProviderAccess.idl
      added exception e_SpecifiedAccessSessionError to endAccessSession()

026   97/07/24  Patrick Farley.    minor change
      FILE: TINAProviderAccess.idl
      added t_PropertyErrorStruct to e_UserCtxtError to allow errors
      in terminal properties to be identified

025   97/07/24  Patrick Farley.    minor change
      FILE: TINAProviderAccess.idl
      changed t_AnnouncementProperties to t_AnnouncementSearchProperties
      in op listSessionAnnouncements (), and made new type:
      typedef t_MatchProperties t_AnnouncementSearchProperties
      so can match some all none of the properties.

024   97/07/23  Patrick Farley.    minor change
      FILE: TINAProviderInitial.idl
      changed authenticationStatus to t_AuthenticationStatus for consistency
      and authStat to authStatus

023   97/07/23  Patrick Farley.    minor change
      removed t_AuthMethodDescList from the exceptions
      e_AuthMethodNotSupported and e_AuthenticationError as they caused
      problems for OrbixWEB (prob due to sequences in sequences in exceptions)

022   97/07/23  Patrick Farley.    minor change
      FILE: TINACommonTypes.idl
      changed property to propertyError to be consistent

021   97/07/23  Patrick Farley.    minor change
      FILE: TINAProviderInitial.idl
      Added e_AuthMethodPropertiesError &  TINACommonTypes::e_ListError
      exceptions to getAuthenticationMethods()
      and typedef t_MatchProperties t_AuthMethodSearchProperties;
      and change t_AuthMethodProperties to t_AuthMethodSearchProperties
      and properties to desiredProperties for getAuthenticationMethods()
```

```
020   97/07/21  Patrick Farley.    minor change
      FILE: TINACommonTypes.idl
      Added comment for SecurityContext property name to t_UserProperties

019   97/07/16  Patrick Farley.    major change
      FILE: TINAProviderAccess.idl
      Added e_SessionError exception to joinSessionWithInvitation() and
      joinSessionWithAnnouncement(), so retailer can refuse joining session
      because it is suspended, ended, session refused, op not supported, etc.

018   97/07/16  Patrick Farley.    major change
      FILE: TINAProviderInitial.idl
      Added e_AccessNotPossible exception to requestNamedAccess() and
      requestAnonymousAccess(), so retailer can deny access without
      giving an authenticate interface.

017   97/07/16  Patrick Farley.    major change
      FILE: TINAProviderInitial.idl
      Added TINAAccessCommonTypes::e_UserPropertiesError to
      requestAnonymousAccess()

016   97/07/14  Richard WestergaMinor
      Moved t_AnnouncementList typedef from accessCommonTypes
      to CommonTypes


## New module and file structure adopted above this point. Filenames below
      no longer exist, but mostly refer to the appropriate interface.

015   97/06/25  Patrick Farley.    major change
      changed all types that identified a specific interface type to
      use (CORBA::) Object instead.
      Otherwise the foward references get too complex, and some components
      will have to include the complete definition fo interfaces that
      they just pass around, and don't use themselves.


014   97/06/24  Patrick Farley.    major change
      FILE: commonTypes.idl, access/i_ProviderAccessRegisterInterfaces.idl
      added: registerInterfaces(), registerInterfacesOutsideAccessSession()
      and unregisterInterfaces() to allow the consumer to register and
      unregister multiple interfaces at once.

013   97/06/24  Patrick Farley.    major change
      FILE: commonTypes.idl, access/i_ProviderAccessGetInterfaces.idl
                access/i_ProviderNamedAccess.idl, streamTypes.idl
      changed the identifier 'type' to 'Type' + something appropriate
      e.g. an interface type name goes from 'type' to 'itfType'

012   97/06/23  Patrick Farley.    major change
      FILE: access/i_UserAccessSessionInfo.idl
      added newSubscribedServicesInfo() to i_UserAccessSessionInfo
```

```
      to allow the user to be informed of new services they have been
      subscribed to (thru another access session, or this one)
      FILE: access/i_ProviderNamedAccess.idl & access/accessCommonTypes.idl
      moved types associated with t_ServiceList from i_ProviderNamedAccess
      to accessCommonTypes.idl, so they can be used in the info above.

011   97/06/23  Patrick Farley.    major change
      FILE: access/i_UserAccess.idl
      added cancelAccessSession() to allow the retailer to cancel
      the current access session.
      FILE: access/i_UserAccessSessionInfo.idl
      added cancelAccessSessionInfo() to inform other access sessions of the
      cancelled access session.

010   97/06/20  Patrick Farley.    minor change
      FILE: streamTypes.idl
      changed struct t_SFEPComDesc {
                i_TCSMCoord coordIfRef; // Associated i_TCSMCoord interface
      to be  struct t_SFEPComDesc {
                t_Interface coordIfRef; // Associated i_TCSMCoord interface
      so didn't have to forward ref i_TCSMCoord, (which isn't defined yet).

009   97/06/20  Patrick Farley.    major change
      FILE: commonTypes.idl
      Changed t_ParticipantSecretId from octet array 16 to
      bounded sequence of octet (16).

008   97/06/17  Richard S. Westerga minor change
      All applicable files: changed forward declaration of interfaces
      to #include statements.

007   97/06/17  Richard S. Westergaminor change
      All applicable files: removed typedefs and scoped parameters/membertypes
      accordingly.

006   97/06/03  Patrick Farley.    major change
      FILE: access/accessCommonTypes.idl and access/i_ProviderNamedAccess.idl
                access/i_UserInitial.idl, access/i_UserInvite.idl
      moved e_InvitationError and t_InvitationErrorCode to accessCommonTypes
      so they can be reused in the i_UserInitial and i_UserInvite interfaces
      for cancelling invitations.

005   97/06/03  Patrick Farley.    major change
      FILE: access/accessCommonTypes.idl
      Added t_UserId inviteeId; to struct t_SessionInvitation {
      so if you receive an invitation (esp outside of an access session)
      you can tell if the invitation was for you. (or who it was for
      if a single interface is registered to receive invitations for

004   97/06/03  Patrick Farley.    major change
      FILE: i_UserInitial.idl, i_UserInvite.idl, accessCommonTypes.idl
      added operations to allow an invitation to be cancelled.
```

```
003   97/06/03  Patrick Farley.    minor change
      FILE: commonTypes.idl & access/accessCommonTypes.idl
      moved t_InvitationReply struct and t_InvitationReplyCodes from
      access/accessCommonTypes.idl to commonTypes.idl, so they can
      be reused in the usage part.

002   97/06/03  Patrick Farley.    major change
      FILE: access/i_UserInitial.idl
      added t_AccessReply, and out reply to requestAccess() to allow
      user to tell provider if they are going to accept their request to
      establish an access session.

001   97/06/02  Patrick Farley.    minor change
      FILE: access/i_ConsumerAccessSessionInfo.idl
      changed   interface i_ConsumerAccesssessioninfo
      to        interface i_ConsumerAccessSessionInfo
```

B.2 Module TINACommonTypes

```
// File TINACommonTypes.idl

#ifndef tinacommontypes_idl
#define tinacommontypes_idl

module TINACommonTypes {


// ElementIds (mainly used in usage part)
//

// Element types
enum t_ElTypes {
      TinaParty,                    // see also t_PartyId
      TinaResource, TinaMember,
      TinaGroup, TinaMemberGroup, TinaPartyGroup, TinaResourceGroup,
      TinaRelation, TinaRelationGroup, TinaControlRelation,
      TinaStreamBinding, TinaStreamFlow, TinaStreamInterface, TinaSFEP
};

// Element Identifier (elements in a service session)
typedef unsigned long t_ElId;

// Element Type Identifiers
typedef t_ElTypes t_ElTypeId;

// Overall element Identifier
struct t_ElementId
{
      t_ElId id;
      t_ElTypeId elType;// $$013$$ changed type to elType
};

typedef sequence <t_ElementId> t_ElementIdList;

// t_PropertyList
//    list of properties, (name value pairs).
//    Used in many operations to allow a list of as yet undefined
//    properties, and values, to be sent.
//

// $$048$$

typedef string t_PropertyName;
typedef sequence<t_PropertyName> t_PropertyNameList;
typedef any t_PropertyValue;

struct t_Property {
      t_PropertyName name;
      t_PropertyValue value;
};
```

```
typedef sequence<t_Property> t_PropertyList;

enum t_HowManyProps {none, some, all};

union t_SpecifiedProps switch (t_HowManyProps) {
      case some: t_PropertyNameList prop_names;
      case none:
      case all: octet dummy;// See $$036$$
};

typedef string Istring;

// $$038$$

// $$031$$ new:
// enum t_ReferenceSort {
//    ObjectRef,
//    StringifiedReference
// };

// $$031$$ new:
// union t_Reference switch(t_ReferenceSort) {
//    case ObjectRef: Object IRef;
//    case StringifiedReference: TINACommonTypes::Istring IORef;
//
// };

enum t_WhichProperties {
      NoProperties,      // don't can ignore all the properties
      SomeProperties,    // match at least one property (name & value)
      SomePropertiesNamesOnly,// check name only (ignore value)
      AllProperties,     // match all properties (name & value)
      AllPropertiesNamesOnly// check name only (ignore value)
};

struct t_MatchProperties {
      t_WhichProperties whichProperties;
      t_PropertyList properties;
};

typedef /*CORBA::*/Object t_Interface;

typedef string t_InterfaceTypeName;
typedef sequence<t_InterfaceTypeName> t_InterfaceTypeList;
typedef t_PropertyList t_InterfaceProperties;

struct t_InterfaceStruct {
        t_InterfaceTypeName itfType; // $$013$$ changed type to itfType
        Object ref;
               // if NULL: use getInterface(type)
               // to get the reference
```

```
        t_InterfaceProperties properties;
               // interface type specific properties
               // interpreted by the session.
};

typedef sequence<t_InterfaceStruct> t_InterfaceList;

typedef unsigned long t_InterfaceIndex;
typedef sequence<t_InterfaceIndex> t_InterfaceIndexList;

// $$014$$ new types
// when registering multiple interfaces need to match index vs itfType & props:
struct t_RegisterInterfaceStruct {
        t_InterfaceTypeName itfType;// set before call to registerInterfaces
        t_InterfaceProperties properties;// as above
      t_InterfaceIndex index;// set on return
};

typedef sequence<t_RegisterInterfaceStruct> t_RegisterInterfaceList;
// $$014$$ end of new types

//
// Session Models
//

// t_SessionModelName name:
// defined names
//    "TINAServiceSessionModel"TINA Service Session Model  // $$050$$
//    "TINACommSessionModel"TINA Communication Session Model
// No other names defined at present
//
// (previous versions of Ret-RP used "TINASessionModel" and "TINA Session Model"
//    for the TINA Service Session Model (previously named TINA Session Model)


typedef string t_SessionModelName;

typedef sequence<t_SessionModelName> t_SessionModelNameList;

// t_SessionModelProperties properties:
//
// t_SessionModelName name: "TINAServiceSessionModel"
// defined Property names:
// name:       "FEATURE SETS"
// value:      t_FeatureSetList
// No other names defined at present

// $$050$$
// t_SessionModelName name: "TINACommSessionModel"
// defined Property names:
// name:       "FEATURE SETS"
// value:      t_FeatureSetList
```

```
//
// No feature sets are defined for the TIACommSessionModel at present.
//
// No other names defined at present


typedef t_PropertyList t_SessionModelProperties;

struct t_SessionModel {
      t_SessionModelName name;// reserved names defined below
      t_SessionModelProperties properties;// properties defined above
};


typedef sequence<t_SessionModel> t_SessionModelList;

enum t_WhichSessionModels {
      NoSessionModels,// can ignore all the SessionModels
      SomeSessionModels,// match at least one SessionModel (name & value)
      SomeSessionModelsNamesOnly,// check name only (ignore value)
      AllSessionModels,// match all SessionModels (name & value)
      AllSessionModelsNamesOnly// check name only (ignore value)
};

struct t_SessionModelReq {
      t_WhichSessionModels which;
      t_SessionModelList sessionModels;
};

typedef string t_FeatureSetName;

struct t_FeatureSet {
      t_FeatureSetName name;
      t_InterfaceList itfs;
                                                        // can return ref or NULL
};

typedef sequence<t_FeatureSet> t_FeatureSetList;// $$044$$ added


//
// User Info
//

typedef Istring t_UserId;
typedef Istring t_UserName;
typedef t_Property t_UserProperty;
typedef t_PropertyList t_UserProperties;
// Property Names defined for t_UserProperties:// $$020$$
// name:"PASSWORD"
// value:string
// use:        user password, as a string.
```

```
// name:"SecurityContext"
// value:opaque
// use: to carry a retailer specific security context
//             e.g. could be used for an encoded user password.

struct t_UserDetails {
      t_UserId id;
      t_UserProperties properties;
};

typedef Istring t_UserCtxtName;
typedef sequence<t_UserCtxtName> t_UserCtxtNameList;// $$039$$

typedef sequence<octet, 16> t_ParticipantSecretId;// $$009$$
typedef t_ElId t_PartyId;// corresponds to TinaParty enum t_ElTypes
typedef sequence<t_PartyId> t_PartyIdList;// $$041$$

// SessionId
//    A SessionId is generated by a UA when a new session is started.
//    This Id is unique within a UA, and can be used to identify a
//    session to the UA.
//    User's joining a session will have a different SessionId generated
//    by their UA for the session.

typedef unsigned long t_SessionId;
typedef sequence<t_SessionId> t_SessionIdList;// $$037$$
typedef t_PropertyList t_SessionProperties;


// Invitation and Announcements
//
// $$003$$
enum t_InvitationReplyCodes {// Based on MMUSIC replys
      SUCCESS,                    // user agrees to participate
      UNSUCCESSFUL,               // couldn't contact user
      DECLINE,                    // user declines
      UNKNOWN,                    // user is unknown
      ERROR,                            // for some unknown reason
      FORBIDDEN,                  // authorisation failure
      RINGING,                    // user is being contacted
      TRYING,                           // some further action is being
taken
      STORED,                           // invitation is stored
      REDIRECT,                   // try this different address
      NEGOTIATE,                  // alternatives described in properties
                           // Not MMUSIC replys, can be treated as UNSUCCESSFUL
      BUSY,                             // couldn't contact because busy
      TIMEOUT                           // timed out while trying to contact
};

typedef t_PropertyList t_InvitationReplyProperties;
```

```
struct t_InvitationReply {
      t_InvitationReplyCodes reply;
      t_InvitationReplyProperties properties;
};
// $$003$$ - end of included types

typedef t_PropertyList t_AnnouncementProperties;

struct t_SessionAnnouncement {
      t_AnnouncementProperties properties;
};

typedef sequence<t_SessionAnnouncement> t_AnnouncementList;   // $$016$$

//
// Exceptions
//
enum t_PropertyErrorCode {
      UnknownPropertyError,
      InvalidProperty,
      // UnknownPropertyName: If the server receives a property name
      // it doesnot know, it can raise an exception, using this code.
      // However, servers may decide to ignore a property with an
      // unknown property name, and not raise an exception.
      UnknownPropertyName,
      InvalidPropertyName,
      InvalidPropertyValue,
      NoPropertyError    // the Property is not in error
};

// defined so it can be used in other exceptions
struct t_PropertyErrorStruct {
      t_PropertyErrorCode errorCode;
      t_PropertyName name;
      t_PropertyValue value;
};

exception e_PropertyError {
      t_PropertyErrorCode errorCode;
      t_PropertyName name;
      t_PropertyValue value;
};

enum t_InterfacesErrorCode {
      UnknownInterfacesError,
      InvalidInterfaceTypeName,// Thats not a valid i/f type name
      InvalidInterfaceRef,
      InvalidInterfaceProperty,
      InvalidInterfaceIndex
};
```

```
// must remain consistent with e_InterfacesError
struct t_InterfacesErrorStruct {
      t_InterfacesErrorCode errorCode;
      t_InterfaceTypeName itfType;// $$013$$ changed type to itfType
      t_PropertyErrorStruct propertyError;// $$022$$
               //PropertyError, if errorCode= InvalidInterfaceProperty
};

exception e_InterfacesError {
      t_InterfacesErrorCode errorCode;
      t_InterfaceTypeName itfType;// $$013$$ changed type to itfType
      t_PropertyErrorStruct propertyError;// $$022$$
               //PropertyError, if errorCode= InvalidInterfaceProperty

};

enum t_RegisterErrorCode {
      UnableToRegisterInterfaceType
};

exception e_RegisterError {
      t_RegisterErrorCode errorCode;
      t_InterfaceTypeName itfType;// $$013$$ changed type to itfType
      t_InterfaceProperties properties;// $$014$$
};

enum t_UnregisterErrorCode {
      UnableToUnregisterInterface
};

exception e_UnregisterError {
      t_UnregisterErrorCode errorCode;
      t_InterfaceIndexList indexes; // $$014$$ can unregister multiple itfs
};


enum t_SessionModelErrorCode {
      UnknownSessionModelError,
      InvalidSessionModelName,// Thats not a valid i/f type name
      SessionModelNotSupported,
      InvalidFeatureSetName,
      FeatureSetNotSupported,
      InvalidFeatureSetInterfaceType
};

exception e_SessionModelError {
      t_SessionModelErrorCode errorCode;
      t_SessionModelName sessionModelName; // $$013$$
      t_FeatureSetName featureSetName;// Only for FeatureSet errs // $$013$$
      t_InterfaceTypeName itfType;// Only for FeatureSet errs // $$013$$
};
```

```
// $$043$$ moved here from TINAUsageCommonTypes.idl
enum t_UserDetailsErrorCode {
      InvalidUserName,
      InvalidUserProperty
};

exception e_UserDetailsError {
      t_UserDetailsErrorCode errorCode;
      t_UserName name;
      t_PropertyErrorStruct propertyError;
                                          // Return the properties in error
};


enum t_ListErrorCode {
      ListUnavailable
};

exception e_ListError {
      t_ListErrorCode errorCode;
};

// $$033$$ moved t_InvitationReplyErrorCode & e_InvitationReplyError
//             from TINAProviderAccess.idl
enum t_InvitationReplyErrorCode {
      InvalidInvitationReplyCode,
      InvitationReplyPropertyError// $$033$$ changed name
};

exception e_InvitationReplyError {
      t_InvitationReplyErrorCode errorCode;
      t_PropertyErrorStruct propertyError;
};


};    // end module TINACommonTypes



#endif
```

Annex C: Abstract IDL for Access Part of Ret-RP

NOTE: Abstract means that these interfaces are not exported over the reference point. The exported interfaces inherit from these and can be found in Appendix D.

C.1 Module TINAAccessCommonTypes

```
// File TINAAccessCommonTypes.idl

#ifndef tinaaccesscommontypes_idl
#define tinaaccesscommontypes_idl

#include "TINACommonTypes.idl"

module TINAAccessCommonTypes {

// User Info

// The following Login-Password combination may be used
// for non-CORBA Security-compliant systems, which still
// relies on a traditional, login name & password combination.
// It is mainly provided for compability reasons for the legacy
// systems, and is not expected to be used with the CORBA compliant
// part at the same time.
// legacy authentication

typedef TINACommonTypes::Istring t_UserPassword;

struct t_UserInfo {
      TINACommonTypes::t_UserId userId;
      TINACommonTypes::t_UserName name;
      TINACommonTypes::t_UserProperties userProperties;
};

// Access Session
//

typedef sequence <octet, 16> t_AccessSessionSecretId; // $$009$$
      // 128b array generated by Retailer(should be self checking)
      // (is big enough to hold the GUID favored by DCE & DCOM)
      // (Globally Unique IDentifier)

typedef unsigned long t_AccessSessionId;

enum t_WhichAccessSession {
      CurrentAccessSession,
      SpecifiedAccessSessions,
      AllAccessSessions
};

typedef sequence<t_AccessSessionId> t_AccessSessionIdList;
```

```
//
// Implementation Note:
//    Orbix does not allow the creator of a union to set the
//    discriminator (switch tag). If true, this union requires
//    dummy cases for the other enums of t_WhichAccessSession.
//    If you encounter this problem on any platform, please
//    contact the Ret contact point, (and Patrick Farley,
//    pads@tinac.com)

union t_SpecifiedAccessSession switch (t_WhichAccessSession) {
      case SpecifiedAccessSessions: t_AccessSessionIdList asIdList;
      case CurrentAccessSession: octet dummy1; // $$052$$
      case AllAccessSessions: octet dummy2; // $$036$$
                                    // dummy var's values should not be processed
};

typedef TINACommonTypes::t_PropertyList t_AccessSessionProperties;

struct t_AccessSessionInfo {
      t_AccessSessionId id;
      TINACommonTypes::t_UserCtxtName ctxtName;
      t_AccessSessionProperties properties;
};

typedef sequence<t_AccessSessionInfo> t_AccessSessionList;


//
// Terminal Info

typedef unsigned long t_TerminalId;
typedef unsigned long t_NAPId;

// $$049$$ :
typedef sequence<t_NAPId> t_NAPIdList;

typedef string t_NAPType;

typedef TINACommonTypes::t_PropertyList t_TerminalProperties;

// t_TerminalProperties properties:
//
// defined Property names:
// name:       "TERMINAL INFO"
// value:      t_TerminalInfo

// name:       "APPLICATION INFO LIST"
// value:      t_ApplicationInfoList
//    Applications on the terminal

// No other names defined at present
```

```
//
// t_TermType
// DESCRIPTION:
// List of terminal types.
// COMMENTS:
//  - This list can be expanded.

enum t_TerminalType {
      PersonalComputer, WorkStation, TVset,
      Videotelephone, Cellularphone, PBX, VideoServer,
      VideoBridge, Telephone, G4Fax
};

// t_TermInfo
// DESCRIPTION:
// This structure contains information related to a specific terminal
// COMMENTS:
//    Not stable. To be defined further.

struct t_TerminalInfo {
      t_TerminalType terminalType;
      string operatingSystem; // includes the version
      TINACommonTypes::t_PropertyList networkCards;
      TINACommonTypes::t_PropertyList devices;
      unsigned short maxConnections;
      unsigned short memorySize;
      unsigned short diskCapacity;
};

// Provider Agent Context
//

struct t_TerminalConfig {
      t_TerminalId terminalId;
      t_TerminalType terminalType;
      t_NAPId napId;
      t_NAPType napType;
      t_TerminalProperties properties;
};


// Service Types
//

typedef unsigned long t_ServiceId;

// $$049$$ :
typedef sequence<t_ServiceId> t_ServiceIdList;

typedef TINACommonTypes::Istring t_UserServiceName;
```

```
typedef TINACommonTypes::t_PropertyList t_ServiceProperties;


struct t_ServiceInfo {
      t_ServiceId id;
      t_UserServiceName name;
      t_ServiceProperties properties;
};

typedef sequence<t_ServiceInfo> t_ServiceList;


// Session State
//

// State of the session as seen from the users point of view

enum t_UserSessionState {
      UserUnknownSessionState,    // Session State is not known
      UserActiveSession,
      UserSuspendedSession,       // Session has been suspended
      UserSuspendedParticipation, // User has suspendedParticipation
                                  // but is continuing in his absence.
                                  // (may have been quit subsequently)
      UserInvited,                // User has been invited to join
      UserNotParticipating        // User is not in the session
};

//
// Session Info
//

typedef TINACommonTypes::Istring t_SessionPurpose;

struct t_SessionOrigin {
      TINACommonTypes::t_UserId userId;// user creating the session
      TINACommonTypes::t_SessionId sessionId;
                                           // id (unique to originating user)
};

struct t_SessionInfo {
      TINACommonTypes::t_SessionId id;// my session id,
                                                    // unique to UA. (scope by
UA).
      t_SessionPurpose purpose;

      TINACommonTypes::t_ParticipantSecretId secretId;
      TINACommonTypes::t_PartyId myPartyId;
      t_UserSessionState state;

      TINACommonTypes::t_InterfaceList itfs;
```

```
      TINACommonTypes::t_SessionModelList sessionModels;

      TINACommonTypes::t_SessionProperties properties;
};

// for listing active/suspended sessions
typedef sequence<t_SessionInfo> t_SessionList;

//
// Invitations and Announcements.
//
typedef unsigned long t_InvitationId;
typedef TINACommonTypes::Istring t_InvitationReason;

struct t_InvitationOrigin {
      TINACommonTypes::t_UserId userId;// user creating the invitation
      TINACommonTypes::t_SessionId sessionId;
                                    // so they which session they invited you
from,
                                    // if you contact them
};

struct t_SessionInvitation {
      t_InvitationId id;
      TINACommonTypes::t_UserId inviteeId;// $$005$$ id of invited user, so you
can check
                                                      // the invitation was for
you.
      t_SessionPurpose purpose;
      t_ServiceInfo serviceInfo;// $$046$$
      t_InvitationReason reason;
      t_InvitationOrigin origin;
};


typedef sequence<t_SessionInvitation> t_InvitationList;

// $$003$$  moved t_InvitationReply types to commonTypes.idl

typedef unsigned long t_AnnouncementId;

// $$016$$ moved t_AnnouncementList to commonTypes.idl

//
// Start Service Properties and  Application Info.
//
// $$051$$ moved here from TINAProviderAccess.idl

typedef TINACommonTypes::Istring t_AppName;
typedef TINACommonTypes::Istring t_AppVersion;
typedef TINACommonTypes::Istring t_AppSerialNum;
typedef TINACommonTypes::Istring t_AppLicenceNum;
```

```
struct t_ApplicationInfo {
      t_AppName name;
      t_AppVersion version;
      t_AppSerialNum serialNum;
      t_AppLicenceNum licenceNum;
      TINACommonTypes::t_PropertyList properties;
      TINACommonTypes::t_InterfaceList itfs;
      TINACommonTypes::t_SessionModelList sessionModels;
      TINAStreamCommonTypes::t_SIDescList streams;

};


// t_StartServiceUAProperties properties:
//    properties to be interpreted by the User Agent, when starting a service
//
// defined Property names:
//    None defined at present
typedef TINACommonTypes::t_PropertyList t_StartServiceUAProperties;

// t_StartServiceSSProperties properties:
//    properties to be interpreted by the Service Session, when starting a
service
//
// defined Property names:
//    None defined at present
typedef TINACommonTypes::t_PropertyList t_StartServiceSSProperties;

// $$051$$ moved stuff finishes here

// Exceptions
//

enum t_AccessErrorCode {
        UnknownAccessError,
        InvalidAccessSessionSecretId,
        AccessDenied,
        SecurityContextNotSatisfied
};

exception e_AccessError {
        t_AccessErrorCode errorCode;
};

enum t_UserPropertiesErrorCode {
      InvalidUserPropertyName,
      InvalidUserPropertyValue
};

exception e_UserPropertiesError {
      t_UserPropertiesErrorCode errorCode;
```

```
      TINACommonTypes::t_UserProperty userProperty;
};

enum t_SpecifiedAccessSessionErrorCode {
      UnknownSpecifiedAccessSessionError,
      InvalidWhichAccessSession,
      InvalidAccessSessionId
};

exception e_SpecifiedAccessSessionError {
      t_SpecifiedAccessSessionErrorCode errorCode;
      t_AccessSessionId id;// Invalid AccessSessionId
};


// $$006$$
enum t_InvitationErrorCode {
      InvalidInvitationId
};

exception e_InvitationError {
      t_InvitationErrorCode errorCode;
};

};

#endif
```

C.2 Module TINAUserInitial

```
// File TINAUserInitial.idl

#ifndef tinauserinitial_idl
#define tinauserinitial_idl

#include "TINACommonTypes.idl"
#include "TINAAccessCommonTypes.idl"

module TINAUserInitial {

// $$034$$ moved all the types to outside the interface, just inside the module

// This is a draft only!!!

// requestAccess() types

typedef string t_ProviderId;

// inviteUserWithoutAccessSession() types

// $$007$$ removed typedefs
```

```
// $$002$$
enum t_AccessReplyCodes {
      SUCCESS,           // user agrees to initiate an access session
      DECLINE,           // user declines to initiate an access session
      FAILED,                     // for some unknown reason
      FORBIDDEN          // authorisation failure
};

typedef TINACommonTypes::t_PropertyList t_AccessReplyProperties;

struct t_AccessReply {
      t_AccessReplyCodes reply;
      t_AccessReplyProperties properties;
};


interface i_UserInitial
{
// behavior
//    behaviorText
//    "This interface is provided to allow a Provider to invite
//    a user to a session outside of an access session; and request
//    the establishment of an access session";
//    usage
//     " ";

      void requestAccess (
               in t_ProviderId providerId,
               out t_AccessReply reply     // $$002$$
      );

      void inviteUserOutsideAccessSession (
               in t_ProviderId providerId,
               in TINAAccessCommonTypes::t_SessionInvitation invitation,
               out TINACommonTypes::t_InvitationReply reply
      );

      // $$004$$
      void cancelInviteUserOutsideAccessSession (
               in t_ProviderId providerId,
               in TINAAccessCommonTypes::t_InvitationId id
      ) raises (
               TINAAccessCommonTypes::e_InvitationError// $$006$$
      );


};   // i_UserInitial

};

#endif
```

C.3 Module TINAUserAccess

```
// File TINAUserAccess.idl

#ifndef tinauseraccess_idl
#define tinauseraccess_idl

#include "TINACommonTypes.idl"
#include "TINAAccessCommonTypes.idl"

module TINAUserAccess {

// $$045$$ moved from inside interface i_UserAccess
typedef TINACommonTypes::t_PropertyList t_CancelAccessSessionProperties;

interface i_UserAccessGetInterfaces {

// behavior
//    behaviorText
//    "This interface allows the provider  domain to get
//    interfaces exported by this user domain."

//    usage
//    "This interface is not to be exported across
//    Ret RP. It is inherited into the exported interfaces."


      void getInterfaceTypes (
                out TINACommonTypes::t_InterfaceTypeList itfTypes
      ) raises (
                TINACommonTypes::e_ListError
      );

      void getInterface (
                                          // $$013$$ changed type to itfType
                in TINACommonTypes::t_InterfaceTypeName itfType,
                in TINACommonTypes::t_MatchProperties desiredProperties,
                out TINACommonTypes::t_InterfaceStruct itf
      ) raises (
                TINACommonTypes::e_InterfacesError,
                TINACommonTypes::e_PropertyError
      );

      void getInterfaces (
                out TINACommonTypes::t_InterfaceList itfs
      ) raises (
                TINACommonTypes::e_ListError
      );



};    // i_UserAccessGetInterfaces
```

```
interface i_UserAccess
               : i_UserAccessGetInterfaces
{
// behavior
//    behaviorText
//    "This interface is provided to a UA to perform actions during an
// access session. It inherits from i_UserAccessGetInterfaces to
// allow the retailer to ask for other interfaces exported by the
// consumer domain. (The retailer cannot register his own // // /
// interfaces!)";
      // usage
      // " ";

// $$011$$ added new operation

// DRAFT: this operation is draft only, any feedback on this operation is
//             most welcome.
      void cancelAccessSession(
               in t_CancelAccessSessionProperties options
      );


}; // i_UserAccess

interface i_UserInvite
{
// behavior
//    behaviorText
//    "This interface is provided to a UA, to invite the user to:
//             - join a service session
//             - request an access session
//    ";
//    usage
//     " ";

// $$007$$

// inviteUser() types


      void inviteUser (
               in  TINAAccessCommonTypes::t_SessionInvitation invitation,
               out TINACommonTypes::t_InvitationReply reply
      );

      // $$004$$
      void cancelInviteUser (
               in TINAAccessCommonTypes::t_InvitationId id
      ) raises (
               TINAAccessCommonTypes::e_InvitationError
      );
```

```
}; // i_UserInvite


interface i_UserTerminal {

// behavior
//    behaviorText
//    "This interface is provided to a UA, to gain information about the
// terminal context.";
      // usage
      // " ";

// getTerminalInfo() types

// $$007$$

      void getTerminalInfo(
                out TINAAccessCommonTypes::t_TerminalInfo terminalInfo
      );
}; // i_UserTerminal

interface i_UserAccessSessionInfo
{

// ...AccessSessionInfo() types

// $$007$$

oneway void newAccessSessionInfo (
                in TINAAccessCommonTypes::t_AccessSessionInfo accessSession
      );

oneway void endAccessSessionInfo (
                in TINAAccessCommonTypes::t_AccessSessionId asId
      );

// $$011$$ added new operation
onewayvoid cancelAccessSessionInfo (
                in TINAAccessCommonTypes::t_AccessSessionId asId
      );

// $$012$$ added new operation
onewayvoid newSubscribedServicesInfo (
                in TINAAccessCommonTypes::t_ServiceList services
      );

};    // i_UserAccessSessionInfo

interface i_UserSessionInfo
{

// $$007$$
```

```
oneway void newSessionInfo (
                in TINAAccessCommonTypes::t_SessionInfo session
      );

oneway void endSessionInfo (
                in TINACommonTypes::t_SessionId sessionId
      );

oneway void endMyParticipationInfo (
                in TINACommonTypes::t_SessionId sessionId
      );

oneway void suspendSessionInfo (
                in TINACommonTypes::t_SessionId sessionId
      );

oneway void suspendMyParticipationInfo (
                in TINACommonTypes::t_SessionId sessionId
      );

oneway void resumeSessionInfo (
                in TINAAccessCommonTypes::t_SessionInfo session
      );

oneway void resumeMyParticipationInfo (
                in TINAAccessCommonTypes::t_SessionInfo session
      );

oneway void joinSessionInfo (
                in TINAAccessCommonTypes::t_SessionInfo session
      );

}; // i_UserSessionInfo

};

#endif
```

C.4 Module TINAProviderInitial

```
// File TINAProviderInitial.idl

#ifndef tinaproviderinitial_idl
#define tinaproviderinitial_idl

#include "TINACommonTypes.idl"
#include "TINAAccessCommonTypes.idl"

module TINAProviderInitial {

enum t_AuthenticationStatus {// $$023$$
      SecAuthSuccess,
```

```
      SecAuthFailure,
      SecAuthContinue,
      SecAuthExpired
};

typedef unsigned long t_AuthMethod;

// $$007$$

typedef TINACommonTypes::t_PropertyList t_AuthMethodProperties;
                                                                    // $$021$$
typedef TINACommonTypes::t_MatchProperties t_AuthMethodSearchProperties;

struct t_AuthMethodDesc {
      t_AuthMethod method;
      t_AuthMethodProperties properties;
};

typedef sequence<t_AuthMethodDesc> t_AuthMethodDescList;


exception e_AuthMethodNotSupported {
      // removed t_AuthMethodDescList authMethods;// $$023$$
};

exception e_AccessNotPossible {
};

exception e_AuthenticationError {
      // removed t_AuthMethodDescList authMethods;// $$023$$
                        // type: i_ProviderAuthenticate// $$015$$
      TINACommonTypes::Istring sIOR;// $$038$$
      // removed t_Reference authIR;// $$031$$
};

exception e_AuthMethodPropertiesError {// $$021$$
      TINACommonTypes::t_PropertyErrorStruct propertyError;
};


interface i_ProviderInitial {

// behavior
//    behaviorText
//    " A reference to an interface of this type is returned to the PA
//    when it has authenticated (or requires no authentication)
//    to obtain specific userAgent interfaces.";
//    usage
//    "to obtain a userAgent reference according to the business
//    needs of the consumer";

// requestNamedAccess() types
```

```
// $$007$$ Removed typedefs

// Operation `requestNamedAccess()'
//    Used when the user is known to the provider and has already been
//    authenticated, either by DPE security or by authenticate()
// input:
//    userId:   (user name identifying requested user agent.)
//              user_name="anonymous" for anonymous access.
//              user_name may be an empty string, if the provider is
//              using userProperties to identify theuser.
//    userProperties: PropertyList which can be used to send
//              additional provider specific user privilege
//              information. This is generic, and can be used to send
//              any type of info to the provider
// output:
//    i_nameduaAccess: return: Interface reference of the UserAgent.
//    accessSessionId: Identifies the access session the operation is
//              associated with. Must be supplied in all subsequent
//              operations with the InitialAgent and UserAgent.

      void requestNamedAccess (
                in TINACommonTypes::t_UserId userId,
                in TINACommonTypes::t_UserProperties userProperties,
                out Object  namedAccessIR,             // type:
i_ProviderNamedAccess $$015$$
                out TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                out TINAAccessCommonTypes::t_AccessSessionId asId
      ) raises (
                e_AccessNotPossible,                  // $$018$$
                e_AuthenticationError,
                TINAAccessCommonTypes::e_UserPropertiesError
      );

// Operation `requestAnonymousAccess()'
//    Used when the user wants to access anonymously to the provider.
//    A secure session may already be established by DPE security or by
//    authenticate() (the laater does not mean the user is known to the
//    provider if a third party authentication protocol is used.)
// input:
//    userProperties: may be a null list
// output:as request_access

      void requestAnonymousAccess (
                in TINACommonTypes::t_UserProperties userProperties,
                out Object anonAccessIR, // type: i_ProviderAnonAccess $$015$$
                out TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                out TINAAccessCommonTypes::t_AccessSessionId asId
      ) raises (
                e_AccessNotPossible,                  // $$018$$
                e_AuthenticationError,
                TINAAccessCommonTypes::e_UserPropertiesError// $$017$$$
```

```
      );

};// i_ProviderInitial

interface i_ProviderAuthenticate {

// behavior
//    behaviorText
//    " A reference to an interface of this type is returned to the PA
//    when it wishes to choose this route to authenticate
//    itself/mutually to the provider. ";
//    usage
//    "to agree authentication options supported, acquire
//    privelege attributes for the consumer and establish
//    an access session between the consumer and the provider";

// getAuthenticationMethods() types

// $$007$$ Removed typedefs.

typedef sequence<octet> t_opaque;

//Operations `getAuthenticationMethods ()'
//input:
//    property list used to filter output
//output:
//    list of available authentication configurations

      void getAuthenticationMethods (
                in  t_AuthMethodSearchProperties desiredProperties,// $$021$$
                out t_AuthMethodDescList authMethods
      ) raises (
                e_AuthMethodPropertiesError,// $$021$$
                TINACommonTypes::e_ListError// $$021$$
      );

// Operation `authenticate()'
//    Used to authenticate a consumer attempting to gain access to a
//    user agent. invocation is a prerequsite to establishing client /
//    side credentails for establishing secure bindings unless
//    an alternative route is used
//input:
//    Method: used to identify the authentication method proposed by
//              the client, reflects the composition and generation
//              method of other opaque data
//    securityName: name assumed by consumer for authentication. may be
//              null accroding to the authentication method used.
//    authenData: opaque data containing consumer attributes to be
//              authenticated
//    privAttribReq: opaque specification of the privileges requested
//              by the consumer to create credential for subsequent
//              interactions.
```

```
//output:
//    privAttrib:privilege attributes returned in response to request.
//    continuationData: contains challenge data for the client if the
//              authentication method requires continuation of the
//              protocol
//    authSpecificData: data specific to the authentication service
//              used.
//    raises:
//    e_AuthMethodNotSupported: when the authentication mechanism used
//              by client is not supported by i_iaAuthenticate

       void authenticate(
                in t_AuthMethod authMethod,
                in string securityName,
                in t_opaque authenData,
                in t_opaque privAttribReq,
                out t_opaque privAttrib,
                out t_opaque continuationData,
                out t_opaque authSpecificData,
                out t_AuthenticationStatus authStatus// $$023$$
      ) raises (
                e_AuthMethodNotSupported
      );

// Operation continue_authentication ()'
//    To complete an authentication protocol initiated by authenticate.
//    used for second and subsequent continuations.
// input:
//    responseData: response from the client to the continuationData
//              output from the to authenticate() or previous calls to
//              continue_authenticate()
// output:
//    continuation_data
//              as per authenticate, if continuation is necessary.
//    credential_data:
//              as per authenticate, initialiation values or extra
//              items.

      void continueAuthentication(
                in t_opaque responseData,
                out t_opaque privAttrib,
                out t_opaque continuationData,
                out t_opaque authSpecificData,
                out t_AuthenticationStatus authStatus// $$023$$
      );

}; // i_ProviderAuthenticate

};

#endif
```

C.5 Module TINAProviderAccess

```
// File TINAProviderAccess.idl

#ifndef tinaprovideraccess_idl
#define tinaprovideraccess_idl

#include "TINACommonTypes.idl"
#include "TINAAccessCommonTypes.idl"
#include "TINAStreamCommonTypes.idl"

module TINAProviderAccess {


// $$042$$types from inside interface i_ProviderAccessRegisterInterfaces

typedef string t_DateTimeRegistered;// DRAFT ONLY (YUCK)

struct t_RegisteredInterfaceStruct {
      TINACommonTypes::t_InterfaceIndex index;
      TINACommonTypes::t_InterfaceStruct interfaceStruct;
                // DateTimeRegistered DRAFT ONLY: need some info on when
                // interface was registered.
      t_DateTimeRegistered when;
      TINACommonTypes::t_UserCtxtName where;
};

typedef sequence<t_RegisteredInterfaceStruct> t_RegisteredInterfaceList;


// $$042$$types from inside interface i_ProviderNamedAccess

// $$007$$ Removed typedefs

struct t_UserCtxt {
      TINACommonTypes::t_UserCtxtName ctxtName;
      TINAAccessCommonTypes::t_AccessSessionId asId;
      Object accessIR;   // type: i_UserAccess// $$015$$
      Object terminalIR; // type: i_UserTerminal// $$015$$
      Object inviteIR;   // type: i_UserInvite// $$015$$
      Object sessionInfoIR;// type: i_UserSessionInfo // $$015$$
      TINAAccessCommonTypes::t_TerminalConfig terminalConfig;
};

// $$039$$ added all the following types

typedef sequence<t_UserCtxt> t_UserCtxtList;

enum t_WhichUserCtxt {
      CurrentUserCtxt,
      SpecifiedUserCtxts,
      AllUserCtxts
};
```

```
//
// Implementation Note:
//    Orbix does not allow the creator of a union to set the
//    discriminator (switch tag). If true, this union requires
//    dummy cases for the other enums of t_WhichUserCtxt.
//    If you encounter this problem on any platform, please
//    contact the Ret contact point, (and Patrick Farley,
//    pads@tinac.com)

union t_SpecifiedUserCtxt switch (t_WhichUserCtxt) {
      case SpecifiedUserCtxts: TINACommonTypes::t_UserCtxtNameList ctxtNames;
      case CurrentUserCtxt:
      case AllUserCtxts: octet dummy; // added dummy var
                                                        // value should not be
processed
};

// $$039$$ added all the preceeding types


// $$012$$ moved types: t_ServiceId thru t_ServiceList TO accessCommonTypes.idl

typedef TINACommonTypes::t_MatchProperties t_DiscoverServiceProperties;
typedef TINACommonTypes::t_MatchProperties t_SubscribedServiceProperties;

typedef TINACommonTypes::t_MatchProperties t_SessionSearchProperties;
                                                                  // $$025$$
typedef TINACommonTypes::t_MatchProperties t_AnnouncementSearchProperties;

enum t_EndAccessSessionOption {
      DefaultEndAccessSessionOption,
      SuspendActiveSessions,
      SuspendMyParticipationActiveSessions,
      EndActiveSessions,
      EndMyParticipationActiveSessions,
      EndAllSessions,
      EndMyParticipationAllSessions
};

// $$051$$ moved following types to TINAAccessCommonTypes.idl
//             and replaced them with typedefs.

typedef TINAAccessCommonTypes::t_AppName t_AppName;
typedef TINAAccessCommonTypes::t_AppVersion t_AppVersion;
typedef TINAAccessCommonTypes::t_AppSerialNum t_AppSerialNum;
typedef TINAAccessCommonTypes::t_AppLicenceNum t_AppLicenceNum;

typedef TINAAccessCommonTypes::t_ApplicationInfo t_ApplicationInfo;

typedef TINAAccessCommonTypes::t_StartServiceUAProperties
TINAAccessCommonTypes;
```

```
typedef TINAAccessCommonTypes::t_StartServiceSSProperties
t_StartServiceSSProperties;

// $$051$$ moved stuff stops here


//
// Exceptions
//

enum t_SessionErrorCode {
      UnknownSessionError,
      InvalidSessionId,
      SessionDoesNotExist,        // $$032$$
      InvalidUserSessionState,
      SessionNotAllowed,
      SessionNotAccepted,
      SessionOpNotSupported
};

exception e_SessionError {
      t_SessionErrorCode errorCode;
      TINAAccessCommonTypes::t_UserSessionState state;
};


enum t_UserCtxtErrorCode {
      InvalidUserCtxtName,        // $$039$$
      InvalidUserAccessIR,
      InvalidUserTerminalIR,
      InvalidUserInviteIR,
      InvalidTerminalId,
      InvalidTerminalType,
      InvalidNAPId,
      InvalidNAPType,
      InvalidTerminalProperty,// $$026$$
      UserCtxtNotAvailable        // $$039$$
};

exception e_UserCtxtError {
      t_UserCtxtErrorCode errorCode;
      TINACommonTypes::t_UserCtxtName ctxtName;// $$039$$
      TINACommonTypes::t_PropertyErrorStruct propertyError;// $$026$$
               //PropertyError, if errorCode= InvalidTerminalProperty
};

enum t_RegisterUserCtxtErrorCode {// $$039$$
      UnableToRegisterUserCtxt
};

exception e_RegisterUserCtxtError {// $$039$$
      t_RegisterUserCtxtErrorCode errorCode;
```

```
};

enum t_EndAccessSessionErrorCode {
      EASE_UnknownError,
      EASE_InvalidOption,
      EASE_ActiveSession,
      EASE_SuspendedSession,
      EASE_SuspendedParticipation
};

exception e_EndAccessSessionError {
      t_EndAccessSessionErrorCode errorCode;
                                          // sessions causing a problem.
      TINACommonTypes::t_SessionIdList sessions;// $$037$$
};



// ExceptionCodes t_ServiceErrorCode;
// Example of Exception codes definition
enum t_ServiceErrorCode {
      InvalidServiceId,
      ServiceUnavailable,// $$030$$
      SessionCreationDenied,// $$030$$
      SessionNotPossibleDueToUserCtxt// $$030$$
      };

exception e_ServiceError {
      t_ServiceErrorCode errorCode;
};


// $$030$$  e_StartServiceUAPropertyError & e_StartServiceSSPropertyError
//    are defined to distinguish property errors in
//    t_StartServiceUAProperties & t_StartServiceUAProperties respectively
exception e_StartServiceUAPropertyError {
      // use the errorCodes as for e_PropertyError
      TINACommonTypes::t_PropertyErrorStruct propertyError;
};

exception e_StartServiceSSPropertyError {
      // use the errorCodes as for e_PropertyError
      TINACommonTypes::t_PropertyErrorStruct propertyError;
};

enum t_ApplicationInfoErrorCode {
      UnknownAppInfoError,
      InvalidApplication,// Can't use this application with this
                              // service/session//$$032$$
      InvalidAppInfo,
      UnknownAppName,         // I didn't recognise your app name
      InvalidAppName,         // I don't understand your app name
```

```
                                        // (eg. badly formatted)
      UnknownAppVersion,
      InvalidAppVersion,
      InvalidAppSerialNum,
      InvalidAppLicenceNum,
      AppPropertyError,
      AppSessionInterfacesError,
      AppSessionModelsError,
      AppSIDescError
};

exception e_ApplicationInfoError {
      t_ApplicationInfoErrorCode errorCode;

// t_PropertyErrorStruct:
//    Contains the t_PropertyName and t_PropertyValue in error,
//    (if t_ApplicationInfoError.errorCode==AppPropertyError
//    OR t_PropertyErrorStruct.errorCode==NoPropertyError),
//             if error is not due to a property

      TINACommonTypes::t_PropertyErrorStruct propertyError;

// t_SessionInterfacesErrorStruct
//    Only used if:
//    t_AppInfoError.errorCode == AppIntRefInfoError

      TINACommonTypes::t_InterfacesErrorStruct itfsError;
};

// $$033$$ moved t_InvitationReplyErrorCode & e_InvitationReplyError to
//             TINACommonTypes.idl


enum t_AnnouncementErrorCode {
      InvalidAnnouncementId
};

exception e_AnnouncementError {
      t_AnnouncementErrorCode errorCode;
};




interface i_ProviderAccessGetInterfaces {

// behavior
//    behaviorText
//    "This interface allows the user domain to get
//    interfaces exported by this provider domain."

//    usage
```

```
//    "This interface is not to be exported across
//    Ret RP. It is inherited into the exported interfaces."

      void getInterfaceTypes (
                in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                out TINACommonTypes::t_InterfaceTypeList itfTypes
      ) raises (
                TINAAccessCommonTypes::e_AccessError,
                TINACommonTypes::e_ListError
      );

      void getInterface (
                in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                                              // $$013$$ changed type to itfType
                in TINACommonTypes::t_InterfaceTypeName itfType,
                in TINACommonTypes::t_MatchProperties desiredProperties,
                out TINACommonTypes::t_InterfaceStruct itf
      ) raises (
                TINAAccessCommonTypes::e_AccessError,
                TINACommonTypes::e_InterfacesError,
                TINACommonTypes::e_PropertyError
      );

      void getInterfaces (
                in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                                                              // $$035$$
                in TINACommonTypes::t_MatchProperties desiredProperties,
                out TINACommonTypes::t_InterfaceList itfs
      ) raises (
                TINAAccessCommonTypes::e_AccessError,
                TINACommonTypes::e_PropertyError,// $$035$$
                TINACommonTypes::e_ListError
      );

};    // i_ProviderAccessGetInterfaces

interface i_ProviderAccessRegisterInterfaces {

// behavior
//    behaviorText
//    "This interface allows the client to register interfaces
//    exported by the client domain."

//    usage
//    "This interface is not to be exported across Ret RP.
//    It is inherited into the exported interfaces."

// $$042$$moved types to just inside module.

                // register interfaces to be used only during
                // current access session
```

```
        void registerInterface (
                 in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                 in TINACommonTypes::t_InterfaceStruct itf,
                 out TINACommonTypes::t_InterfaceIndex index
        ) raises (
                 TINAAccessCommonTypes::e_AccessError,
                 TINACommonTypes::e_InterfacesError,
                 TINACommonTypes::e_RegisterError
        );

        // $$014$$ new op
        void registerInterfaces (
                 in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                 inout TINACommonTypes::t_RegisterInterfaceList itfs
        ) raises (
                 TINAAccessCommonTypes::e_AccessError,
                 TINACommonTypes::e_InterfacesError,
                 TINACommonTypes::e_RegisterError
        );


                 // register interfaces which will be accessible
                 // outside the access session.

        void registerInterfaceOutsideAccessSession (
                 in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                 in TINACommonTypes::t_InterfaceStruct itf,
                 out TINACommonTypes::t_InterfaceIndex index
        ) raises (
                 TINAAccessCommonTypes::e_AccessError,
                 TINACommonTypes::e_InterfacesError,
                 TINACommonTypes::e_RegisterError
        );

        // $$014$$ new op
        void registerInterfacesOutsideAccessSession (
                 in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                 inout TINACommonTypes::t_RegisterInterfaceList itfs
        ) raises (
                 TINAAccessCommonTypes::e_AccessError,
                 TINACommonTypes::e_InterfacesError,
                 TINACommonTypes::e_RegisterError
        );

        void listRegisteredInterfaces (
                 in TINAAccessCommonTypes:: t_AccessSessionSecretId asSecretId,
                 in TINAAccessCommonTypes::t_SpecifiedAccessSession as,
                 out t_RegisteredInterfaceList itfs
        ) raises (
                 TINAAccessCommonTypes::e_AccessError,
                 TINAAccessCommonTypes::e_SpecifiedAccessSessionError,
                 TINACommonTypes::e_ListError
```

```
        );

        void unregisterInterface (
                  in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                  in TINACommonTypes::t_InterfaceIndex index
        ) raises (
                  TINAAccessCommonTypes::e_AccessError,
                  TINACommonTypes::e_InterfacesError,
                  TINACommonTypes::e_UnregisterError

        );

        // $$014$$ new op
        void unregisterInterfaces (
                  in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                  in TINACommonTypes::t_InterfaceIndexList indexes
        ) raises (
                  TINAAccessCommonTypes::e_AccessError,
                  TINACommonTypes::e_InterfacesError,
                  TINACommonTypes::e_UnregisterError
        );


  };    // i_ProviderAccessRegisterInterfaces

  interface i_ProviderAccessInterfaces
                          : i_ProviderAccessGetInterfaces,
                            i_ProviderAccessRegisterInterfaces
  {
  // behavior
  //    behaviorText
  //    "This interface allows the client to get interfaces
  //    exported by this domain, and register interfaces exported
  //    by the client domain."

  //    usage
  //    "This interface is not to be exported across Ret RP.
  //    It is inherited into the exported interfaces."

  // No additional operations are defined

  };    // i_ProviderAccessInterfaces


  interface i_ProviderAccess: i_ProviderAccessInterfaces
  {
  // behavior
  //    behaviorText
  //    "This interface is the place to put operations which should be
  //    shared between i_ProviderNamedAccess and i_ProviderAnonAccess.
  //    Currently none are defined."
```

```
//    usage
//    "This interface is not to be exported across Ret RP.
//    It is inherited into the exported interfaces."

// No additional operations are defined

};    // interface i_ProviderAccess


interface i_ProviderNamedAccess
                : i_ProviderAccess
{

// $$042$$moved types to just inside module.

      void setUserCtxt (
                in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                in t_UserCtxt userCtxt
      ) raises (
                TINAAccessCommonTypes::e_AccessError,
                e_UserCtxtError
      );

// $$039$$ new op
      void getUserCtxt (
                in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                in TINACommonTypes::t_UserCtxtName ctxtName,
                out t_UserCtxt userCtxt
      ) raises (
                TINAAccessCommonTypes::e_AccessError,
                e_UserCtxtError
      );

// $$039$$ new op
      void getUserCtxts (
                in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                in t_SpecifiedUserCtxt ctxt,
                out t_UserCtxtList userCtxts
      ) raises (
                TINAAccessCommonTypes::e_AccessError,
                e_UserCtxtError,
                TINACommonTypes::e_ListError
      );

// $$039$$ new op
      void getUserCtxtsAccessSessions (
                in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                in TINAAccessCommonTypes::t_SpecifiedAccessSession as,
                out t_UserCtxtList userCtxts
      ) raises (
                TINAAccessCommonTypes::e_AccessError,
                TINAAccessCommonTypes::e_SpecifiedAccessSessionError,
```

```
                TINACommonTypes::e_ListError
        );

    // $$039$$ new op
        void registerUserCtxtsAccessSessions (
                in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                in TINAAccessCommonTypes::t_SpecifiedAccessSession as
        ) raises (
                TINAAccessCommonTypes::e_AccessError,
                TINAAccessCommonTypes::e_SpecifiedAccessSessionError,
                e_RegisterUserCtxtError
        );


        void listAccessSessions (
                in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                out TINAAccessCommonTypes::t_AccessSessionList asList
        ) raises (
                TINAAccessCommonTypes::e_AccessError,
                TINACommonTypes::e_ListError
        );

        void endAccessSession(
                in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                in TINAAccessCommonTypes::t_SpecifiedAccessSession as,
                in t_EndAccessSessionOption option
        ) raises (
                TINAAccessCommonTypes::e_AccessError,
                                                                // $$027$$
                TINAAccessCommonTypes::e_SpecifiedAccessSessionError,
                e_EndAccessSessionError
        );

        void getUserInfo(
                in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                out TINAAccessCommonTypes::t_UserInfo userInfo
        ) raises (
                TINAAccessCommonTypes::e_AccessError
        );


        void listSubscribedServices (
                in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                in t_SubscribedServiceProperties desiredProperties,
                                        // $$012$$ moved type t_ServiceList
                out TINAAccessCommonTypes::t_ServiceList services
        ) raises (
                TINAAccessCommonTypes::e_AccessError,
                TINACommonTypes::e_PropertyError,
                TINACommonTypes::e_ListError
        );
```

```
        void discoverServices(
                  in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                  in t_DiscoverServiceProperties desiredProperties,
                  in unsigned long howMany,
                                              // $$012$$ moved type t_ServiceList
                  out TINAAccessCommonTypes::t_ServiceList services,
                                    // type: i_DiscoverServicesIterator //
  $$015$$
                  out Object iteratorIR
        ) raises (
                  TINAAccessCommonTypes::e_AccessError,
                  TINACommonTypes::e_PropertyError,
                  TINACommonTypes::e_ListError
        );

        void listServiceSessions (
                  in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                  in TINAAccessCommonTypes::t_SpecifiedAccessSession as,
                  in t_SessionSearchProperties desiredProperties,
                  out TINAAccessCommonTypes::t_SessionList sessions
        ) raises (
                  TINAAccessCommonTypes::e_AccessError,
                  TINAAccessCommonTypes::e_SpecifiedAccessSessionError,
                  TINACommonTypes::e_PropertyError,
                  TINACommonTypes::e_ListError
        );

        void getSessionModels (
                  in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                  in TINACommonTypes::t_SessionId sessionId,
                  out TINACommonTypes::t_SessionModelList sessionModels
        ) raises (
                  TINAAccessCommonTypes::e_AccessError,
                  e_SessionError,
                  TINACommonTypes::e_ListError
        );

        void getSessionInterfaceTypes (
                  in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                  in TINACommonTypes::t_SessionId sessionId,
                  out TINACommonTypes::t_InterfaceTypeList itfTypes
        ) raises (
                  TINAAccessCommonTypes::e_AccessError,
                  e_SessionError,
                  TINACommonTypes::e_ListError
        );

        void getSessionInterface (
                  in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                  in TINACommonTypes::t_SessionId sessionId,
                                              // $$013$$ changed type to itfType
```

```
            in TINACommonTypes::t_InterfaceTypeName itfType,
            out TINACommonTypes::t_InterfaceStruct itf
    ) raises (
            TINAAccessCommonTypes::e_AccessError,
            e_SessionError,
            TINACommonTypes::e_InterfacesError
                    // $$028$$ removed TINACommonTypes::e_ListError
    );

    void getSessionInterfaces (
            in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
            in TINACommonTypes::t_SessionId sessionId,
            out TINACommonTypes::t_InterfaceList itfs
    ) raises (
            TINAAccessCommonTypes::e_AccessError,
            e_SessionError,
            TINACommonTypes::e_ListError
    );



    void listSessionInvitations (
            in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
            out TINAAccessCommonTypes::t_InvitationList invitations
    ) raises (
            TINAAccessCommonTypes::e_AccessError,
            TINACommonTypes::e_ListError
    );

    void listSessionAnnouncements (
            in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                                                            // $$025$$
            in t_AnnouncementSearchProperties desiredProperties,
            out TINACommonTypes::t_AnnouncementList announcements
    ) raises (
            TINAAccessCommonTypes::e_AccessError,
            TINACommonTypes::e_PropertyError,
            TINACommonTypes::e_ListError
    );

    void startService (
            in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
            in TINAAccessCommonTypes::t_ServiceId serviceId,
            in t_ApplicationInfo app,
            in TINACommonTypes::t_SessionModelReq sessionModelReq,
            in t_StartServiceUAProperties uaProperties,
            in t_StartServiceSSProperties ssProperties,
            out TINAAccessCommonTypes::t_SessionInfo sessionInfo
    ) raises (
            TINAAccessCommonTypes::e_AccessError,
            e_ServiceError,
            e_ApplicationInfoError,
```

```
            TINACommonTypes::e_SessionModelError,
            e_StartServiceUAPropertyError,// $$030$$
            e_StartServiceSSPropertyError// $$030$$
    );

    void endSession (
            in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
            in TINACommonTypes::t_SessionId sessionId
    ) raises (
            TINAAccessCommonTypes::e_AccessError,
            e_SessionError
    );


    void endMyParticipation (
            in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
            in TINACommonTypes::t_SessionId sessionId
    ) raises (
            TINAAccessCommonTypes::e_AccessError,
            e_SessionError
    );

    void suspendSession (
            in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
            in TINACommonTypes::t_SessionId sessionId
    ) raises (
            TINAAccessCommonTypes::e_AccessError,
            e_SessionError
    );

    void suspendMyParticipation (
            in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
            in TINACommonTypes::t_SessionId sessionId
    ) raises (
            TINAAccessCommonTypes::e_AccessError,
            e_SessionError
    );

    void resumeSession (
            in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
            in TINACommonTypes::t_SessionId sessionId,
            in t_ApplicationInfo app,
            out TINAAccessCommonTypes::t_SessionInfo sessionInfo
    ) raises (
            TINAAccessCommonTypes::e_AccessError,
            e_SessionError,
            e_ApplicationInfoError
    );

    void resumeMyParticipation (
            in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
            in TINACommonTypes::t_SessionId sessionId,
```

```
                in t_ApplicationInfo app,
                out TINAAccessCommonTypes::t_SessionInfo sessionInfo
        ) raises (
                TINAAccessCommonTypes::e_AccessError,
                e_SessionError,
                e_ApplicationInfoError
        );

        void joinSessionWithInvitation (
                in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                in TINAAccessCommonTypes::t_InvitationId invitationId,
                in t_ApplicationInfo app,
                out TINAAccessCommonTypes::t_SessionInfo sessionInfo
        ) raises (
                TINAAccessCommonTypes::e_AccessError,
                e_SessionError,                                 // $$019$$
                TINAAccessCommonTypes::e_InvitationError,// $$006$$
                e_ApplicationInfoError
        );

        void joinSessionWithAnnouncement (
                in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                in TINAAccessCommonTypes::t_AnnouncementId announcementId,
                in t_ApplicationInfo app,
                out TINAAccessCommonTypes::t_SessionInfo sessionInfo
        ) raises (
                TINAAccessCommonTypes::e_AccessError,
                e_SessionError,                                 // $$019$$
                e_AnnouncementError,
                e_ApplicationInfoError
        );

        void replyToInvitation (
                in TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
                in TINAAccessCommonTypes::t_InvitationId invitationId,
                in TINACommonTypes::t_InvitationReply reply
        ) raises (
                TINAAccessCommonTypes::e_AccessError,
                TINAAccessCommonTypes::e_InvitationError,// $$006$$
                TINACommonTypes::e_InvitationReplyError
        );

};  // i_ProviderNamedAccess


interface i_ProviderAnonAccess
                        : i_ProviderAccess
{
        // No additional operations defined.

};  // i_ProviderAnonAccess
```

```
// $$029$$ moved to here from TINARetRetailerAccess.idl
interface i_DiscoverServicesIterator {

      exception e_UnknownDiscoverServicesMaxLeft {};


      void maxLeft (
               out unsigned long n
      ) raises (
               e_UnknownDiscoverServicesMaxLeft
      );

      // moreLeft: true if there are more interfaces,
      //       (accessible thru the iterator (This interface)),
      //       false if there are no more interfaces to retrieve

      void nextN (
               in unsigned long n,
                                          // n >= number returned in services
               out TINAAccessCommonTypes::t_ServiceList services,
               out boolean moreLeft
      ) raises (
               TINACommonTypes::e_ListError
      );

      void destroy ();

}; // i_DiscoverServicesIterator


}; // module TINAProviderAccess

#endif
```

Annex D: Ret-RP Specific IDL for Access Part

All modules inherit from modules defined in Annex C.

D.1 Module TINARetConsumerInitial

```
// File TINARetConsumerInitial.idl

#ifndef tinaretconsumerinitial_idl
#define tinaretconsumerinitial_idl

#include "TINAUserInitial.idl"

module TINARetConsumerInitial {

interface i_ConsumerInitial: TINAUserInitial::i_UserInitial {

// No additional operations for Consumer.

};

};

#endif
```

D.2 TINARetConsumerAccess

```
// File TINARetConsumerAccess.idl

#ifndef tinaretconsumeraccess_idl
#define tinaretconsumeraccess_idl

#include "TINAUserAccess.idl"

module TINARetConsumerAccess {

interface i_ConsumerAccess: TINAUserAccess::i_UserAccess {

// No additional operations for Consumer.

}; // i_ConsumerAccess


interface i_ConsumerInvite: TINAUserAccess::i_UserInvite {

// No additional operations for Consumer.

}; // i_ConsumerInvite


interface i_ConsumerTerminal: TINAUserAccess::i_UserTerminal {
```

```
// No additional operations for Consumer.

}; // i_ConsumerTerminal


interface i_ConsumerAccessSessionInfo: TINAUserAccess::i_UserAccessSessionInfo
{     // $$001$$

// No additional operations for Consumer.

}; // i_ConsumerAccessSessioninfo


interface i_ConsumerSessionInfo: TINAUserAccess::i_UserSessionInfo {

// No additional operations for Consumer.

}; // i_ConsumerSessionInfo

};

#endif
```

D.3 TINARetRetailerInitial

```
// File TINARetRetailerInitial.idl

#ifndef tinaretretailerinitial_idl
#define tinaretretailerinitial_idl

#include "TINAProviderInitial.idl"

module TINARetRetailerInitial {

interface i_RetailerInitial: TINAProviderInitial::i_ProviderInitial {

// No Retailer specific operations defined.

}; // i_RetailerInitial


interface i_RetailerAuthenticate: TINAProviderInitial::i_ProviderAuthenticate {

// No Retailer specific operations defined.

}; // i_RetailerAuthenticate


};

#endif
```

D.4 TINARetRetailerAccess

```
// File TINARetRetailerAccess.idl

#ifndef tinaretretaileraccess_idl
#define tinaretretaileraccess_idl


#include "TINAProviderAccess.idl"

module TINARetRetailerAccess {

interface i_RetailerAccess {

// behavior
//    behaviorText
//    "This interface is the place to put operations which should be
//    shared between i_RetailerNamedAccess and i_RetailerAnonAccess.
//    These are specific to the Ret RP, so they don't go in i_ProviderAccess
//    Currently none are defined."

//    usage
//    "This interface is not to be exported across Ret RP.
//    It is inherited into the exported interfaces."

// No retailer specific operations are defined

}; // i_RetailerAccess


interface i_RetailerNamedAccess
                        :TINAProviderAccess::i_ProviderNamedAccess,
                        i_RetailerAccess
{
      // Change the name of the interface for Ret RP.
}; // i_RetailerNamedAccess


interface i_RetailerAnonAccess
                        : TINAProviderAccess::i_ProviderAnonAccess,
                        i_RetailerAccess
{
      // Change the name of the interface for Ret RP
}; // i_RetailerAnonAccess

// $$029$$ moved interface i_DiscoverServicesIterator to TINAProviderAccess.idl

};    // module TINARetRetailerAccess

#endif
```

Annex E: IDL for Usage Part of Ret-RP

This annex contains the idl definitions for the feature sets.

E.1 Module TINAUsageCommonTypes

```
// File TINAUsageCommonTypes.idl

#ifndef tinausagecommontypes_idl
#define tinausagecommontypes_idl

#include "TINACommonTypes.idl"

module TINAUsageCommonTypes {

typedef TINACommonTypes::Istring Istring;

typedef long t_IndId; //$$U003$$

typedef unsigned short t_PartyType;// use is service specific
                                          //max 2^16 party types (65536)

struct t_PartyDetails {
      TINACommonTypes::t_PartyId id;
      t_PartyType partyType;// $$U004$$
      Istring partyName;// printable for human consumption
                                  // not necess userId
};
typedef sequence<t_PartyDetails> t_PartyDetailsList;


typedef unsigned long t_InvitationId;


// Exceptions
//

enum t_UsageErrorCode {
      UnknownUsageError,
      InvalidParticipantSecretId, //
      UsageNotAllowed,// You don't have permission to do it
      UsageNotAccepted,// Owners have declined request
      UsageOpNotSupported// op is not supported by session
};

exception e_UsageError {
      t_UsageErrorCode errorCode;
};


enum t_PartyDomainErrorCode {
      PD_UnknownError,
```

```
      PD_InvalidSessionId,
      PD_OpNotSupported
};

exception e_PartyDomainError {
      t_PartyDomainErrorCode errorCode;//$$U001$$
};


enum t_PartyErrorCode {
      InvalidPartyId,
      InvalidPartyType,
      PartySuspended
};

exception e_PartyError {
      t_PartyErrorCode errorCode;
      TINACommonTypes::t_PartyId id;
      t_PartyType partyType;      // $$U004$$
};

// $$043$$ moved t_UserDetailsErrorCode & e_UserDetailsError
//             to TINACommonTypes.idl


enum t_AnnouncementErrorCode {
      UnknownAnnouncementError,
      InvalidAnnouncementProperty
};

exception e_AnnouncementError {
      t_AnnouncementErrorCode errorCode;
      TINACommonTypes::t_PropertyErrorStruct propertyError;
                        // Return the properties in error
};

enum t_IndErrorCode {
      UnknownIndError,
      UnknownIndId,
      InvalidIndId};

exception e_IndError {
      t_IndErrorCode errorCode;};


};    // module TINAUsageCommonTypes

#endif
```

E.2 Module TINAProviderBasicUsage

```
// File TINAProviderBasicUsage.idl

#ifndef tinaproviderbasicusage_idl
#define tinaproviderbasicusage_idl

#include "TINACommonTypes.idl"
#include "TINAUsageCommonTypes.idl"
#include "TINASessionModel.idl"

module TINAProviderBasicUsage {


interface i_ProviderGetInterfaces {

      void getInterfaceTypes (
                in TINACommonTypes::t_ParticipantSecretId myId,
                out TINACommonTypes::t_InterfaceTypeList itfTypeList
      ) raises (
                TINAUsageCommonTypes::e_UsageError
      );

      void getInterface (
                in TINACommonTypes::t_ParticipantSecretId myId,
                in TINACommonTypes::t_InterfaceTypeName type,
                in TINACommonTypes::t_MatchProperties desiredProperties,
                out TINACommonTypes::t_InterfaceStruct itf
      ) raises (
                TINAUsageCommonTypes::e_UsageError,
                TINACommonTypes::e_InterfacesError,
                TINACommonTypes::e_PropertyError
      );

      void getInterfaces (
                in TINACommonTypes::t_ParticipantSecretId myId,
                in TINACommonTypes::t_MatchProperties desiredProperties,
                out TINACommonTypes::t_InterfaceList itfList
      ) raises (
                TINAUsageCommonTypes::e_UsageError,
                TINACommonTypes::e_PropertyError
      );

};    // interface i_ProviderGetInterfaces


interface i_ProviderRegisterInterfaces {

      void registerInterface (
                in TINACommonTypes::t_ParticipantSecretId myId,
                in TINACommonTypes::t_InterfaceStruct itf,
                out TINACommonTypes::t_InterfaceIndex itfIndex
      ) raises (
```

```
                TINAUsageCommonTypes::e_UsageError,
                TINACommonTypes::e_InterfacesError
        );

        void registerInterfaces (
                in TINACommonTypes::t_ParticipantSecretId myId,
                inout TINACommonTypes::t_RegisterInterfaceList itfs
        ) raises (
                TINAUsageCommonTypes::e_UsageError,
                TINACommonTypes::e_InterfacesError,
                TINACommonTypes::e_RegisterError
        );

        void registerInterfaceTypes (
                in TINACommonTypes::t_ParticipantSecretId myId,
                in TINACommonTypes::t_InterfaceTypeList types
        ) raises (
                TINAUsageCommonTypes::e_UsageError,
                TINACommonTypes::e_InterfacesError
        );

        void listRegisteredInterfaces (
                in TINACommonTypes::t_ParticipantSecretId myId,
                out TINACommonTypes::t_RegisterInterfaceList registeredItfs
        ) raises (
                TINAUsageCommonTypes::e_UsageError
        );

        void unregisterInterface (
                in TINACommonTypes::t_ParticipantSecretId myId,
                in TINACommonTypes::t_InterfaceIndex index
        ) raises (
                TINAUsageCommonTypes::e_UsageError,
                TINACommonTypes::e_InterfacesError,
                TINACommonTypes::e_UnregisterError
        );


        void unregisterInterfaces (
                in TINACommonTypes::t_ParticipantSecretId myId,
                in TINACommonTypes::t_InterfaceIndexList indexes
        ) raises (
                TINAUsageCommonTypes::e_UsageError,
                TINACommonTypes::e_InterfacesError,
                TINACommonTypes::e_UnregisterError

        );

};    // interface i_ProviderRegisterInterfaces


interface i_ProviderInterfaces
```

```
                        : i_ProviderGetInterfaces,
                        i_ProviderRegisterInterfaces
{
};    // interface i_ProviderInterfaces


interface i_ProviderBasicReq
               : i_ProviderInterfaces,
               TINASessionModel::i_SessionModel
{

      void endSessionReq (
               in TINACommonTypes::t_ParticipantSecretId myId
      ) raises (
               TINAUsageCommonTypes::e_UsageError
      );

      void suspendSessionReq(
               in TINACommonTypes::t_ParticipantSecretId myId,
               out TINACommonTypes::t_SessionId sessionId
      ) raises (
               TINAUsageCommonTypes::e_UsageError
      );

};    // interface i_ProviderBasicReq



};    // module TINAProviderBasicUsage


#endif
```

E.3 Module TINAPartyBasicExtUsage

```
// File TINAPartyBasicExtUsage.idl

#ifndef tinapartybasicextusage_idl
#define tinapartybasicextusage_idl

#include "TINACommonTypes.idl"
#include "TINAUsageCommonTypes.idl"
#include "TINASessionModel.idl"

module TINAPartyBasicExtUsage {


interface i_PartyGetInterfaces {

      void getInterfaceTypes (
               in TINACommonTypes::t_SessionId sessionId,
               out TINACommonTypes::t_InterfaceTypeList itfTypeList
      ) raises (
```

```
                TINAUsageCommonTypes::e_UsageError
        );

        void getInterface (
                in TINACommonTypes::t_SessionId sessionId,
                in TINACommonTypes::t_InterfaceTypeName type,
                in TINACommonTypes::t_MatchProperties desiredProperties,
                out TINACommonTypes::t_InterfaceStruct itf
        ) raises (
                TINAUsageCommonTypes::e_UsageError,
                TINACommonTypes::e_InterfacesError,
                TINACommonTypes::e_PropertyError
        );

        void getInterfaces (
                in TINACommonTypes::t_SessionId sessionId,
                in TINACommonTypes::t_MatchProperties desiredProperties,
                out TINACommonTypes::t_InterfaceList itfList
        ) raises (
                TINAUsageCommonTypes::e_UsageError,
                TINACommonTypes::e_PropertyError
        );

    };    // interface i_PartyGetInterfaces


    interface i_PartyBasicExtReq
                : i_PartyGetInterfaces,
                TINASessionModel::i_SessionModel
    {

    };    // interface i_PartyBasicExtReq



    };    // module TINAPartyBasicExtUsage


    #endif
```

E.4 Module TINAProviderMultiPartyUsage

```
    // File TINAProviderMultipartyUsage.idl

    #ifndef tinaprovidermultipartyusage_idl
    #define tinaprovidermultipartyusage_idl

    #include "TINACommonTypes.idl"
    #include "TINAUsageCommonTypes.idl"

    module TINAProviderMultipartyUsage {

    interface i_ProviderMultipartyReq {
```

```
void listParties (
          in TINACommonTypes::t_ParticipantSecretId myId,
          out TINACommonTypes::t_PartyIdList partyIdList
) raises (
          TINAUsageCommonTypes::e_UsageError
);


void listPartiesWithDetails (
          in TINACommonTypes::t_ParticipantSecretId myId,
          out TINAUsageCommonTypes::t_PartyDetailsList partyDetailsList
) raises (
          TINAUsageCommonTypes::e_UsageError
);

void getPartyDetails (
          in TINACommonTypes::t_ParticipantSecretId myId,
          in TINACommonTypes::t_PartyId partyId,
          out TINAUsageCommonTypes::t_PartyDetailsList partyDetailsList
) raises (
          TINAUsageCommonTypes::e_UsageError,
          TINAUsageCommonTypes::e_PartyError
);

void getMyPartyDetails (
          in TINACommonTypes::t_ParticipantSecretId myId,
          out TINAUsageCommonTypes::t_PartyDetails myDetails
) raises (
          TINAUsageCommonTypes::e_UsageError
);

void modifyPartyTypeReq (
          in TINACommonTypes::t_ParticipantSecretId myId,
          in TINACommonTypes::t_PartyId partyId,
          in TINAUsageCommonTypes::t_PartyType newType
) raises (
          TINAUsageCommonTypes::e_UsageError,
          TINAUsageCommonTypes::e_PartyError
);

void endMyParticipationReq (
          in TINACommonTypes::t_ParticipantSecretId myId
) raises (
          TINAUsageCommonTypes::e_UsageError
);

void endPartyReq (
          in TINACommonTypes::t_ParticipantSecretId myId,
          in TINACommonTypes::t_PartyId endPartyId
) raises (
```

```
                TINAUsageCommonTypes::e_UsageError,
                TINAUsageCommonTypes::e_PartyError
        );

        void suspendMyParticipationReq (
                in TINACommonTypes::t_ParticipantSecretId myId
        ) raises (
                TINAUsageCommonTypes::e_UsageError
        );

        void suspendPartyReq (
                in TINACommonTypes::t_ParticipantSecretId myId,
                in TINACommonTypes::t_PartyId suspendPartyId
        ) raises (
                TINAUsageCommonTypes::e_UsageError,
                TINAUsageCommonTypes::e_PartyError
        );

        void inviteUserReq (
                in TINACommonTypes::t_ParticipantSecretId myId,
                in TINACommonTypes::t_UserDetails invitedUser,
                out TINAUsageCommonTypes::t_InvitationId invitationId,
                out TINACommonTypes::t_InvitationReply reply
        ) raises (
                TINAUsageCommonTypes::e_UsageError,
                TINACommonTypes::e_UserDetailsError
        );

        void announceSessionReq (
                in TINACommonTypes::t_ParticipantSecretId myId,
                in TINACommonTypes::t_AnnouncementProperties announcement
        ) raises (
                TINAUsageCommonTypes::e_UsageError,
                TINAUsageCommonTypes::e_AnnouncementError
        );

    };    // interface i_ProviderMultipartyReq



    };    // module TINAProviderMultipartyUsage


    #endif
```

E.5 Module TINAPartyMultipartyUsage

```
// File TINAPartyMultipartyUsage.idl

#ifndef tinapartymultipartyusage_idl
#define tinapartymultipartyusage_idl

#include "TINACommonTypes.idl"
#include "TINAUsageCommonTypes.idl"

module TINAPartyMultipartyUsage {


interface i_PartyMultipartyExe {

      void modifyPartyTypeExe (
               in TINACommonTypes::t_SessionId sessionId,
               in TINAUsageCommonTypes::t_PartyType newType
      ) raises (
               TINAUsageCommonTypes::e_PartyDomainError,
               TINAUsageCommonTypes::e_PartyError
      );

      void endSessionExe (
               in TINACommonTypes::t_SessionId sessionId
      ) raises (
               TINAUsageCommonTypes::e_PartyDomainError
      );

      void endPartyExe (
               in TINACommonTypes::t_SessionId sessionId
      ) raises (
               TINAUsageCommonTypes::e_PartyDomainError
      );

      void suspendSessionExe (
               in TINACommonTypes::t_SessionId sessionId
      ) raises (
               TINAUsageCommonTypes::e_PartyDomainError
      );

      void suspendPartyExe (
               in TINACommonTypes::t_SessionId sessionId
      ) raises (
               TINAUsageCommonTypes::e_PartyDomainError
      );


};    // interface i_PartyMultipartyExe


interface i_PartyMultipartyInfo {
```

```
  onewayvoid modifyPartyTypeInfo (
                in TINACommonTypes::t_SessionId sessionId,
                in TINAUsageCommonTypes::t_PartyDetails partyDetails
       );


  onewayvoid endPartyInfo (
                in TINACommonTypes::t_SessionId sessionId,
                in TINACommonTypes::t_PartyId partyId
       );

  onewayvoid suspendPartyInfo (
                in TINACommonTypes::t_SessionId sessionId,
                in TINACommonTypes::t_PartyId partyId
       );

  onewayvoid resumePartyInfo (
                in TINACommonTypes::t_SessionId sessionId,
                in TINACommonTypes::t_PartyId partyId
       );

  onewayvoid joinSessionInfo (
                in TINACommonTypes::t_SessionId sessionId,
                in TINAUsageCommonTypes::t_PartyDetails partyDetails,
                in TINACommonTypes::t_UserDetails userDetails// $$047$$
       );

  onewayvoid inviteUserInfo (
                in TINACommonTypes::t_SessionId sessionId,
                in TINACommonTypes::t_UserDetails userDetails,
                in TINAUsageCommonTypes::t_InvitationId invitationId
       );
  onewayvoid inviteReplyInfo (
                in TINACommonTypes::t_SessionId sessionId,
                in TINAUsageCommonTypes::t_InvitationId invitationId,
                in TINACommonTypes::t_InvitationReply reply
       );

  onewayvoid announceSessionInfo (
                in TINACommonTypes::t_SessionId sessionId,
                in TINACommonTypes::t_AnnouncementProperties announcement
       );

  };    // interface i_PartyMultipartyInfo


  };    // module TINAPartyMultipartyUsage

  #endif
```

E.6 Module TINAPartyMultipartyIndUsage

```
// File TINAPartyMultipartyIndUsage.idl

#ifndef tinapartymultipartyindusage_idl
#define tinapartymultipartyindusage_idl

#include "TINACommonTypes.idl"
#include "TINAUsageCommonTypes.idl"


module TINAPartyMultipartyIndUsage {


interface i_PartyMultipartyInd {

      void operationCancelled (
                in TINACommonTypes::t_SessionId sessionId,
                in TINAUsageCommonTypes::t_IndId indId
      ) raises (
                TINAUsageCommonTypes::e_PartyDomainError,
                TINAUsageCommonTypes::e_IndError
      );

      void modifyPartyTypeInd(
                in TINACommonTypes::t_SessionId sessionId,
                in TINAUsageCommonTypes::t_IndId indId,
                in TINACommonTypes::t_PartyId reqPartyId,
                in TINAUsageCommonTypes::t_PartyDetails partyDetails
      ) raises (
                TINAUsageCommonTypes::e_PartyDomainError,
                TINAUsageCommonTypes::e_PartyError
      );


      void endSessionInd (
                in TINACommonTypes::t_SessionId sessionId,
                in TINAUsageCommonTypes::t_IndId indId,
                in TINACommonTypes::t_PartyId reqPartyId
      ) raises (
                TINAUsageCommonTypes::e_PartyDomainError,
                TINAUsageCommonTypes::e_PartyError
      );

      void endPartyInd (
                in TINACommonTypes::t_SessionId sessionId,
                in TINAUsageCommonTypes::t_IndId indId,
                in TINACommonTypes::t_PartyId reqPartyId,
                in TINACommonTypes::t_PartyId partyId
      ) raises (
                TINAUsageCommonTypes::e_PartyDomainError,
                TINAUsageCommonTypes::e_PartyError
      );
```

```
      void suspendSessionInd (
                in TINACommonTypes::t_SessionId sessionId,
                in TINAUsageCommonTypes::t_IndId indId,
                in TINACommonTypes::t_PartyId reqPartyId
      ) raises (
                TINAUsageCommonTypes::e_PartyDomainError,
                TINAUsageCommonTypes::e_PartyError
      );

      void resumeSessionInd(
                in TINACommonTypes::t_SessionId sessionId,
                in TINAUsageCommonTypes::t_IndId indId,
                in TINACommonTypes::t_PartyId reqPartyId
      ) raises (
                TINAUsageCommonTypes::e_PartyDomainError,
                TINAUsageCommonTypes::e_PartyError
      );

      void suspendPartyInd (
                in TINACommonTypes::t_SessionId sessionId,
                in TINAUsageCommonTypes::t_IndId indId,
                in TINACommonTypes::t_PartyId reqPartyId,
                in TINACommonTypes::t_PartyId partyId
      ) raises (
                TINAUsageCommonTypes::e_PartyDomainError,
                TINAUsageCommonTypes::e_PartyError
      );

      void resumePartyInd (
                in TINACommonTypes::t_SessionId sessionId,
                in TINAUsageCommonTypes::t_IndId indId,
                in TINACommonTypes::t_PartyId partyId
      ) raises (
                TINAUsageCommonTypes::e_PartyDomainError,
                TINAUsageCommonTypes::e_PartyError
      );

  // $$046$$ removed in TINACommonTypes::t_PartyId reqPartyId, and
  //                                          TINAUsageCommonTypes::e_PartyError
      void joinSessionInd (
                in TINACommonTypes::t_SessionId sessionId,
                in TINAUsageCommonTypes::t_IndId indId,
                in TINACommonTypes::t_UserDetails userDetails
      ) raises (
                TINAUsageCommonTypes::e_PartyDomainError,
                TINACommonTypes::e_UserDetailsError
      );

      void inviteUserInd (
                in TINACommonTypes::t_SessionId sessionId,
                in TINAUsageCommonTypes::t_IndId indId,
```

```
                in TINACommonTypes::t_PartyId reqPartyId,
                in TINACommonTypes::t_UserDetails userDetails
        ) raises (
                TINAUsageCommonTypes::e_PartyDomainError,
                TINAUsageCommonTypes::e_PartyError,
                TINACommonTypes::e_UserDetailsError
        );

        void announceSessionInd (
                in TINACommonTypes::t_SessionId sessionId,
                in TINAUsageCommonTypes::t_IndId indId,
                in TINACommonTypes::t_PartyId reqPartyId, // Requesting Party
                in TINACommonTypes::t_AnnouncementProperties announcement
        ) raises (
                TINAUsageCommonTypes::e_PartyDomainError,
                TINAUsageCommonTypes::e_PartyError,
                TINAUsageCommonTypes::e_AnnouncementError
        );


};    // interface i_PartyMultipartyInd

};    // module TINAPartyMultipartyIndUsage

#endif
```

E.7 Module TINAProviderVotingUsage

```
// File TINAProviderVotingUsage.idl

#ifndef tinaprovidervotingusage_idl
#define tinaprovidervotingusage_idl

#include "TINACommonTypes.idl"
#include "TINAUsageCommonTypes.idl"


module TINAProviderVotingUsage {

enum t_VoteResponse {
      NoVote,
      Agree,
      Disagree,
      Abstain
};

typedef short t_VoteValue;// use is Service dependant
                                  // could be used to weight the response,
                                  // but effect is service dependant

struct t_Vote {
      t_VoteResponse response;
      t_VoteValue value;          // may be ignored
```

```
};



enum t_VoteErrorCode {
      UnknownVoteError,
      VoteTooLate
};

exception e_VoteError {
      t_VoteErrorCode errorCode;
};



interface i_ProviderVotingReq {


      void voteReq(
                in TINACommonTypes::t_ParticipantSecretId myId,
                in TINAUsageCommonTypes::t_IndId indId,
                in t_Vote vote
      ) raises (
                TINAUsageCommonTypes::e_UsageError,
                TINAUsageCommonTypes::e_IndError,
                e_VoteError
      );

};    // interface i_ProviderVotingReq

};    // module TINAProviderVotingUsage

#endif
```

E.8 Module TINAPartyVotingUsage

```
// File TINAPartyVotingUsage.idl

#ifndef tinapartyvotingusage_idl
#define tinapartyvotingusage_idl

#include "TINACommonTypes.idl"
#include "TINAUsageCommonTypes.idl"


module TINAPartyVotingUsage {


enum t_VoteResult {
      UnknownVoteResult,// I don't know what the result is, but
                                  // the voting has finished
      VoteAgreed,
      VoteNotAgreed
```

```
};


interface i_PartyVotingInfo {

onewayvoid voteInfo (
               in TINACommonTypes::t_SessionId sessionId,
               in TINAUsageCommonTypes::t_IndId indId,
               in t_VoteResult result
      );

};    // interface i_PartyVotingInfo

};    // module TINAPartyVotingUsage

#endif
```

E.9 Module TINAControlSRTypes

```
// File TINAControlSRTypes.idl

#ifndef tinacontrolsrtypes_idl
#define tinacontrolsrtypes_idl

#include "TINACommonTypes.idl"
#include "TINAUsageCommonTypes.idl"


module TINAControlSRTypes {

enum t_ControlType {
      NoControl,
      ReadPermission,
      WritePermission,
      Ownership
};

struct t_ControlDescription {
      boolean useDefaultControl;
      t_ControlType controlType;
               // only relevant if useDefaultControl==FALSE
};

struct t_ControlInfo {
      t_ControlDescription controlDescription;
      TINAUsageCommonTypes::t_PartyDetails controllerInfo;
      TINACommonTypes::t_ElementId controlledObject;
};

typedef sequence<t_ControlInfo> t_ControlInfoList;

struct t_DefaultControlInfo {
      t_ControlType controlType;
```

```
      TINACommonTypes::t_ElementId controlledObject;
};

typedef sequence<t_DefaultControlInfo> t_DefaultControlInfoList;

};    // module TINAControlSRTypes

#endif
```

E.10 Module TINAProviderControlSRUsage

```
// File TINAProviderControlSRUsage.idl

#ifndef tinaprovidercontrolsrusage_idl
#define tinaprovidercontrolsrusage_idl

#include "TINACommonTypes.idl"
#include "TINAUsageCommonTypes.idl"
#include "TINAControlSRTypes.idl"

module TINAProviderControlSRUsage {

interface i_ProviderControlSRReq {


void setControlReq(
      in TINACommonTypes::t_ParticipantSecretId reqPartySecretId,
      in TINACommonTypes::t_PartyId controllerPartyId,
      in TINACommonTypes::t_ElementId controlledId,
      in TINAControlSRTypes::t_ControlDescription control
) raises (
      TINAUsageCommonTypes::e_UsageError,
      TINAUsageCommonTypes::e_PartyError
);

};    // interface i_ProviderControlSRReq

};    // module TINAProviderControlSRUsage

#endif
```

E.11 Module TINAPartyControlSRUsage

```
// File TINAPartyControlSRUsage.idl

#ifndef tinapartycontrolsrusage_idl
#define tinapartycontrolsrusage_idl

#include "TINACommonTypes.idl"
#include "TINAUsageCommonTypes.idl"
#include "TINAControlSRTypes.idl"


module TINAPartyControlSRUsage {
```

```
interface i_PartyControlSRInd {

      void setControlInd (
                in TINACommonTypes::t_SessionId sessionId,
                in TINAUsageCommonTypes::t_IndId indId,
                in TINAControlSRTypes::t_ControlInfo controlInfo
      ) raises (
                TINAUsageCommonTypes::e_PartyDomainError,
                TINAUsageCommonTypes::e_PartyError
      );

};    // interface i_PartyControlSRInd



interface i_PartyControlSRInfo {

onewayvoid setControlInfo (
                in TINACommonTypes::t_SessionId sessionId,
                in TINAControlSRTypes::t_ControlInfo controlInfo
      );

};    // interface i_PartyControlSRInfo


};    // module TINAPartyControlSRUsage


#endif
```

Annex F: IDL Stream Binding and Communication Session

This annex contains all idl definitions for stream binding and communication session.

F.1 FILE: README .edits

This file contains an indexed list of edits to the stream-related IDL files. It contains edits made to the original files supplied by Stephanie Hogg, in order that they compile with our standard IDL compilers. This list is not particularly interesting, and newer edits to these files are documented in the main README.edits file, (see Annex B.1). This file is not longer used for recording edits to these files. It is included here as the IDL still contains edit tags of the format: $$s000$$. (The ‘s’ denotes stream edits.) Since these have not been removed from the IDL, then I felt it would be worhtwhile showing the edits that they represent.

```
// README.edits for stream idl

// DO NOT USE THIS FILE.

// Any additional edits to the Ret-RP streams IDL should be documented in the
// main README.edits file, along with the edits for all the other sections.

// DO NOT ADD EDITS TO THIS FILE.

s008  Richard Westerga   061097   major
      Changed name TINAPartyCommS.idl and module to
      TINAPartyCommSUsage following naming conventions.

s007  Richard Westerga   031097   major
      Due to name change of TINASBCommSCommonTypes (two 'm'
      in stead of one) file and
      module, changed scoping of types accordingly in the
      rest of the idl files.

s006  Patrick Farley     281097   major
      File: TINASBComSCommonTypes.idl
      removed module m_STATE under Stephanie's instructions


s005  Patrick Farley     171097   major
      in file TINAStreamCommonTypes.idl
      removed  typedef TINACommonTypes::t_Interface t_SIRef
      changed all occurances of t_SIRef to Object (NEO bug)

s004  Patrick Farley     151097   major
      in file TINASBComSCommonTypes.idl
      changed the names:
      type  -->  notifyType
      operational    --> operationalState
      usage -->  usageState
      administrative -->  administrativeState
      source --> sourceId
      OffDutty-> OffDuty
```

```
s003  Richard Westerga   090997    major
      in file TINASBComSCommonTypes.idl
      removed t_Interface definition.

s002  Richard Westerga   090997    major
      in file TINAStreamCommonTypes.idl
      removed typedefs that only re-scope same typename

s001  Richard Westerga   090997    major
      in file TINASBComsCommonTypes.idl
      in file TINAStreamCommonTypes.idl
      removed all #include statements, included actual files
```

F.2 Module TINASBCommSCommonTypes

```
#ifndef _tinasbcoms_commontypes_idl
#define _tinasbcoms_commontypes_idl

// FILE:
//   TINASBCommSCommonTypes.idl
// DESCRIPTION:
//   Common data types used for stream bindings and the
//   communication session.
//   Includes: names, attributes, SFEP names, NFEP Descriptions,
// AUTHOR:
//   Stephanie Hogg
// CREATION DATE:
//  19/08/97
// UPDATES:
//  09/09/97: Richard Westerga. Removed all #include statements and included the
actual file
//

// $$053$$ deleted #ifdef defined(PLATyTools ...

module TINASBCommSCommonTypes {

// #include "states.idl" $$s001$$

//   DESCRIPTION:
//      State Management Definitions.
//
//   COMMENTS: Based in TINA Defs file (deviations are commented)

/* 1. Generic Attributes ------------------------------------- */

/* 1.1. State Attributes ------------------------------------- */

// module m_STATE {

enum t_OperationalState {/* single-valued and read-only */
  Disabled,    /* resource totally inoperable and unable to
                           provide service to the users */
```

```
  Enabled      /* resource partially or fully operable and
                          available for use */
};

enum t_UsageState {/* single-valued and read-write */
  /* Not all values are applicable to every class of managed object */
  Idle,                /* resource not currently in use */
  Active,              /* resource in use,
                          and with sufficient spare operating capacity */
  Busy,                /* resource in use, but no spare operating capacity */
  Reserved             /* resource reserved. This is NOT an ISO state */
};

enum t_AdministrativeState {/* single-valued and read-write */
  /* Not all values are applicable to every class of managed object */
  Locked,             /* prohibited from performing services for its users */
  ShuttingDown,        /* permitted to existing instances of use only */
  Unlocked             /* permitted to perform services for its users.
                          This is independent of its inherent operability */
};

struct t_ManagementState {
  t_OperationalState operationalState; // $$s004$$
  t_UsageState usageState; // $$s004$$
  t_AdministrativeState administrativeState; // $$s004$$
};

/* 1.2. Status Attributes (qualify the state attribute) -------- */

enum AlarmStatus {/* set-valued and read-write */
  UnderRepair,         /* resource currently being repaired */
  Critical,            /* some critical alarms have not yet been cleared */
  Major,               /* some major alarms have not yet been cleared */
  Minor,               /* some minor alarms have not yet been cleared */
  AlarmOutstanding     /* see additional attributes */
};

enum ProceduralStatus {/* set-valued and read-write */
  InitializationRequired, /* the resource requires
                             initialization to be invoked by the manager */
  NotInitialized,      /* the resource initializes ifself autonomously */
  Initializing,        /* initialization procedure not yet completed */
  Reporting,           /* the resource is notifying the results of an
                          operation. Its operational state is Enabled */
  Terminating          /* the resource is in termination phase */
};

enum AvailabilityStatus {/* set-valued and read-only */
  InTest,              /* the resource is undergoing a test procedure */
  Failed,              /* the resource has an internal fault.
                          Its operational state is Disabled */
  PowerOff,            /* the resource is not powered on.
```

```
                              Its operational state is Disabled */
    OffLine,                 /* the resource requires to be placed on-line.
                              Its operational state is Disabled */
    OffDuty,                // $$s004$$
                             /* the resource has been made inactive internally.
                              Its operational state is Disabled or Enabled */
    Dependency,              /* the resource cannot operate because some other
                              resource on which it depends is unavailable.
                              Its operational state is Disabled */
    Degraded,                /* but its operational state is Enabled */
    NotInstalled,            /* the resource represented by the managed
                              object is not present or is incomplete.
                              Its operational state is Disabled */
    LogFull                  /* log full condition (see Rec. X.735) */
  };

  enum ControlStatus {/* set-valued and read-write */
    SubjectToTest,           /* available for users, but tests may be
                              conducted on it */
    PartOfServicesLocked,    /* administrative state = Unlocked */
    ReservedFortest,         /* administrative state = Locked */
    Suspended                /* administrative state = Unlocked */
  };

  enum StandByStatus {/* set-valued and read-only */
    /* its value is only meaningful when the back-up relationship
       role exists (see Rec. X.732) */
    HotStandBy,              /* Not providing service, but operating in synchronism
                              with the resource that is to be backed-up */
    ColdStandBy,             /* Not providing service. Take-over
                              requires some initialization activity */
    ProvidingService         /* */
  };

  /* From ETSI/NA4, Network Level View: ServiceState ------------- */
  /* ServiceState values are defined as a combination of OperationalState,
     UsageState, AdministrativeState, AvailabilityStatus and ControlStatus */

  enum ServiceState {
    Planned,
    InServiceAssignedBusy,
    InServiceAssignedActive,
    InServiceReserved,
    InServiceSpare,
    UnavailableFaultyAssigned,
    UnavailableFaultyReserved,
    UnavailableFaultySpare,
    UnavailableLockedAssigned,
    UnavailableLockedReserved,
    UnavailableLockedSpare,
    UnderTestAssigned,
    UnderTestReserved,
```

```
    UnderTestSpare,
    CeasingShuttingDown,
    CeasingShutDown,
    Decommissioned
  };

  //}; // module m_STATE

  // #include "naming.idl" $$s001$$

  // DESCRIPTION:
  //
  // IDL type definitions for TINA naming conventions
  //
  // TINA name is a sequence of strings, each having format
  // "attribute=value".  Both the attribute names, and value
  // semantics have to be documented for each named object when such
  // object is defined.
  //
  // NOTE: Possible typedefs for various object name types must be
  // done where the object is specified, NOT HERE!
  // AUTHOR:
  //   Jarno Rajahalme/Natarajan Narayanan
  // CREATION DATE:
  //  xx/08/97
  // UPDATES:
  //  21/08/97 by Stephanie Hogg
  //    brought into line with Ret cross compiler guidelines
  //    expanded comments to include file name, author, updates
  //    added t_Identifier and t_TypeId parameters
  //

  typedefsequence<string>t_TinaName;
  typedefsequence<t_TinaName>t_TinaNameList;
  typedefstring                    t_TinaNameAttribute;
  typedefstring                    t_TinaNameValue;

  // General identifiers for names (instances) and types
  typedef t_TinaName t_Identifier;
  typedef string t_TypeId;
  typedef string t_ObjectIdentifier;
  typedefsequence<t_ObjectIdentifier>t_ObjectIdentifierList;

  // #include "attribute.idl" $$s001$$

  // DESCRIPTION
  //
  // Attribute Data Types
  //
  // An attribute consists of a pair of an identifier and a
  // value. The identifier identifies the semantics of the attribute,
  // and in particular, the actual data type of the value.
```

```
//
// The "tag" is defined as t_TinaName to follow the TINA naming
// conventions. The value is defined as any to allow carriage of
// IDL structured values without explicit parsing etc. (which would
// be required, if e.g. string would be used).
//
// AUTHOR:
// Created by Jarno Rajahalme
// UPDATES:
// 97/08/06 by Frank Steegmans
//   Replace 'TinaAttrib' with 'Attrib'
//  21/08/97 by Stephanie Hogg
//      brought into line with Ret cross compiler guidelines
//      expanded comments to include file name
//

//#include "naming.idl"

struct t_Attrib {
      //t_TinaName  id;
      t_ObjectIdentifier  id;
      any          value;
};

//
// Attribute list
//
typedef sequence <t_Attrib> t_AttribList;
typedef t_ObjectIdentifierList t_AttribIdList;
//typedef t_TinaNameList t_AttribIdList;

// #include "notifytypes.idl" $$s001$$

// DESCRIPTION:
//   Notification Types
//   General notification related types.
//   These data types describe event type information.
//   They are derived from TMN event report classes.
//   These structures assume that all notifications need
//   an identifier, a type, an origin identification and
//   origin type.  All other parameters are described by
//   a sequence of attributes.
// AUTHOR:
//   Stephanie Hogg
// CREATION DATE:
//  20/08/97
// UPDATES:
//

// #include "attribute.idl"

// General status descriptions
```

```
typedef unsigned long t_NotifyId;
typedef t_Identifier t_OriginatorId;
typedef t_TypeId t_OriginatorType;
typedef t_TypeId t_NotifyType;
typedef sequence<t_NotifyType> t_NotifyTypeList;
typedef t_AttribList t_StatusInfo;

// Notification related data types
struct t_Notification
  {
      t_NotifyId id;
      t_NotifyType notifyType; // $$s004$$
      t_OriginatorId sourceId; // $$s004$$
      t_OriginatorType sourceType;
      t_StatusInfo info;
  };
typedef sequence<t_Notification> t_NotificationList;

// These four parameters should uniquely identify a notification
struct t_NotifyIdentifier
  {
      t_NotifyId id;
      t_NotifyType notifyType; // $$s004$$
      t_OriginatorId sourceId; // $$s004$$
      t_OriginatorType sourceType;
  };
typedef sequence<t_NotifyIdentifier> t_NotifyIdentifierList;

// #include "sfep.idl" $$s001$$

// DESCRIPTION:
//
// Stream Flow End Point type definitions
// Currently only includes SFEP name related definitions
//
// AUTHOR:
//   Jarno Rajahalme
// CREATION DATE:
//  xx/08/97
// UPDATES:
//  21/08/97 by Stephanie Hogg
//      brought into line with Ret cross compiler guidelines
//


// #include "naming.idl"

//
// At the moment only naming of SFEPs is included here (nothing else
// is required by the TCSM).
//
```

```
typedef t_TinaName t_SFEPName;
typedef sequence<t_SFEPName> t_SFEPNameList;

// #include "media.idl" $$s001$$

// DESCRIPTION:
//   Media Types
//   Media type related data structures.
//   These include general type descriptors based on attributes,
//   and media type descriptions which relate to SFEPs and SFCs types.
// AUTHOR:
//   Stephanie Hogg
// CREATION DATE:
//  18/08/97
// UPDATES:
//

// #include "attribute.idl"

// General type descriptor
struct t_TypeDesc
{
     t_TypeId   id;        // type of type
     t_AttribList desc;    // type attributes list
};
typedef sequence<t_TypeDesc> t_TypeDescList;

struct t_TypeChangeDesc
{
     t_TypeId   id;        // type of type
     t_AttribList idAttribs;   // identifying attributes
     t_AttribList newAttsVals;  // new attributes or current attribs' values
     t_AttribIdList oldAttribs; // attributes to remove
};
typedef sequence<t_TypeChangeDesc> t_TypeChangeDescList;

// Media descriptors
typedef t_TypeDesc t_MediaDesc;
typedef t_TypeDescList t_MediaDescList;
typedef t_TypeChangeDesc t_MediaChangeDesc;
typedef t_TypeChangeDescList t_MediaChangeDescList;

// #include "sfepcoms.idl"  $$s001$$

// DESCRIPTION:
//   SFEP Communication Session Descriptor
//   Gives the SFEP description structure used by the communication
//   session.  This structure is included in the service session
//   descriptor.
// AUTHOR:
//   Stephanie Hogg
// CREATION DATE:
```

```
//  18/08/97
// UPDATES:
//

// #include "sfep.idl"
// #include "states.idl"
// #include "media.idl"

enum t_SFEPDirection { SFlowSink, SFlowSource };
// typedef /*CORBA::*/Object t_Interface; $$s003$$
//typedef m_STATE::t_AdministrativeState t_AdministrativeState;

struct t_SFEPComDesc {
      t_SFEPName name; // Unique SFEP id for terminal/local domain
      t_SFEPDirection dir; // sink, source, etc.
      t_AdministrativeState adState; // Current state: active/inactive
      t_MediaDesc media; // High level typing, qos parameters
      Object ifRef; // Associated TCSM interface ref $$s003$$
                           // this needs to be cast to appropriate if type
};
typedef sequence<t_SFEPComDesc> t_SFEPComDescList;


};

#endif
       // _tinasbcoms_commontypes_idl
```

F.3 Module TINAConSCommSCommonTypes

```
#ifndef _TINAConSCommSTypes_MODULE_
#define _TINAConSCommSTypes_MODULE_

// FILE:
//   TINAConSCommSCommonTypes.idl
// DESCRIPTION:
//   The common types module for the Communication
//   and Connectivity Sessions.
//   This includes the NFEP parameters defined previously by
//   Frank and Natarajan.  NFC naming data types have been added.
//   Data types have been updated to include use data from the
//   TINASBCommSCommonTypes module.
// AUTHOR:
//   Stephanie Hogg (from the nfep.idl file of FSTE)
// CREATION DATA:
//   01/10/1997
// UPDATES:
//
// COMMENTS:
//   This module should possibly be combined with the
//   TINASBCommSCommonTypes module
//
```

```
#include "TINASBCommSCommonTypes.idl"

module TINAConSCommSCommonTypes {

typedef TINASBCommSCommonTypes::t_AttribList t_AttribList;
typedef TINASBCommSCommonTypes::t_TinaName t_TinaName;
typedef t_TinaName t_FlowConnName;
typedef sequence <t_FlowConnName> t_FlowConnNameSeq;
typedef t_FlowConnName t_NFCName;
typedef t_FlowConnNameSeq t_NFCNameList;

#ifndef _NFEP_IDL_
#define _NFEP_IDL_

//
// Created by Frank Steegmans (1997)
// History:
// 97/08/08 initial contribution by Frank Steegmans
//


//#include "attribute.idl"// t_AttribList

// The datastructures defined in this file are compliant with the NRA v3.0 and
NRIM v2.1
// Minor additions have been done for passing the information in operations.
// Context:
// NFEP = Network Flow End Point (end point of a Network Flow Connection (NFC))
// An ANfep is the abstract superclass from which both the NfepPool and the
// plain NFEP are derived.
// The NFEP represents a specific Termination Point in a specific Layer Network.
// This is either a TP or TPpool. LNW specific details may be passed in the
// attribute list and will only be interpreted by the LNC level.
// However the t_LNType has to be specified and be different from 'NotDefined'.
// The NFEP pool is a collection of NFEPs and NFEP pools. The t_LNType may be
// specified if all contained NFEPs and NFEP pools are of the same t_LNType.
// (This is for processing optimizations.)

//module m_NFEP {

typedef t_TinaName t_NfepName;

typedefstring  t_LNType;

      // Folowing list of predefined LN types is not exhaustive and may be
expanded
      // in the future. This is the main reason for using a string as identifier
      // and not an enum. The reason why a string and not an ordenary number is
      // that it allows a bigger name space to be used.
      // Therefore, following constants should be moved to some
      // implementation specific file.
```

```
const t_LNType  ND_LNType="NotDefined";
      // NFEP pool containing NFEPs refering to different LNWs
const t_LNType  IP_LNType="IP";
const t_LNType  ATM_VC_LNType="ATM_VC";
const t_LNType  ATM_VP_LNType="ATM_VP";


enum t_ANfepType {
  NFEPPool, NFEP
};

enum t_NfepDir {
  receiveNfep,
  transmitNfep,
  receiveAndTransmitCapabilty
};


// The following structure represent the nfep datastructures.
// It contains the attributes for the two derived classes because it is not
// possible to do any forward declarations in IDL except in a 'struct'.
// Otherwise the structure would have looked like the one defined in the
// comments below the actual structure.
// The attribute list of a nfep will contain more information related to
// the TP the nfep represents and is therefore layer network dependent.
// E.g. required Traffic and QoS parameters for the selected codecs.
// Hence, the symantics have to be specified in layer network dependent
// idls.
// Representing an ANfep as just an attribute list has not been done
// because it also requires that the syntax is specified somewhere else.

struct t_ANfep {
      t_NfepNamename;
      t_LNType  lnType;
      t_ANfepTypenfepType;

      // Following two attributes are only valid if the nfepType is NFEP
      t_NfepDir dir;
      t_AttribListattributes;

      // Following attribute is only valid if the nfepType is NFEPPool
      sequence<t_ANfep> pool;
};

/*****************************************************************

Following structure would be a better representation of the inheritance
relationship but it is not compliant with the IDL grammar.

struct t_Nfep {
      t_NfepDir dir;               // Nfep directionality
      t_AttribListattributes;// Nfep attributes
```

```
};

struct t_ANfep {
      t_NfepName    name;
      t_LNType      lnType;
      union t_NfepSpecialization
               switch (t_NfepSubClass) {
               case Pool: sequence<t_ANfep> pool;
               case Plain: t_Nfep       nfep;
      } nfepSpec;
};

*************************************************************************/

typedef sequence<t_ANfep> t_ANfepList;

typedef sequence<t_NfepName> t_NfepNameList;

//}; // m_NFEP

#endif
       // _NFEP_IDL_

}; // End of TINAConSCommSCommonTypes module

#endif
      // _TINAConSCommSTypes_MODULE_
```

8.1 Module TINACommSCommonTypes

```
#ifndef _TINACommSCommon_MODULE_
#define _TINACommSCommon_MODULE_

// FILE:
//   TINACommSCommonTypes.idl
// DESCRIPTION:
//   The common types module for the Communication Session
//   This includes the capability parameters defined previously by
//   Jarno Rajahalme
//   Capabilities have been updated to include use data from the
//   TINASBCommSCommonTypes module.
//   New data types to support the expanded i_TerminalFlowControl
//   interface have been added.
// AUTHOR:
//   Stephanie Hogg (from the Capabilities.idl file of Jarno R.)
// CREATION DATA:
//   01/10/1997
// UPDATES:
//

#include "TINASBCommSCommonTypes.idl"
#include "TINAConSCommSCommonTypes.idl"
```

```
module TINACommSCommonTypes {

typedef TINASBCommSCommonTypes::t_ObjectIdentifier t_ObjectIdentifier;
typedef TINASBCommSCommonTypes::t_AttribList t_AttribList;
typedef TINASBCommSCommonTypes::t_TinaName t_TinaName;
typedef TINASBCommSCommonTypes::t_SFEPName t_SFEPName;
typedef TINAConSCommSCommonTypes::t_NFCName t_NFCName;
typedef TINAConSCommSCommonTypes::t_NFCNameList t_NFCNameList;
typedef TINAConSCommSCommonTypes::t_ANfep t_ANfep;
typedef TINAConSCommSCommonTypes::t_ANfepList t_ANfepList;

#ifndef _CAPABILITY_IDL_
#define _CAPABILITY_IDL_

// IDL definition for terminal capability information.  This is
// based on ASN.1 code given in ITU-T H.245, with following
// extensions:
//
// 1. Capability set can also have session and transport protocol
// capabilities.
//
// 2. Each codec, protocol, etc. capability is explicitly
// identified with ASN.1 OBJECT IDENTIFIER. This will identify the
// semantics of the capability and the semantics of the associated
// attributes. This is like the non-standard capability in H.245.
//
// 3. A capability can require other capabilities. For example, a
// codec may only be supported over RTP session protocol over UDP
// transport. Another instance of the same codec could be
// available over ATM interface only, etc.
//
// 4. The same data structure is used to indicate the desired mode
// of operation, when the capabilities are selected. No separate
// data structures exist for mode selection.
//
// See H.245 section 7.2.1 for overview on terminal capabilities
//
// Problems:
//
// 1. Usage of 'any' (via t_AttribList)
//
// HISTORY
//
// 07/24/97 Jarno Rajahalme  Initial Contribution
// 08/06/97 Frank Steegmans  IDL Syntax fix
// 09/29/97 Jarno Rajahalme  Added use of t_ObjectIdentifier,
//     changed dependencies (w/ role tag),
//     added instance identifier
// 10/01/97 Stephanie Hogg   Updated data types to fit in module
//    structures.

//#include "naming.idl"  // t_ObjectIdentifier
```

```
//#include "attribute.idl" // t_AttribList

//
// Type for capability unique semantic identifier.  This should
// from a subtree of ASN.1 OBJECT IDENTIFIER name space
//
typedef t_ObjectIdentifier t_CapabilityId;

//
// Type for terminal unique capability key
//
// The instance identifier shares the same type definition
//
typedef unsigned long t_CapabilityKey;
typedef sequence<t_CapabilityKey> t_CapabilityKeyList;

//
// Capability dependencies
//

// Capability dependency, including the role tag for this capability.
// The role tag is a 32bit integer, and the numbering of roles starts
// always from 0 with increments of 1.
//
// The role tag is defined only in the scope of the capability having
// it in the dependency list. For example, if two codecs are dependent
// on the same multiplex, the multiplex has different role tag in each
// dependency. The semantics of the role is not "multiplex" in either
// dependency, but something like "stream" or "control". Even if the
// numerical value of the tag is the same, the semantics of the tag can
// be different. The semantics of the tag is defined by the codec having
// the dependency.
//
// The endpoints need to agree on the used role tag values. This allows
// proper matching of the dependencies by the network.
//
typedef unsigned long t_CapabilityRoleTag;

struct t_AlternativeDependencies {
 t_CapabilityRoleTag roletag;
 t_CapabilityKeyList alternatives;
};

// t_SimultaneousDependencies is a sequence of t_AlternativeDependencies.
// Semantics is such that one capability from each   = t_AlternativeDependencies
// must be selected. Each t_AlternativeDependencies list has a role tag,
// that defines the role of the dependency in a unique way depending on   = the
// protocol which the higher level capability is representing.
// Having multiple dependencies is useful to express dependency of a   = multi-
// channel ISDN link, for example.
//
typedef sequence<t_AlternativeDependencies> t_SimultaneousDependencies;
```

```
//
// Directionality of the capability.
//
// NOTE: This is not related to the directionality of an SFEP! Two
// SFEPs would be needed for e.g. video receive and transmit, even
// if only one capability with directionality
// 'receiveAndTransmitCapability' is indicated.
//
enum t_CapabilityDir {
  receiveCapability,
  transmitCapability,
  receiveAndTransmitCapabilty
};

//
// Capability
//
struct t_Capability {
  t_CapabilityId   name; // unique semantic identifier
  t_CapabilityKey  key;  // Terminal unique id
  t_CapabilityKey  instanceId; // Terminal unique id
  t_CapabilityDir  dir;  // Capability directionality
  t_AttribList     attributes; // Capability attributes
  t_SimultaneousDependencies dependencies; // Required capabilities
};
// list of capabilities:
typedef sequence<t_Capability> t_CapabilityList;

//
// Capability Combinations
//

//
// Terminal unique id for a CapabilityDescriptor.
//
// The number indicates preference of mode of operation by the
// terminal: descriptors with lower numbers are preferred (i.e. 0
// =3D=3D most preferred)
//
typedef unsigned long t_CapabilityDescriptorNumber;

//
// List of alternative capabilities. Any ONE of these can be done
// at one time - list items are mutually exclusive (within this
// list only)
//
typedef t_CapabilityKeyList t_AlternativeCapabilities;

//
// Capability Descriptor. This contains a terminal unique
// identifier for this set of capabilities and a set of
```

```
// capabilities supported simultaneously
//
struct t_CapabilityDescriptor {
  t_CapabilityDescriptorNumber       capabilityDescriptorNumber;
  sequence<t_AlternativeCapabilities> simultaneousCapabilities;
};

//
// Capability Set
//
struct t_CapabilitySet {
  sequence<t_CapabilityDescriptor> capabilityDescriptors;
  t_CapabilityList   capabilityTable;
};

#endif
      // _CAPABILITY_IDL_


// Notes:
// TFCs identified by a t_TFCName
// TFC branches identified by a t_CorrelationId
// Assuming: t_CorrelatinId will be a t_TINAName (needs to be unique for
// terminal as can be used without t_TFCName later - especially in TCon
// part) Assuming: t_TFCName will be a t_TINAName

typedef t_TinaName t_TFCName;
typedef t_TinaName t_CorrelationId;
typedef sequence<t_CorrelationId> t_CorrelationIdList;

struct t_SFEPCorrelation {
      t_SFEPName branch;
      t_CorrelationId correlator;
};
typedef sequence <t_SFEPCorrelation> t_SFEPCorrelationList;

struct t_NFCCorrelation {
      t_NFCName branch;
      t_CorrelationId correlator;
      t_ANfepList requiredNfeps;
};
typedef sequence <t_NFCCorrelation > t_NFCCorrelationList;

enum t_NFEPUpdate {
      TINAComS_NoNFEPChange,
      TINAComS_ModifyNFEP,
      TINAComS_NewNFEP
};

struct t_BranchUpdate {
      t_CorrelationId branch;
      t_NFEPUpdate reqMod;
```

```
      t_ANfepList requiredNfep;
};
typedef sequence <t_BranchUpdate> t_BranchUpdateList;

struct t_SFEPSelect {
      t_SFEPName sfep;
      t_CapabilityList caps;
};
typedef sequence <t_SFEPSelect> t_SFEPSelectList;

}; // End of TINACommSCommonTypes module

#endif
      // _TINACommSCommon_MODULE_
```

F.4 Module TINAStreamCommonTypes

```
#ifndef _tinastreamcommontypes_idl
#define _tinastreamcommontypes_idl

// FILE:
//   TINAStreamCommonTypes.idl
// DESCRIPTION:
//   TINA Stream Common Types Module
//   Common data types used for stream binding.
//   Includes: SFEP descriptions, SI Descriptions,
//             SB criteria types, SB type and media descriptions,
//             and SB return types
// AUTHOR:
//   Stephanie Hogg
// CREATION DATE:
//  18/08/97
// UPDATES:
//  09/09/97:Richard Westerga. Removed all #include statements
//             replaced them with the actual file
//

#include "TINACommonTypes.idl"
#include "TINASBCommSCommonTypes.idl"


module TINAStreamCommonTypes {

// #include "sbcriteria.idl" $$s001$$

// DESCRIPTION:
//   Stream Binding Criteria
//   Operational success criteria and recovery action and success
//   criteria for stream bindings.
// AUTHOR:
//   Stephanie Hogg
// CREATION DATE:
//  18/08/97
```

```
// UPDATES:
//

// PART I Stream binding success criteria

// A) Overall stream binding operation success criteria
enum t_SBSuccessCriteria
  {
      SBSuccessDefault, // Use default success criteria (SL specific)
      SBBestEffort,     // Best effort for each participant for each flow
      SBBEOnParties,    // Best effort to connect each party for all flows:
                // i.e. if cant connect party for all, dont connect
      SBBEOnFlows,   // Setup all flows: best effort for participants
                // i.e. failure if cant connect for at least one flow
                // SBAllOrNone, Must connect all parties for all flows
                // (fails if not)
      SBPartySpecific   // Use specified criteria for each party
  };

// B) Operation success criteria specified for a particular
//    stream binding member
enum t_ParticipantSuccess
  {
      ParticipantDefault,// Use default for stream binding
      ParticipantMustAll,// Participant must be connected for all flows
                                  // Otherwise SB operation will fail
      ParticipantMustBE, // Particpant must be connected for >= 1 flow
                                  // Otherwise SB operation will fail
      ParticipantBE, // Best effort to connect as many flows as possible
      ParticipantBEAll // Best effort to connect all flow: do not connect
                          // any if can not connect all flows
  };

// C) Operation success criteria specified for a particular
//    stream binding stream flow connection
enum t_SFCSuccess
  {
      SFCSuccessDefault, // Use default for stream binding
      SFCSuccessBE, // Best effort to setup this SFC
      SFCSuccessAll // Connect all SFEPs or setup fails
  };

// PART II Recovery Criteria
// Stream binding recovery criteria: Note the default may be set for
// all stream bindings by the Fault Management Context.
// A) Overall stream binding recovery action and success criteria
enum t_SBRecoveryCriteria
  {
      DefaultRecovery, // Use default overall recovery actions
      IgnoreFailure, // Take no actions on a full or partial failure
      ReestablishBE,// Try to reestablish: best effort on previous setup
      ReestablishBEOnParties, // Participant must have previous setup
```

```
                                    // or drop it from the binding
        RestablishBEOnFlows, // Recover flow to previous state or
                                    // drop it from stream binding
        ReestablishAll, // All flows and all participants as before
        DeleteAll, // Pull down rest of stream binding on partial failure
        PartySpecificRecovery, // Use specified criteria for each party
        FlowSpecificRecovery // Use criteria for Flow
    };

  // Overall recovery descriptor:
  struct t_SBRecover
    {
        boolean    valid;  // set true if recovery info is valid
        t_SBRecoveryCriteria criteria; // criteria
        unsigned short retries;// Retries for reestablishment
    };


  // B) Recovery action and success criteria specified for a particular
  //    stream binding member
  enum t_ParticipantRecovery
    {
        RecoverDefault,  // Use default overall recovery actions
        RecoverMustAll, // Participant must be recovered for all flows
                        // Otherwise SB recovery will fail
        RecoverMustBE,  // Particpant must be reconnected for >= 1 flow
                        // Otherwise SB recovery will fail
        RecoverBEAll,  // Best effort to recover all flow: do not reconnect
                        // any if can not reconnect all flows
        RecoverBE, // Best effort to recover as many flows as possible
        IgnoreParticipantFailure, // Ignore failure of this participant
        DeleteOnParticipantFailure // Delete SB if this participant fails
    };

  // B) Recovery action and success criteria specified for a particular
  //    stream binding stream flow connection
  enum t_SFCRecovery
    {
        RecoverSFCDefault, // Default specified for overall recovery
        RecoverSFCMustAll, // Must recover SFC to previous state or SB fails
        RecoverSFCMustBE,  // Must at least part recover SFC, or SB fails
        RecoverSFCBEAll,   // Best effort to recover SFC to previous state:
                           // Delete SFC if can not recover state
        RecoverSFCBE,      // Best effort to recover SFC to previous status
        IgnoreSFCFailure,  // Ignore part/full failure of this SFC
        DeleteOnSFCFailure // Delete SB on failure of this SFC
    };


  // #include "StreamDefinitions.idl" $$s001$$

  // Session elements
```

```
// typedef TINACommonTypes::t_ElementId t_ElementId; $$s002$$
// typedef TINACommonTypes::t_ElementIdList t_ElementIdList; $$s002$$

// Stream binding session identifiers
typedef TINACommonTypes::t_ElementId t_SBId;
typedef TINACommonTypes::t_ElementIdList t_SBIdList;

// General interface reference
// typedef TINACommonTypes::t_Interface t_Interface; $$s002$$

// Stream Binding type identifier
typedef TINASBCommSCommonTypes::t_TypeId t_SBType;

// #include "sbrettypes.idl" $$s001$$

// DESCRIPTION:
//   Stream Binding Return Types
//   Types returned to describe the status of an stream binding
//   or stream binding element after creation or some operation.
//   Stream binding elements include SFEPs, SIs, participants and
//   SFCs.
// AUTHOR:
//   Stephanie Hogg
// CREATION DATE:
//  18/08/97
// UPDATES:
//

// #include "StreamDefinitions.idl"

// Identifies requests being processed asynchronously
typedef TINASBCommSCommonTypes::t_TypeId t_RequestType;
struct t_RequestId
  {
      TINACommonTypes::t_ElementId party; // Id of requester
      unsigned long id; // request number
  };

// Error codes: set to what ever failure codes are
typedef unsigned short t_FailureCode;

// Local identifier to session identifier mapping
// Some elements (SFEPs,SFCs, SIs) have ids local to the terminal
// as well as to the session.  In many cases, the session member
// contributing the element needs to know the session id for later
// manipulations.
struct t_RetElementId
  {
      TINASBCommSCommonTypes::t_Identifier localId; // Local identifier
      TINACommonTypes::t_ElementId id; // Session identifier
  };
typedef sequence<t_RetElementId> t_RetElementIdList;
```

```
// Basic element failure descriptor
struct t_FailedElementDesc {
      t_RetElementId id; // Identifier (SFEP/SI/SFC)
      unsigned long errorCode; // Reason for failure
      unsigned long state; // Operational or Activation status as approp
  };
typedef sequence<t_FailedElementDesc> t_FailedElementDescList;

// Complex element success descriptor
struct t_SBElementSuccess
  {
      t_RetElementId id; // Participant or stream flow session identifier
      boolean all; // set if all flows/flow end points added successfully
      t_RetElementIdList succeeded;
                // List of SFEPs/Flows connected successfully (optional)
      t_FailedElementDescList failed;
                // List of SFEPs/Flows not connected, reason, state (opt)
  };
typedef sequence<t_SBElementSuccess> t_SBElementSuccessList;

// Complex element failure description
struct t_SBElementFailure
  {
      t_RetElementId id; // Participant or SFC session ID
      boolean all; // If set: All connections failed
      t_FailedElementDescList failed; // list of failed elements (opt)
  };
typedef sequence<t_SBElementFailure> t_SBElementFailureList;

// State of the stream binding at the completion of a binding operation
// This is the same for all SBFS types
struct t_SBBindState
  {
      t_SBId id; // binding identifier
      boolean all; // set true if all flows setup correctly
      t_SBElementSuccessList success; // opt: successfully setup elements
      t_FailedElementDescList failed; // opt: list of failed elements
  };

enum t_ProblemType
 {
      Problem_SingleElement,
      Problem_Parameters,
      Problem_ComplexFailure
 };

union t_ReqProblem switch (t_ProblemType)
 {
      case Problem_SingleElement:  TINACommonTypes::t_ElementId  problemEl;
                          // Id of problem element
      case Problem_Parameters:     sequence<string>   problemParams;
```

```
                        // Id of problem paramaters (if not elements)
      case Problem_ComplexFailure: t_FailedElementDescList problemInfo;
                        // Complex problem: list of failed elements
                        // Used for communication setup errors
 };

// #include "sfepservs.idl" $$s001$$

// DESCRIPTION:
//   SFEP Service Session Descriptor
//   Gives the SFEP description structure used by the service
//   session.  This structure is based on the communication session
//   descriptor. It also includes a local session identifier which
//   is set by the service session, a t_SFEPBindName tag which is
//   is set by the terminal and used by the service logic to decide
//   which SFEPs need to be connected. Finally it includes a reference
//   to the Stream Interface the SFEP is associated with.  When there
//   is DPE support of stream binding, we assume this will be needed
//   for Data control support.
// AUTHOR:
//   Stephanie Hogg
// CREATION DATE:
//  18/08/97
// UPDATES:
//

// #include "StreamDefinitions.idl"

// SFEP service session identifiers
typedef TINACommonTypes::t_ElementId t_SFEPId;
typedef TINACommonTypes::t_ElementIdList t_SFEPIdList;

// SI names (required here): local to terminal or user domain
typedef TINASBCommSCommonTypes::t_Identifier t_SIName;
typedef sequence <t_SIName> t_SINameList;

// Stream interface references:
//   We use the general interface reference as we don't know what
//   they will actually look like.
//typedef TINACommonTypes::t_Interface t_SIRef; // $$s005$$
// t_SIRef replaced with Object

// SFEBBindName: the binding tag used to determine which SFEPs
//   need to be connected.
typedef string t_SFEPBindName;

struct t_SFEPServDesc
  {
      t_SFEPId id; // Session identifier for SFEP (if known - optional)
                        // Note: this is added for Flow SBFS and SI reg
      TINASBCommSCommonTypes::t_SFEPComDesc desc;
      t_SFEPBindName tag; // Tag: used for determining SB structure
```

```
      t_SIName si; // Associated SI
      Object siIfRef; // SI reference: optional, NIL = not set // $$s005$$
  };
typedef sequence<t_SFEPServDesc> t_SFEPServDescList;


// #include "sidesc.idl"  $$s001$$

// DESCRIPTION:
//   SI Descriptor
//   SI descriptor is basically a list of SFEP service descriptors
//   with SI interface reference, overall state information, and
//   optional synchronisation information.
// AUTHOR:
//   Stephanie Hogg
// CREATION DATE:
//  18/08/97
// UPDATES:
//
// #include "sfepservs.idl"

// Stream Interface session identifiers
typedef TINACommonTypes::t_ElementId t_SIId;
typedef TINACommonTypes::t_ElementIdList t_SIIdList;

// The synchronisation descriptor gives a list of SFEPs to be
// synchronsied and the type of synchronisation required.
struct t_SyncDesc
  {
      TINASBCommSCommonTypes::t_SFEPNameList synchedFlows;// Flows (SFEPs) to be
synched
      TINASBCommSCommonTypes::t_TypeDesc synchDesc;
                                            // Description of type of synch
  };

// Stream Interface Descriptor:
//    Describes the stream interface in terms of:
//    - a local name
//    - a session id (this is assigned by session, field need not
//                be filled by the terminal or user domain)
//      - interface reference: a dpe interface supporting the stream
//              (used if the DPE supports streams)
//    - optional synchronsiation information
//    - list of SFEPs (service level descriptors)
struct t_SIDesc
  {
      t_SIName name; // Unique SIid for terminal/local domain
      t_SIId id;// Session identifier for SI (if known - optional)
      Object siRef; // SI reference: optional, NIL = not set // $$s005$$
      TINACommonTypes::t_ElementId sbMember; // SB member controlling SI (opt)
      t_SBType si_type;// Overall SI type
      t_SyncDesc siSync; // SFC synchronising info (optional)
```

```
        t_SFEPServDescList sfeps; // SFEP descriptions
    };
  typedef sequence <t_SIDesc> t_SIDescList;


  };

  #endif
         // _tinastreamcommontypes_idl
```

F.5 Module TINAPaSBTypes

```
#ifndef _tinapasbtypes_idl_
#define _tinapasbtypes_idl_

// FILE:
//   TINAPaSBCommon.idl
// DESCRIPTION:
//   Common data types used for participant oriented stream binding.
//   Includes: TINAStreamCommon types, participant descriptions
//             and participant stream binding descriptions
// AUTHOR:
//   Stephanie Hogg
// CREATION DATE:
//  18/08/97
// UPDATES:
//  09/09/97:Richard Westerga. Removed #include statements
//             replaced them with the actual file.
//

#include "TINAStreamCommonTypes.idl"
#include "TINAUsageCommonTypes.idl"

module TINAPaSBTypes
{

// #include "sbparticipant.idl" $$s001$$

// DESCRIPTION:
//   Stream Binding Participant Descriptor
//   The participant descriptor structure describes the relation
//   of a session member in regard to a stream binding. The descriptor
//   includes the session member identifier, their overall role,
//   control relationship to stream binding, initial administrative
//   state, success criteria for operations and recovery, and
//   any additional information that affects their participation.

// AUTHOR:
//   Stephanie Hogg
// CREATION DATE:
//  18/08/97
// UPDATES:
//
```

```
// #include "StreamCommonTypes.idl"

typedef TINACommonTypes::t_ElementId t_ParticipantId;
typedef TINACommonTypes::t_ElementIdList t_ParticipantIdList;

// Criteria for success of operation and success of recovery
struct t_ParticipantCriteria
  {
      TINAStreamCommonTypes::t_ParticipantSuccess success;
      TINAStreamCommonTypes::t_ParticipantRecovery recoverAct;
  };

// Participant's criteria: criteria for a specific participant
struct t_PCriteria
  {
      t_ParticipantCriteria myCriteria;
      TINACommonTypes::t_ElementId myId;
  };
typedef sequence<t_PCriteria> t_PCriteriaList;

// Participation role requested of session member
enum t_SBFSParticipationType
  {
      SBFSApplicationSpecific, // Application will determine role
      SBFSSinkAll, SBFSSourceAll, SBFSSinkSourceAll, // For all flows
      SBFSSinkSource, // General type: can sink or source flows
                         // as applicable to the participant
      SBFSAssociate,  // Resource that can support SB: e.g. bridge
      SBFSInitiator   // Initiator - when not direct participant
  };

// Control relationship of session member to stream binding
enum t_SBControlSRType // Typedef to generic control type when known
  {
      DefaultSBControl,
      OwnershipSBControl,
      ReadSBControl,
      WriteSBControl
  };

struct t_ParticipantDesc
  {
      t_ParticipantId id;// Participants id
      t_SBFSParticipationType standing; // Role information
      t_SBControlSRType control; // Control SR with SB: e.g. owner
      TINASBCommSCommonTypes::t_AdministrativeState state; // admin state:
active inactive
      t_ParticipantCriteria criteria; //opt: success/recovery critieria
      string addInfo; // Additional information for this participant
  };
typedef sequence <t_ParticipantDesc> t_ParticipantDescList;
```

```
// #include "sbdesc.idl" $$s001$$

// DESCRIPTION:
//   SB Descriptor
//   The Stream Binding Descriptor is basically a stream binding
//   type, some additional media information relating to that type
//   and a list of session members that can participate in the
//   stream binding.  As well, it includes the initial adminstrative
//   state of the overall binding plus the overall success and
//   recovery criteria.
// AUTHOR:
//   Stephanie Hogg
// CREATION DATE:
//  18/08/97
// UPDATES:
//

// #include "StreamCommonTypes.idl"
// #include "sbparticipant.idl"

struct t_PaSBFSDesc
  {
      TINAStreamCommonTypes::t_SBType sbType;
      TINASBCommSCommonTypes::t_MediaDescList media;
                // High level requested typing, qos parameters
      TINASBCommSCommonTypes::t_AdministrativeState adState;
      t_ParticipantDescList sbMembers;
      TINAStreamCommonTypes::t_SBSuccessCriteria criteria;
      TINAStreamCommonTypes::t_SBRecoveryCriteria recoveryActs;
  };

typedef t_PaSBFSDesc t_SBDesc;


};

#endif
       // _tinapasbtypes_idl_
```

F.6 Module TINAPartyPaSBUsage

```
#ifndef _TINA_PARTY_PASB_FS_IDL_
#define _TINA_PARTY_PASB_FS_IDL_

// FILE:
//   TINAPartyPaSBUsage.idl
// DESCRIPTION:
//  TINA Participant Oriented Stream Binding Feature Set
//  Party Module
//
// Interfaces and types needed to support the participant
// oriented stream binding feature set for users.
```

```
// AUTHOR:
//   Stephanie Hogg
// CREATION DATE:
//  20/08/97
// UPDATES:
//

#include "TINAPaSBTypes.idl"

module TINAPartyPaSBUsage {

// #include "PaSBCommonTypes.idl"
// #include "PaSBPartyErrorTypes.idl" $$s001$$

// DESCRIPTION:
//   Participant oriented stream binding party error types
//   The error codes and exception types associated with the
//   party part of the normal particiapant oriented stream binding.
// AUTHOR:
//   Stephanie Hogg
// CREATION DATE:
//  20/08/97
// UPDATES:
//

// #include "StreamCommonTypes.idl"

enum t_PaSBPartySetupErrors
   {
        PaSBPartySetup_InvalidSBId,          // Unknown SB
        PaSBPartySetup_InvalidSBOp,  // Invalid op for this SB
        PaSBPartySetup_UnknownSBType, // Unknown stream binding type
        PaSBPartySetup_UnknownMediaType,// Unknown media type
        PaSBPartySetup_IncompatibleParameters, // Incompatible params:
               // E.g. media type and sb type, media params with media type
        PaSBPartySetup_UnknownCriteria,// Unknown criterion // $$055$$
        PaSBPartySetup_InvalidCriteria, // Not valid for this SB
        PaSBPartySetup_UnsupportedCriteria, // Not supported by session //
$$055$$
        PaSBPartySetup_QoSCannotBeMet,          // QoS requirements not met
        PaSBPartySetup_InsufficientResources // No resources for connection
   };

exception e_PaSBPartySetupError
   {
      t_PaSBPartySetupErrors errorCode;// Error
      TINACommonTypes::t_ElementId  problemEl;// Element causing problem
               // Valid for appropriate error codes, e.g. Invalid participant
               // Invalid SB (returns given SBId), invalid participant type
   };
```

```
enum t_PaSBPartyExeErrors
   {
        PaSBPartyExe_InvalidSBId,          // Unknown SB
        PaSBPartyExe_InvalidSBOp, // Invalid op for this SB
        PaSBPartyExe_CannotComply,        // Can't comply - resource err
      PaSBPartyExe_InvalidState // Wrong state for this request
   };


exception e_PaSBPartyExeError
   {
      t_PaSBPartyExeErrors errorCode;// Error
      TINACommonTypes::t_ElementId  problemEl;// Element causing problem
               // Valid for appropriate error codes, e.g. Invalid participant
               // Invalid SB (returns given SBId)
   };

// #include "i_PartyPaSBExe.idl" $$s001$$

// DESCRIPTION:
//   i_PartyPaSBExe interface
//   The participant oriented stream binding exe interface
//   Makes exe requests on stream binding members.
//   The set of exe requests is simpler than the full set of
//   stream binding requests as the same set of exes can support
//   a number of request operations.
//
//   The exe interface allows a provider to request a session member
//   join - become a stream binding member
//   leave - end participation in the stream binding
//   modify - modify participation  i.e. change MediaTypes supported
//   modify criteria - modify their local success and recover criteria
//
// AUTHOR:
//   Stephanie Hogg
// CREATION DATE:
//  20/08/97
// UPDATES:
//

// #include "PaSBCommonTypes.idl"
// #include "PaSBPartyErrorTypes.idl"

interface i_PartyPaSBExe
{
      // This interface stream binding responses and notifications to
      // parties or resources which are partipants in a stream binding

      // Join: Request to a participant to join the SBFS and
      // return supporting SI reference and information
      void joinPartyPaSBExe(
               in TINACommonTypes::t_SessionId sessionId, // Session Id
```

```
                in TINAStreamCommonTypes::t_SBId sbId, // ID stream binding to
join
                in TINAStreamCommonTypes::t_SBType reqType,// Overall type
                in TINASBCommSCommonTypes::t_MediaDescList media,// Overall req
of flows
                in TINAPaSBTypes::t_ParticipantIdList others,// other members id
                in TINAPaSBTypes::t_ParticipantDesc reqParticipation,
                          // Specification of required participation
                          // for this session member
                in TINAStreamCommonTypes::t_RequestId reqId,// op id used for
later info ops.
                out TINAStreamCommonTypes::t_SFEPServDescList participantSIs)
                          // Description of type: include SI ref and
                          // description, deviation from requested type
                raises ( TINAUsageCommonTypes::e_PartyDomainError,
                           e_PaSBPartySetupError );

      // Leave: Request to a participant to leave the SBFS
      void leavePartyPaSBExe(
                in TINACommonTypes::t_SessionId sessionId, // Session Id
                in TINAStreamCommonTypes::t_SBId sbId, // ID stream binding to
leave
                in TINAStreamCommonTypes::t_RequestId reqId) // op id used for
later info ops.
                raises ( TINAUsageCommonTypes::e_PartyDomainError,
                           e_PaSBPartyExeError );

      // Modify: Request to a change participation, including QoS,
      // Supported media types (large scale changes)
      void modifyPartyPaSBExe(
                in TINACommonTypes::t_SessionId sessionId, // Session Id
                in TINAStreamCommonTypes::t_SBId sbId, // ID stream binding
                in TINASBCommSCommonTypes::t_MediaDescList newTypes, // Media
types to add or modify
                in TINASBCommSCommonTypes::t_MediaDescList oldTypes, // Media
types to be removed
                in TINASBCommSCommonTypes::t_MediaChangeDescList modTypes,//
Media types to modify
                in TINAStreamCommonTypes::t_RequestId reqId,// op id used for
later info ops.
                out TINAStreamCommonTypes::t_SFEPServDescList participantSIs) //
New participation info
                raises ( TINAUsageCommonTypes::e_PartyDomainError,
                           e_PaSBPartySetupError );

      // Change administrative status (i.e. activate/deactivate
      void changeStatePartyPaSBExe(
                in TINACommonTypes::t_SessionId sessionId, // Session Id
                in TINAStreamCommonTypes::t_SBId sbId, // ID stream binding
                in TINASBCommSCommonTypes::t_AdministrativeState state, // New
state for participant
                in boolean allFlows,// All sub types or listed sub types?
```

```
                in TINASBCommSCommonTypes::t_MediaDescList reqFlows,
                          // Media types to be deactivated, valid allFlows=false
                in TINAStreamCommonTypes::t_RequestId reqId)// op id used for
  later info ops.
                raises ( TINAUsageCommonTypes::e_PartyDomainError,
                           e_PaSBPartyExeError );

       // Change participation criteria (success and recovery)
       void modifyCriteriaPartyPaSBExe(
                in TINACommonTypes::t_SessionId sessionId, // Session Id
                in TINAStreamCommonTypes::t_SBId sbId, // ID stream binding
                in TINAPaSBTypes::t_ParticipantCriteria newPCriteria)
                raises ( TINAUsageCommonTypes::e_PartyDomainError,
                           e_PaSBPartySetupError );
  };

  // #include "i_PartyPaSBInfo.idl" $$s001$$

  //   i_PartyPaSBInfo interface
  //   The participant oriented stream binding information interface
  //   Allows status reports on a synchronous operations,
  //   the distribution of SIs (in a very simple way),
  //   the informs on the widthdrawals of SIs (and other elements)
  //   and the notification of communication errors
  ///  Based on the general stream interface.
  // AUTHOR:
  //   Stephanie Hogg
  // CREATION DATE:
  //  20/08/97
  // UPDATES:
  //

  // #include "i_GeneralStreamInfo.idl"

  // DESCRIPTION:
  //   i_GeneralStreamInfo interface
  //   A general stream information interface
  //   Allows status reports on a synchronous operations,
  //   the distribution of SIs (in a very simple way),
  //   the informs on the widthdrawals of SIs (and other elements)
  //   and the notification of communication errors
  // AUTHOR:
  //   Stephanie Hogg
  // CREATION DATE:
  //  20/08/97
  // UPDATES:
  //

  // #include "StreamCommonTypes.idl"

  interface i_GeneralStreamInfo {
```

```
      // Report unexpected error with binding during normal operations
      oneway void notifyGSInfo(
                in TINASBCommSCommonTypes::t_Notification event);

      // Update on error
      oneway void notifyUpdateGSInfo(
                in TINASBCommSCommonTypes::t_NotifyIdentifier changedEvent,//
identifier
                in TINASBCommSCommonTypes::t_StatusInfo eventChange); // Changed
parameters

      // cancellation of previous error:
      // for those notification for which this is relevent
      oneway void notifyCancelGSInfo(
                in TINASBCommSCommonTypes::t_NotifyIdentifier changedEvent);
};

interface i_PartyGeneralStreamInfo : i_GeneralStreamInfo {

      // Confirms success of an asynchronous operation
      // identified by reqId
      oneway void confirmPartyGSInfo(
                in TINACommonTypes::t_SessionId sessionId,
                in TINAStreamCommonTypes::t_RequestId reqId,
                in TINAStreamCommonTypes::t_RequestType reqType,
                in TINAStreamCommonTypes::t_SBBindState info);

      // Reports failure of an asynchronous operation
      // identified by reqId
      oneway void failurePartyGSInfo(
                in TINACommonTypes::t_SessionId sessionId,
                in TINAStreamCommonTypes::t_RequestId reqId,
                in TINAStreamCommonTypes::t_RequestType reqType,
                in TINAStreamCommonTypes::t_FailureCode  error, // cause of
failure
                in boolean additionalInfo, // Additional info flag
                in TINAStreamCommonTypes::t_ReqProblem  reqProblem);
                                  // Elements causing problem (optional)
                                  // Only valid if additional info flag set

      // Distributes SIs to SB members
      // SIs group SFEPs and identify the associated participant
      // this a useful way of distributing SFEP data
      oneway void SIDistribPartyGSInfo(
                in TINACommonTypes::t_SessionId sessionId,
                in TINAStreamCommonTypes::t_SBId sbId,
                in TINAStreamCommonTypes::t_SIDescList newSIs);
                // This allows participants to know which SIs are available
                // and who they belong to.

      // Distributes SFEPs to SB members
      // Alternative to the above: could be useful if SI is already known
```

```
        oneway void SFEPDistribPartyGSInfo(
                  in TINACommonTypes::t_SessionId sessionId,
                  in TINAStreamCommonTypes::t_SBId sbId,
                  in TINAStreamCommonTypes::t_SFEPServDescList newSFEPs);
                  // This allows participants to know which SFEPS are available

        // Notify withdrawal of elements to an SB
        // Elements could be SFEPs, SIs, SFCs, SB members
        oneway void notifyWithdrawnElementsPartyGSInfo(
                  in TINACommonTypes::t_SessionId sessionId,
                  in TINAStreamCommonTypes::t_SBId sbId,
                  in TINACommonTypes::t_ElementIdList gone);
};

interface i_PartyPaSBInfo : i_PartyGeneralStreamInfo// $$056$$
{
        // No additional operations or attributes
        // Identified (at least not yet...)
};


};

#endif
        // _TINA_PARTY_PASB_FS_IDL_
```

F.7 Module TINAProviderPaSBUsage

```
#ifndef _TINA_PROVIDER_PASB_FS_IDL_
#define _TINA_PROVIDER_PASB_FS_IDL_

// FILE:
//   TINAProviderPaSBUsage.idl
// DESCRIPTION:
//  TINA Participant Oriented Stream Binding Feature Set
//  Provider Module
//
// Interfaces and types needed to support the participant
// oriented stream binding feature set for providers.
// AUTHOR:
//   Stephanie Hogg
// CREATION DATE:
//  20/08/97
// UPDATES:
//

#include "TINAPaSBTypes.idl"

module TINAProviderPaSBUsage {

// #include "PaSBCommonTypes.idl"
// #include "PaSBProviderErrorTypes.idl" $$s001$$
```

```
// DESCRIPTION:
//   Participant oriented stream binding provider error types
//   The error codes and exception types associated with the
//   provider part of the normal particiapant oriented stream binding.
// AUTHOR:
//   Stephanie Hogg
// CREATION DATE:
//  20/08/97
// UPDATES:
//

// #include "StreamCommonTypes.idl"

enum t_PaSBSetupErrors
   {
        PaSBSetup_InvalidSBId,          // Unknown SB
        PaSBSetup_InvalidSBOp,  // Invalid op for this SB
        PaSBSetup_UnknownSBType, // Unknown stream binding type
        PaSBSetup_UnknownMediaType, // Unknown media type
        PaSBSetup_IncompatibleParameters, // Incompatible params:
               // E.g. media type and sb type, media params with media type
        PaSBSetup_InvalidParticipantId,         // Unknown participant
      PaSBSetup_UnknownParticipantType,// Unknown type of participant
      PaSBSetup_SuspendedParticipant, // Suspended participant
        PaSBSetup_UnknownCriteria,// Unknown criterion// $$055$$
        PaSBSetup_InvalidCriteria,       // Not valid for this SB
        PaSBSetup_UnsupportedCriteria, // Not supported by session// $$055$$
        PaSBSetup_CriteriaNotMet,            // Success criteria not met
        PaSBSetup_CommsNotAvailable, // Supporting communications
                                             // not available
        PaSBSetup_InsufficientBandwidth, // Not enough bandwidth // $$055$$
        PaSBSetup_QoSCannotBeMet,               // QoS requirements not met
        PaSBSetup_InsufficientResources, // No resources for connection
        PaSBSetup_NoPathFound, // Could not connect points
        PaSBSetup_UnknownSFEP, // Given SFEP not known
        PaSBSetup_UnknownRFEP // No supporting RFEP
   };

// Exception on sb creation, addion of participants, type modifications
exception e_PaSBSetupError
   {
      t_PaSBSetupErrors errorCode;// Error
      TINACommonTypes::t_ElementId  problemEl;// Element causing problem
               // Valid for appropriate error codes, e.g. Invalid particpant
               // Invalid SB (returns given SBId), invalid participant type
   };


enum t_PaSBOperationErrors
   {
        PaSBOper_InvalidSBId,          // Unknown SB
        PaSBOper_InvalidSBOp,  // Invalid op for this SB
```

```
        PaSBOper_InvalidParticipantId,          // Unknown participant
      PaSBOper_SuspendedParticipant, // Suspended participant
        PaSBOper_CriteriaNotMet,            // Success criteria not met
        PaSBOper_CommsNotAvailable, // Supporting communications
        PaSBOper_InsufficientResources, // No resources for activation
        PaSBOper_CommunicationFailure // Comms unable to meet request
   };


// Exception on state change ops (delete, activate, deactivate)
exception e_PaSBOperationError
   {
      t_PaSBOperationErrors errorCode;// Error
      TINACommonTypes::t_ElementId  problemEl;// Element causing problem
               // Valid for appropriate error codes, e.g. Invalid particpant
               // Invalid SB (returns given SBId)
   };


enum t_PaSBQueryErrors
   {
        PaSBQuery_InvalidSBId,          // Unknown SB
        PaSBQuery_InvalidSBOp,  // Invalid op for this SB
        PaSBQuery_InvalidElementId,    // Unknown element:
                                                 // sfep or pariticpant
        PaSBQuery_UnknownMediaType, // Unknown media type
        PaSBQuery_IncompatibleParameters, // Incompatible params:
      PaSBQuery_SuspendedParticipant // Suspended participant
   };


// Exception on query type operations
exception e_PaSBQueryError
   {
      t_PaSBQueryErrors errorCode;// Error
      TINACommonTypes::t_ElementId  problemEl;// Element causing problem
               // Valid for appropriate error codes, e.g. Invalid particpant
               // Invalid SB (returns given SBId)
   };

// Exception on asynchronous handling of operation
exception e_NoSynchronousReqResp
   {
      TINAStreamCommonTypes::t_RequestId reqId; // Request Identifier for later
response
   };

// #include "i_ProviderPaSBReq.idl" $$s001$$

// DESCRIPTION:
//   i_ProviderPaSBReq interface
//   The participant oriented stream binding request interface
```

```
//   Allows session parties to create a stream binding, modify
//   its configuration, remove the stream binding, and make general
//   queries on stream bindings.
//
//   This interface supports a participant oriented stream binding
//   model.  As a result, operations are generally in terms of:
//   stream binding type
//   media types
//   participants (session members who have a relation with the SB)
//   stream binding identifiers (subsequent operations from creation)
// AUTHOR:
//   Stephanie Hogg
// CREATION DATE:
//  20/08/97
// UPDATES:
//

// #include "PaSBCommonTypes.idl"
// #include "PaSBProviderErrorTypes.idl"

interface i_ProviderPaSBReq
{
      // This interface provides control for all stream bindings
      // to which the requester has access.

      // Establish a stream binding, in terms of type, media, & participants
      void addProviderPaSBReq(
                in TINACommonTypes::t_ParticipantSecretId myId, // Requesters id
                in TINAStreamCommonTypes::t_SBType reqType,// type identifier:
overall type
                in TINASBCommSCommonTypes::t_MediaDescList media,// Media of
assoc flow
                          // additional overall type info,&/or implicit flow
types
                          // Eg. SB type = multimedia, flow types = video, audio
                          // Additional flow type parameters, such as:
                          // QoS: service level qos related to media types
                          // Eg. for audio: CD, FM stereo, AM, phone, mobile
                in TINAPaSBTypes::t_ParticipantDescList reqMembers,
                          // List of participants, each participant described by:
                          // id, type, control SR, and role (source/sink/etc.),
                          // media parameters modifying overal media requirements
                          // success & recovery criteria
                          // initial administrative state state
                in TINAStreamCommonTypes::t_SFEPServDescList requesterSIs,
                          // Optional: if 0 SFEPs, requester is not participating
                in TINAStreamCommonTypes::t_SBSuccessCriteria criteria,
                          // What is necessary for this operation to succeed?
                in TINAStreamCommonTypes::t_SBRecover recActions,
                          // Actions on failure to recover stream binding
                          // Success of recovery criteria
                in boolean wait, // wait for reply flag: true = wait
```

```
                out TINAStreamCommonTypes::t_SBBindState status)// Details of
  stream binding
                raises ( TINAUsageCommonTypes::e_UsageError,
                          e_PaSBSetupError, e_NoSynchronousReqResp );

      // Remove a stream binding
      void deleteProviderPaSBReq(
                in TINACommonTypes::t_ParticipantSecretId myId, // Requesters id
                in TINAStreamCommonTypes::t_SBId sbId, // stream binding id
                in boolean wait, // wait for reply flag: true = wait
                out TINAStreamCommonTypes::t_SBBindState status)// Details of
  stream binding
                raises ( TINAUsageCommonTypes::e_UsageError,
                          e_PaSBOperationError, e_NoSynchronousReqResp );

      // Add participants to a stream binding
      void addParticipantsProviderPaSBReq(
                in TINACommonTypes::t_ParticipantSecretId myId, // Requesters id
                in TINAStreamCommonTypes::t_SBId sbId, // stream binding id
                in TINAPaSBTypes::t_ParticipantDescList reqMembers, // List of
  participants
                in TINAStreamCommonTypes::t_SFEPServDescList requesterSIs, //
  Optional requester SFEPs
                in boolean wait, // wait for reply flag: true = wait
                out TINAStreamCommonTypes::t_SBBindState status) // Details of
  stream binding
                raises ( TINAUsageCommonTypes::e_UsageError,
                          e_PaSBSetupError, e_NoSynchronousReqResp );

      // Remove participants from a stream binding
      void deleteParticipantsProviderPaSBReq(
                in TINACommonTypes::t_ParticipantSecretId myId, // Requesters id
                in TINAStreamCommonTypes::t_SBId sbId, // stream binding id
                in boolean all,// All participants or listed participants?
                in TINAPaSBTypes::t_ParticipantIdList reqMembers,
                          // list of participants to be removed, valid all=false
                in boolean wait, // wait for reply flag: true = wait
                out TINAStreamCommonTypes::t_SBBindState status) // Details of
  stream binding
                raises ( TINAUsageCommonTypes::e_UsageError,
                          e_PaSBOperationError, e_NoSynchronousReqResp );

      // Activate participants in a stream binding/ activate implicit
      // flows specified by type
      void activateParticipantsProviderPaSBReq(
                in TINACommonTypes::t_ParticipantSecretId myId, // Requesters id
                in TINAStreamCommonTypes::t_SBId sbId, // stream binding id
                in boolean all,// All participants or listed participants?
                in TINAPaSBTypes::t_ParticipantIdList reqMembers,
                          // list of members to be activated, valid if all=false
                in boolean allFlows,// All sub types or listed sub types?
                in TINASBCommSCommonTypes::t_MediaDescList reqFlows,
```

```
                    // Media types to be activated, valid allFlows = false
                in boolean wait, // wait for reply flag: true = wait
                out TINAStreamCommonTypes::t_SBBindState status) // Details of
stream binding
                raises ( TINAUsageCommonTypes::e_UsageError,
                          e_PaSBOperationError, e_NoSynchronousReqResp );

      // Deactivate participants in a stream binding/ deactivate implicit
      // flows specified by type
      void deactiviateParticipantsProviderPaSBReq(
                in TINACommonTypes::t_ParticipantSecretId myId, // Requesters id
                in TINAStreamCommonTypes::t_SBId sbId, // stream binding id
                in boolean all, // All participants or listed participants?
                in TINAPaSBTypes::t_ParticipantIdList reqMembers,
                          // list of members to be deactivated, valid all=false
                in boolean allFlows,// All sub types or listed sub types?
                in TINASBCommSCommonTypes::t_MediaDescList reqFlows,
                          // Media types to be deactivated, valid allFlows=false
                in boolean wait, // wait for reply flag: true = wait
                out TINAStreamCommonTypes::t_SBBindState status) // Details of
stream binding
                raises ( TINAUsageCommonTypes::e_UsageError,
                          e_PaSBOperationError, e_NoSynchronousReqResp );

      // Modify the stream binding by adding, modifying, or removing
      // media for the overall stream binding or specified participants
      void modifyParticipantsProviderPaSBReq(
                in TINACommonTypes::t_ParticipantSecretId myId, // Requesters id
                in TINAStreamCommonTypes::t_SBId sbId, // stream binding id
                in boolean all,// All participants or listed participants?
                in TINAPaSBTypes::t_ParticipantIdList reqMembers,//
                          // list of participants modify, valid if all=false
                in TINASBCommSCommonTypes::t_MediaDescList newTypes,// Media
types to add
                in TINASBCommSCommonTypes::t_MediaDescList oldTypes, // Media
types to be removed
                in TINASBCommSCommonTypes::t_MediaChangeDescList modTypes,//
Media types to modify
                in TINAStreamCommonTypes::t_SFEPServDescList requesterSIs, //
Optional requester SFEPs
                in boolean wait, // wait for reply flag: true = wait
                out TINAStreamCommonTypes::t_SBBindState status) // Details of
stream binding
                raises ( TINAUsageCommonTypes::e_UsageError,
                          e_PaSBSetupError, e_NoSynchronousReqResp );

      // Modify the recovery and participation criteria
      void modifyCriteriaProviderPaSBReq( // $$055$$
                in TINACommonTypes::t_ParticipantSecretId myId, // Requesters id
                in TINAStreamCommonTypes::t_SBId sbId, // stream binding id
                in TINAStreamCommonTypes::t_SBSuccessCriteria criteria,// For SB
```

```
                in TINAStreamCommonTypes::t_SBRecover recActions, // Actions
  recover stream binding
                in TINAPaSBTypes::t_PCriteriaList newPCriteria)
                          // List of participants with their new criteria
                raises ( TINAUsageCommonTypes::e_UsageError,
                          e_PaSBSetupError );

      // Notify the stream binding of a single participants relation to
      // the stream binding. (Not sure if needed...)
      void notifyProviderPaSBReq(
                in TINACommonTypes::t_ParticipantSecretId myId, // Requesters id
                in TINAStreamCommonTypes::t_SBId sbId, // stream binding id
                in TINAStreamCommonTypes::t_SFEPServDescList myStatus)
                raises ( TINAUsageCommonTypes::e_UsageError,
                          e_PaSBQueryError );

      // Allows a SB member to withdraw SFEPs (or SIs) from a stream binding
      //   this causes the rerunning of the binding algorithm
      void withdrawSFEPsProviderPaSBReq(
                in TINACommonTypes::t_ParticipantSecretId myId, // Requesters id
                in TINAStreamCommonTypes::t_SBId sbId, // stream binding id
                in TINASBCommSCommonTypes::t_SFEPNameList fepList, // SFEPs to
  withdraw (local only)
                out TINAStreamCommonTypes::t_SBBindState status) // Details of
  stream binding
                raises ( TINAUsageCommonTypes::e_UsageError,
                          e_PaSBSetupError, e_NoSynchronousReqResp);

      // Allows a SB member to register new SFEPs with a stream binding
      //   this causes the rerunning of the binding algorithm
      void registerSFEPsProviderPaSBReq(
                in TINACommonTypes::t_ParticipantSecretId myId, // Requesters id
                in TINAStreamCommonTypes::t_SBId sbId, // stream binding id
                in TINAStreamCommonTypes::t_SFEPServDescList fepList,// SFEPs to
  register with SB
                out TINAStreamCommonTypes::t_SBBindState status) // Details of
  stream binding
                raises ( TINAUsageCommonTypes::e_UsageError,
                          e_PaSBSetupError, e_NoSynchronousReqResp);

      // Rebind: allows a SB to rebind if external session factors
      //   affecting stream binding membership have occurred.
      void rebindProviderPaSBReq(
                in TINACommonTypes::t_ParticipantSecretId myId, // Requesters id
                in TINAStreamCommonTypes::t_SBId sbI, // Stream binding id
                out TINAStreamCommonTypes::t_SBBindState status) // Details of
  stream binding
                raises ( TINAUsageCommonTypes::e_UsageError,
                          e_PaSBSetupError, e_NoSynchronousReqResp);

      // List stream bindings in the service session
      void listProviderPaSBReq(
```

```
                in TINACommonTypes::t_ParticipantSecretId myId, // Requesters id
                in boolean all, // All stream bindings?: set true if all
                in TINACommonTypes::t_ElementIdList particpants,
                                    // List stream bindings with these
participants
                out TINAStreamCommonTypes::t_SBIdList sbList ) // List of SB ids
                raises ( TINAUsageCommonTypes::e_UsageError,
                          e_PaSBQueryError );

      // Get information on a particular stream binding
      void getInfoProviderPaSBReq(
                in TINACommonTypes::t_ParticipantSecretId myId, // Requesters id
                in TINAStreamCommonTypes::t_SBId sbId, // Stream binding id
                out TINAPaSBTypes::t_SBDesc thisSB) // Description of a stream
binding
                raises ( TINAUsageCommonTypes::e_UsageError,
                          e_PaSBQueryError );
};


};

#endif
       // _TINA_PROVIDER_PASB_FS_IDL_
```

F.8 Module TINAPartyPaSBIndUsage

```
#ifndef _TINA_PARTY_PASB_IND_FS_IDL_
#define _TINA_PARTY_PASB_IND_FS_IDL_

// FILE:
//   TINAPartyPaSBIndUsage.idl
// DESCRIPTION:
//  TINA Participant Oriented Stream Binding with Indications
//  Feature Set Party Module
//
// Interfaces and types needed to support the participant
// oriented stream binding extended feature set for users.
// This extended feature set supports indications.
// Note: no extra interfaces need be supported by the provider.
// AUTHOR:
//   Stephanie Hogg
// CREATION DATE:
//  20/08/97
// UPDATES:
//

#include "TINAUsageCommonTypes.idl"
#include "TINAPaSBTypes.idl"

module TINAPartyPaSBIndUsage {

// #include "PaSBCommonTypes.idl"
```

```
// #include "PaSBIndErrorTypes.idl" $$S002$$

// DESCRIPTION:
//   Participant oriented stream binding indication error types
//   The error codes and exception types associated with indications
//   and the extended participant oriented stream binding feature set.
//   Indication operations are usually made on user interfaces.
// AUTHOR:
//   Stephanie Hogg
// CREATION DATE:
//  20/08/97
// UPDATES:
//

// #include "StreamCommonTypes.idl"

enum t_PaSBIndErrors
{
        PaSBInd_InvalidSBId,          // Unknown SB
        PaSBInd_InvalidSBOp,  // Invalid op for this SB
        PaSBUserSetup_UnknownSBType, // Unknown stream binding type
        PaSBInd_UnknownMediaType, // Unknown media type
        PaSBInd_IncompatibleParameters, // Incompatible params
        PaSBInd_InvalidCriteria, // Unknown/invalid criteria
        PaSBInd_UnsupportedCriteria, // Unsupported criteria
        PaSBInd_InvalidElementId,    // Unknown element:
                                                    //  sfep or pariticipant
        PaSBInd_InvalidElementType    // Element type is wrong for op
};

exception e_PaSBIndError
{
      t_PaSBIndErrors errorCode;// Error
      TINACommonTypes::t_ElementId  problemEl;// Element causing problem
               // Valid for appropriate error codes, e.g.
               // Invalid SB - returns given SBId
};

// #include "i_PartyPaSBInd.idl" $$s001$$

// DESCRIPTION:
//   i_PartyPaSBInd interface
//   The participant oriented stream binding indication interface
//   Allows the stream binding to provider to report requested
//   operations to stream members.
//   If using the session graph model, this should be all parties
//   who have an ownership control relatin with the stream binding
// AUTHOR:
//   Stephanie Hogg
// CREATION DATE:
//  20/08/97
// UPDATES:
```

```
//

// #include "PaSBCommonTypes.idl"
// #include "TINAUsageCommonTypes.idl"
// #include "PaSBIndErrorTypes.idl"

interface i_PartyPaSBInd
{
      // Request to establish a stream binding
      void addPartyPaSBInd(
                in TINACommonTypes::t_SessionId sessionId, // Session Id
                in TINAUsageCommonTypes::t_IndId indId,
                          // Indication Id: used in voting to identify
                          // what is being voted on (i.e. this op)
                in TINAStreamCommonTypes::t_RequestId reqId,
                          // Request Id: identifies responses to op
                in TINAStreamCommonTypes::t_SBType reqType,// type identifier:
overall type
                in TINASBCommSCommonTypes::t_MediaDescList media,// Overall
media of assoc flow
                in TINAPaSBTypes::t_ParticipantIdList reqMembers,
                          // List of participants, each participant described
                in TINAStreamCommonTypes::t_SBSuccessCriteria criteria,
                          // What is necessary for this operation to succeed?
                in TINAStreamCommonTypes::t_SBRecover recActions)
                          // Actions on failure to recover stream binding
                          // Success of recovery criteria
                raises ( TINAUsageCommonTypes::e_PartyDomainError,
                           TINAUsageCommonTypes::e_IndError, e_PaSBIndError );

      // Request to remove a stream binding
      void deletePartyPaSBInd(
                in TINACommonTypes::t_SessionId sessionId, // Session Id
                in TINAUsageCommonTypes::t_IndId indId,
                in TINAStreamCommonTypes::t_RequestId reqId,
                in TINAStreamCommonTypes::t_SBId sbId) // stream binding id
                raises ( TINAUsageCommonTypes::e_PartyDomainError,
                           TINAUsageCommonTypes::e_IndError, e_PaSBIndError );

      // Request to add participants to a stream binding
      void addParticipantsPartyPaSBInd(
                in TINACommonTypes::t_SessionId sessionId, // Session Id
                in TINAUsageCommonTypes::t_IndId indId,
                in TINAStreamCommonTypes::t_RequestId reqId,
                in TINAStreamCommonTypes::t_SBId sbId, // stream binding id
                in TINAPaSBTypes::t_ParticipantIdList reqMembers) // alt: id list
                          // List of participants, each participant described
                raises ( TINAUsageCommonTypes::e_PartyDomainError,
                           TINAUsageCommonTypes::e_IndError, e_PaSBIndError );

      // Request to remove participants from a stream binding
      void deleteParticipantsPartyPaSBInd(
```

```
                in TINACommonTypes::t_SessionId sessionId, // Session Id
                in TINAUsageCommonTypes::t_IndId indId,
                in TINAStreamCommonTypes::t_RequestId reqId,
                in TINAStreamCommonTypes::t_SBId sbId, // stream binding id
                in boolean all,// All participants or listed participants?
                in TINAPaSBTypes::t_ParticipantIdList reqMembers)
                raises ( TINAUsageCommonTypes::e_PartyDomainError,
                          TINAUsageCommonTypes::e_IndError, e_PaSBIndError );

        // Request to activate by participants (and opt. implicit flows)
        void activateParticipantsPartyPaSBInd(
                in TINACommonTypes::t_SessionId sessionId, // Session Id
                in TINAUsageCommonTypes::t_IndId indId,
                in TINAStreamCommonTypes::t_RequestId reqId,
                in TINAStreamCommonTypes::t_SBId sbId, // stream binding id
                in boolean all,// All participants or listed participants?
                in TINAPaSBTypes::t_ParticipantIdList reqMembers,
                          // list of members to activate, valid if all=false
                in boolean allFlows,// All sub types or listed sub types?
                in TINASBCommSCommonTypes::t_MediaDescList reqFlows)
                          // Media types to activate, valid if allFlows=false
                raises ( TINAUsageCommonTypes::e_PartyDomainError,
                          TINAUsageCommonTypes::e_IndError, e_PaSBIndError );

        // Request to deactivate by participants (and opt. implicit flows)
        void deactivateParticipantsPartyPaSBInd( // $$055%%
                in TINACommonTypes::t_SessionId sessionId, // Session Id
                in TINAUsageCommonTypes::t_IndId indId,
                in TINAStreamCommonTypes::t_RequestId reqId,
                in TINAStreamCommonTypes::t_SBId sbId, // stream binding id
                in boolean all,// All participants or listed participants?
                in TINAPaSBTypes::t_ParticipantIdList reqMembers,
                          // list of members to deactivate, valid if all=false
                in boolean allFlows,// All sub types or listed sub types?
                in TINASBCommSCommonTypes::t_MediaDescList reqFlows)
                          // Media types to deactivate, valid if allFlows=false
                raises ( TINAUsageCommonTypes::e_PartyDomainError,
                          TINAUsageCommonTypes::e_IndError, e_PaSBIndError );

        // Request to modify participation
        void modifyParticipantsPartyPaSBInd(
                in TINACommonTypes::t_SessionId sessionId, // Session Id
                in TINAUsageCommonTypes::t_IndId indId,
                in TINAStreamCommonTypes::t_RequestId reqId,
                in TINAStreamCommonTypes::t_SBId sbId, // stream binding id
                in boolean all,// All participants or listed participants?
                in TINAPaSBTypes::t_ParticipantIdList reqMembers,//
                          // list of participants to modify, valid if all=false
                in TINASBCommSCommonTypes::t_MediaDescList newTypes,// Flow
  types to add or modify
                in TINASBCommSCommonTypes::t_MediaDescList oldTypes, // Media
  types to be removed
```

```
                in TINASBCommSCommonTypes::t_MediaChangeDescList modTypes)//
  Media types to modify
                raises ( TINAUsageCommonTypes::e_PartyDomainError,
                          TINAUsageCommonTypes::e_IndError, e_PaSBIndError );

      // Modify the recovery and participation criteria
      void modifyCriteriaPartyPaSBInd( // $$055$$
                in TINACommonTypes::t_SessionId sessionId, // Session Id
                in TINAUsageCommonTypes::t_IndId indId,
                in TINAStreamCommonTypes::t_RequestId reqId,
                in TINAStreamCommonTypes::t_SBId sbId, // stream binding id
                in TINAStreamCommonTypes::t_SBSuccessCriteria criteria,// For SB
                in TINAStreamCommonTypes::t_SBRecover recActions, // Actions
  recover stream binding
                in TINAPaSBTypes::t_PCriteriaList newPCriteria)
                         // List of participants with their new criteria
                raises ( TINAUsageCommonTypes::e_PartyDomainError,
                          TINAUsageCommonTypes::e_IndError, e_PaSBIndError );

      // Request from a participant to withdraw SFEPs from SB
      void withdrawSFEPsPartyPaSBInd(
                in TINACommonTypes::t_SessionId sessionId, // Session Id
                in TINAUsageCommonTypes::t_IndId indId,
                in TINAStreamCommonTypes::t_SBId sbId, // stream binding id
                in TINASBCommSCommonTypes::t_SFEPNameList fepList) // SFEPs to
  withdraw (local only)
                raises ( TINAUsageCommonTypes::e_PartyDomainError,
                          TINAUsageCommonTypes::e_IndError, e_PaSBIndError );

      void registerSFEPsPartyPaSBInd( // Allows a particpant register SFEPs
                in TINACommonTypes::t_SessionId sessionId, // Session Id
                in TINAUsageCommonTypes::t_IndId indId,
                in TINAStreamCommonTypes::t_SBId sbId, // stream binding id
                in TINAStreamCommonTypes::t_SFEPServDescList fepList)// SFEPs to
  register with SB
                raises ( TINAUsageCommonTypes::e_PartyDomainError,
                          TINAUsageCommonTypes::e_IndError, e_PaSBIndError );

      void rebindPaSBFSInd(
                in TINACommonTypes::t_SessionId sessionId, // Session Id
                in TINAUsageCommonTypes::t_IndId indId,
                in TINAStreamCommonTypes::t_RequestId reqId,
                in TINAStreamCommonTypes::t_SBId sbI) // Stream binding id
                raises ( TINAUsageCommonTypes::e_PartyDomainError,
                          TINAUsageCommonTypes::e_IndError, e_PaSBIndError );
  };

  };

  #endif
        // _TINA_PARTY_PASB_IND_FS_IDL_
```

8.2 Module TINAPartyCommSUsage

```
#ifndef _TINAPartyCommSUsage_MODULE_
#define _TINAPartyCommSUsage_MODULE_

// FILE:
//   TINAPartyCommSUsage.idl// $$054$$
// DESCRIPTION:
//   The communication session interface module for the Party side
//   of the Ret reference
//   This i_TerminalFlowControl interface and communication session
//   exceptions.
//   Jarno Rajahalme
// AUTHOR:
//   Stephanie Hogg (from the i_TerminalFlowControl.idl file of Jarno R.)
// CREATION DATA:
//   01/10/1997
// UPDATES:
//


#include "TINASBCommSCommonTypes.idl"
#include "TINAConSCommSCommonTypes.idl"
#include "TINACommSCommonTypes.idl"


// Notes:
//
// Correlation Id:
// This is a terminal unique identifier that identifies a TFC branch
// joining a SFEP and a NFEP.  It is devised by the terminal and may include
// the terminal domain name.  The correlation Id is issued when a TFC branch
// is initiated.
//
// TFC configurations:
// The following TFC configurations are allowed:
// Point to point: single SFEP to a single NFEP (simplest case)
// Point (SFEP) to Multipoint (SFEP/NFEP): supports one SFC over multiple
//             NFCs (useful if party has multiplexing resources)
// Point (NFEP) to Multipoint (SFEP): supports multiple SFCs over the
//        one NFC, in particular supports bidirectional NFCs

module TINAPartyCommSUsage {  // $$s008$$

typedef TINASBCommSCommonTypes::t_AdministrativeState t_AdministrativeState;
typedef TINASBCommSCommonTypes::t_TinaName t_TinaName;
typedef TINASBCommSCommonTypes::t_ObjectIdentifier t_ObjectIdentifier;
typedef TINASBCommSCommonTypes::t_AttribList t_AttribList;
typedef TINASBCommSCommonTypes::t_SFEPName t_SFEPName;
typedef TINASBCommSCommonTypes::t_SFEPNameList t_SFEPNameList;
typedef TINACommSCommonTypes::t_CapabilityList t_CapabilityList;
typedef TINACommSCommonTypes::t_CapabilitySet t_CapabilitySet;
typedef TINACommSCommonTypes::t_TFCName t_TFCName;
```

```
typedef TINACommSCommonTypes::t_CorrelationId t_CorrelationId;
typedef TINACommSCommonTypes::t_CorrelationIdList t_CorrelationIdList;
typedef TINACommSCommonTypes::t_SFEPCorrelationList t_SFEPCorrelationList;
typedef TINACommSCommonTypes::t_NFCCorrelationList t_NFCCorrelationList;
typedef TINACommSCommonTypes::t_SFEPSelect t_SFEPSelect;
typedef TINACommSCommonTypes::t_SFEPSelectList t_SFEPSelectList;
typedef TINAConSCommSCommonTypes::t_NFCName t_NFCName;
typedef TINAConSCommSCommonTypes::t_NFCNameList t_NFCNameList;
typedef TINAConSCommSCommonTypes::t_ANfepList t_ANfepList;



#ifndef _EXCEPTIONS_IDL_
#define _EXCEPTIONS_IDL_


// Errors related to capability selection
enum t_CapabilityErrorCode
{
      CapabilityError_CapabilityNotKnown,
      CapabilityError_CapabilityNotAvailable,
      CapabilityError_CapabilityNotSupported,
      CapabilityError_CapabilitiesIncompatible,
      CapabilityError_SFEPIncompatible
};
exception e_CapabilityError
{
      t_CapabilityErrorCode errorCode;
      t_ObjectIdentifier capability;// Capability id
      t_SFEPName element;// Associated SFEP
      string addInfo;//
};

// Errors related to TFC initation, TFC update, or new TFC branches
enum t_TFCErrorCode
{
      TFCError_TFCUnknown,// Udpates and additions only
      TFCError_CorrelationIdUnknown,// Updates only
      TFCError_InsufficientResources,// No resources currently
      TFCError_NoTerminalSupport, // No terminal resources (ever)
      TFCError_InvalidResourcesForEntity,// Not valid with SFEP/TFC
      TFCError_QoSNotMet,// Can't meet QoS requirements
      TFCError_MediaNotSupported, // Problem with SFEP type
      TFCError_SFEPNotSetup, // SFEP has not been setup yet
      TFCError_SFEPIncorrectlySetup, // Error setting up SFEP
      TFCError_NoNetworkSupport // No network support available for TFC
};
exception e_TFCError
{
      t_TFCErrorCode errorCode;
      t_TinaName element;// TFC or Correlation identifier
      string addInfo;
```

```
};

// Exceptions to support simple queries
// and or simple operations like delete, activate, deactivated
enum t_CSQueryErrorCode
{
      CSQueryError_SFEPUnknown,
      CSQueryError_TFCUnknown,
      CSQueryError_CorrelationIdUnknown,
      CSQueryError_OperationNotValidOnElement
};

exception e_CSQueryError
{
      t_CSQueryErrorCode errorCode;
      t_TinaName elementId;
      string   addInfo;
};


// Associate / dissociate errors
enum t_CSNFEPErrorCode
{
      CSNFEPError_NFEPUnknown,
      CSNFEPError_CorrelationIdUnknown,
      CSNFEPError_InsufficientResources,
      CSNFEPError_QoSNotMet,
      CSNFEPError_SFEPNotSetup
};

exception e_CSNFEPError
{
      t_CSNFEPErrorCode errorCode;
      t_TinaName elementId;
      string   addInfo;
};

enum t_PartyDomainErrorCode {
        PD_UnknownError,
        PD_UnAuthorisedAccess,
        PD_InvalidSessionId,
        PD_OpNotSupported
};

exception e_PartyDomainError {
        t_PartyDomainErrorCode errorCode;
};

#endif

// Adapted from: the TCSM component defintions
// Created by Jarno Rajahalme
```

```
// Updates:
// 97/08/06 by Frank Steegmans
//    Changes to get the file compiled
// 97/08/08 by Frank Steegmans
//    Complete part with respect to NFEPs
// 97/09/29 by Stephanie Hogg
//    Major changes to:
//    a) Replace module names
//    b) allow terminals to set TFCs
//    c) allow multiple branch TFCs
//    d) rationalise operations
// 97/10/01 by Stephanie Hogg
//    Split SelectCapabilities into two operations
//    Add combined select and initiate operations
//
//
// REMARK:
// All these functions are Ret-RP functions.
//


#ifndef _I_TFlow_Control_IDL_
#define _I_TFlow_Control_IDL_

// Interface: i_BasicTerminalFlowControl
// This interface supports basic operations required to
// configure SFEPs and initiate and control TFCs and TFC branches.
//

  interface i_BasicTerminalFlowControl {

    //
    // CSM uses queryCapabilities to get the terminal
    // capabilities available for given set of SFEPs. CSM will
    // query the capabilities of each terminal involved in the
    // SFC, and will determine a common mode of operation.
    //
    // A single capability set will be returned containing all the
    // terminal capability information relating to the SFEPs in
    // question.
    //

    void queryCapabilities (
               in  t_SFEPNameList sfeps,
               out t_CapabilitySet capabilities)
      raises (e_PartyDomainError, e_CSQueryError);


    // Select Capability type functions:
    //
    // CSM uses these functions to select the set of
    // capabilities (codecs, session protocols, transport
```

```
// protocols) for an SFEP (for one at a time).
//
// The selected capabilities must form a contiquous chain as
// dictated by the dependencies of each capability. The
// capability chain is given in depth-first order (which is
// significant only if a capability in the chain requires (is
// dependent on) more than one other capability).
//
// Typically the chain will contain a codec capability, a
// session protocol capability and a transport protocol
// capability (in this order).
//
// Generally, capabilities are chosen for the SFC source followed by
// the sinks. If capability selection fails for the source SFEP,
// the SFC can not be set up. If this operation fails for a sink SFEP,
// failure of the SFC is dependent on the policy of the SFC setup (all
// or nothing, etc.)

// Data Types:
// List of attribute lists corresponding to capabilityChain
// 97/08/06 by fste:
// typedef taken out of procedure declaration to comply with new
// idl grammer specifications.
typedef sequence<t_AttribList> t_AttribListChain;
typedef sequence<t_AttribListChain> t_SinkAttributes;

// Operation: selectSFCCapabilities
// CSM calls this operation first for the source SFEP of the SFC, passing
// the attributes of each sink for each capability present in
// the chosen capability chain. This assumes the attributes are not
// interpreted by the CSM, so the CSM delegates the matching of
// the attributes to the source SFEP. The localCaps argument
// contains arguments selected from the Source's capability set
// obtained during a previous QueryCapabilities call.  These capabilities
// are reserved for the local SFEPs use. The TCSM is
// responsible for considering the attributes of the sinks, and
// then returning a common set of attributes in commonCaps.
// It may return its transport requirements in the transportReqs parameter.

void selectSFCCapabilities (
           in t_SFEPName      sfep,
           in t_CapabilityListlocalCaps,
           in t_SinkAttributessfcCaps,
           out t_CapabilityListcommonCaps,
           out t_CapabilityListtransportReqs)
  raises (e_PartyDomainError, e_CSQueryError, e_CapabilityError);

// Operation: selectSFEPCapabilities
// Then, for sink SFEPs, CSM will pass the desired capabilities
// (chosen by the source in the SelectSFCCapabilities op) using the
// SelectSFEPCapabilities operation. These capabilities are reserved
// for the SFEPs use. The capabilityList returned by
```

```
      // a sink SFEP contains the transport requirements for the sink.

      void selectSFEPCapabilities (
                in t_SFEPName      sfep,
                in t_CapabilityListlocalCaps,
                out t_CapabilityListtransportReqs) // $$057$$
        raises (e_PartyDomainError, e_CSQueryError, e_CapabilityError);

      // The transport protocol capability attributes returned by the
      // operations can also be passed to CC by CSM. Typically these
      // attributes will contain the QoS parameters meaningful for
      // the transport protocol(s) in question.
      // Comment: Actually - Protocol and QoS requirements will also be
      // part of the ANfep returned when the IniateTFC operation is called.


      //
      // Terminal Flow Connection management
      // (Basic functionality)
      //

      // Operation: initiateTFC
      // Initiate a TFC to associate one or more SFEPs with a NFC
      // This operation is given a list of the TFC's SFEPS. It returns
      // a set of anfeps which can be used with the TFC and the correlation
      // Ids associated with each branch of the TFC.
      // The CSM will need to decide which one(s) is passed over ConS.
      // This one anfep can still be a pool from which a certain nfep and
      // eventually a tp has to be selected.
      // An exception will be thrown if the codecs and session protocols
     // have not been selected yet or if one of the sfeps is already exclusively
  // associated with a particular nfep.
      void initiateTFC(
                in t_SFEPNameList  sfeps,
                in t_NFCName         connection,
                in t_AdministrativeState  state,
                out t_TFCName        newTFC,
                out t_SFEPCorrelationList correlation,
                out t_ANfepList      requiredNfeps)
        raises (e_PartyDomainError, e_TFCError);

      // Operation: initiateMultiNFEPTFC
      // Initiate a TFC to associate one SFEP with one or more NFCs
      // This operation is given the root SFEP and a list of the
      // TFC's associated NFCs. It returns list of correlation ids associated
      // with each branch of the TFC and a set of anfeps for each branch.
      // The CSM will need to decide which one(s) is passed over ConS for
      // each branch. These anfeps can still be a pool from which a certain
      // nfep and eventually a tp has to be selected.
      // An exception will be thrown if the codecs and session protocols
      // have not been selected yet or if the sfep is already excusively
      // associated with a particular TFC.
```

```
void initiateMultiNFEPTFC(
           in t_SFEPName       sfep,
           in t_NFCNameList  connections,
           in t_AdministrativeState state,
           out t_TFCName       newTFC,
           out t_NFCCorrelationList correlation)
  raises (e_PartyDomainError, e_TFCError);


// Operation: addTFCBranches
// Add branches to an existing TFC by associating more
// SFEPs with it.  The TFC already has a root Anfep or NFEP.
// The branch correlation identifiers are returned.
void addTFCBranches (
           in t_TFCName         aTFC,
           in t_SFEPNameList   sfeps,
           in t_AdministrativeState  state,
           out t_SFEPCorrelationList correlation)
  raises (e_PartyDomainError, e_TFCError);

// Operation: addMultiNFEPTFCBranches
// Add branches to an existing TFC by associating more
// NFCs with it.  The TFC already has a root SFEP.
// The branch correlation identifiers and their anfep list
// is returned.  The anfep list is resolved by the CSM to the
// anfep passed over the cons reference point.
void addMultiNFEPTFCBranches (
           in t_TFCName aTFC,
           in t_NFCNameList connections,// $$057$$
           in t_AdministrativeState state,
           out t_NFCCorrelationList correlation)
  raises (e_PartyDomainError, e_TFCError);

// Operation: deleteTFCBranches
// Delete the entire TFC or the indicated branches.
// The correlation identifiers identify the branches.
// The correlation ids are invalid once the branch has been deleted.
// The TFC name is invalid once the entire TFC is deleted.
// Calling this operation frees all resources used by the related
// SFEPs and NFEPs.
//
void deleteTFCBranches (
           in t_TFCNameaTFC,
           in booleanall,
           in t_CorrelationIdList branches)
  raises(e_PartyDomainError, e_CSQueryError);

// Operation: activateTFCBranches
// Activate the entire TFC or selected branches.
// Set administrative state of the TFC branches and related SFEP and
// NFEP to 'unlocked' state as appropriate
//
```

```
void activateTFCBranches (
           in t_TFCNameaTFC,
           in booleanall,
           in t_CorrelationIdList branches)
  raises(e_PartyDomainError, e_CSQueryError);

// Operation: deactivateTFCBranches
// Deactivate the entire TFC or selected branches.
// Set administrative state of the TFC branch and related SFEPs and
// NFEPs to a 'locked' state as appropriate.
//
void deactivateTFCBranches (
           in t_TFCNameaTFC,
           in booleanall,
           in t_CorrelationIdList branches)
  raises(e_PartyDomainError, e_CSQueryError);

// Operation: updateTFCBranches
// Requests the update of an entire TFC or certain of its branches
// following the modification of SFEP capabilities.
// The update may require no change the NFEP, a modification of
// an existing NFEP (hence NFC modification) or the use of a new NFEP
// These choises are set by the enum t_NFEPUpdate
// If a NFEP modification is required, the mods parameter describes the
// modifications.
// If a new NFEP type is required, the mods parameter describes possible
// new anfeps to replace the existing one.
void updateTFCBranches (
           in t_TFCName aTFC,
           in boolean all,
           in t_CorrelationIdList branches,
           in boolean noDisruption,
           out TINACommSCommonTypes::t_NFEPUpdate reqChange,
           out t_ANfepList mods)
  raises(e_PartyDomainError, e_TFCError);

// Operation: updateMultiNFEPTFCBranches
// Requests the update of an entire TFC or certain of its branches
// following the modification of SFEP capabilities.
// As the TFC has multiple associated NFEPs, updates for each
// NFEP may be required. Required changes are given by the
// mods parameter (a list of branches, update requirements
// and anfep descriptions)
void updateMultiNFEPTFCBranches (
           in t_TFCName aTFC,
           in boolean all,
           in t_CorrelationIdList branches,
           in boolean noDisruption,
           out TINACommSCommonTypes::t_BranchUpdateList mods)
  raises(e_PartyDomainError, e_TFCError);
```

```
      // Following operations are used to manage the NFEPs within the
      // TFC.
      //
      // With ConS-RP compliant providers, the local TLA will be
      // invoking these operations, while non ConS-RP compliant
      // operator will most probably call these from the Retailer
      // domain.
      //

      // Operation: associateNFEP
      // Resolve the NFEP for one or more TFC branches. This will make TFC
      // operational. It assumes the initial state set by the initiate
      // TFC operation.  If the branches are inactive, an activateTFC()
      // operation is required to activate the branches.
      //
      void associateNFEP (
                 in t_CorrelationIdList branches, // $$057$$
                 in t_TinaName aNFEP)
        raises(e_PartyDomainError, e_CSNFEPError);

      // Operation: removeNFEP
      // Remove NFEP from the TFC branches. This will make TFC branches
      // unoperable, and will implicitly deactivate the TFC branches.
      //
      //
      void removeNFEP (
                 in t_TinaName aNFEP,
                 in boolean all,             // $$057$$
                 in t_CorrelationIdList branches) // $$057$$
        raises(e_PartyDomainError, e_CSNFEPError);

    }; // interface i_BasicTerminalFlowControl



  // Interface: i_TerminalFlowControl
  // This interface extends the basic terminal flow control
  // functionality to include combined operations that let the
  // CSM select SFEP capabilities and initiate TFCs (or add branches)
  // in a single step.
  //

    interface i_TerminalFlowControl : i_BasicTerminalFlowControl {

      // Combined operations:
      //
      // The following combination operations are provided for efficency
      // in setting up TFCs.
      //

      // Operation: for a multiple SFEP to NFC combination where
      // the SFEP capabilities have been resolved.
```

```
      void resolveSFEPforTFC (
                in t_SFEPSelectList  sfeps,
                in t_NFCName         connection,
                in t_AdministrativeState  state,
                out t_TFCName       newTFC,
                out t_SFEPCorrelationList correlation,
                out t_ANfepList      requiredNfeps)
        raises (
                e_PartyDomainError,
                e_TFCError,        // $$055$$ included from document
                e_CapabilityError
        );

      // Operation: for a SFEP to multiple NFC combination where
      // the SFEP capabilities have not been resolved.
      void resolveSFEPforMultiNFEPTFC (
                in t_SFEPSelect     sfep,
                in t_NFCNameList connections,// $$057$$
                in t_AdministrativeState state,// $$057$$
                in t_SinkAttributes sfcCaps,// $$057$$
                out t_TFCName       newTFC,
                out t_CapabilityList     commonCaps,
                out t_NFCCorrelationList correlation)
        raises (e_PartyDomainError, e_TFCError, e_CapabilityError);

  // $$057$$ tags  (3 lines above) replaced the following 4 lines
  //             in t_CapabilityList localCaps,
  //             in t_AdministrativeState state,
  //             in t_NFCNameList connections,
  //             in t_SinkAttributes  sfcCaps,


      // Operation:  resolveSFEPforBiTFC
      // For a SFEP for bidirectional NFC support where
      // the SFEP capabilities have not been resolved.
      // First one end of the birdectional connection is setup
      // using this command.  The other end can be set using
      // the resolveSFEPforTFC command. (This ignores the normal
      // convention of selecting capability for root SFEP first!)
      void resolveSFEPforBiTFC (
                in t_SFEPSelectList sfeps, // Only two SFEPs
                in t_NFCName connection,// $$057$$
                in t_AdministrativeState state,// $$057$$
                in t_SinkAttributes  sfcInCaps,
                in t_SinkAttributes  sfcOutCaps,
                out t_TFCName       newTFC,
                out t_CapabilityList     commonInCaps,
                out t_CapabilityList     commonOutCaps,
                out t_SFEPCorrelationList correlation,
                out t_ANfepList      requiredNfeps)
        raises (e_PartyDomainError, e_TFCError, e_CapabilityError);
```

```
// $$057$$ tags  (2 lines above) replaced the following 2 lines
//             in t_AdministrativeState state,
//             in t_NFCNameList connections,


    // Operation: resolveSFEPforTFCBranches
    // Resolve SFEP capabilities and initiate new TFC branches.
    // This operation assumes the SFEP capabilities have already
    // been determined.
    void resolveSFEPforTFCBranches (
                in t_TFCName         aTFC,
                in t_SFEPSelectList  sfeps,
                in t_AdministrativeState  state,
                out t_SFEPCorrelationList correlation)
      raises (e_PartyDomainError, e_TFCError);

  }; // end of i_TerminalFlowControl

#endif
       // _I_TFlow_Control_IDL_

}; // module TINAPartyCommSUsage

#endif
       // _TINAPartyCommSUsage_MODULE_
```