6. Planning Subsystem (PLN) CSCIs

6.1 Overview

The Planning subsystem is comprised of the PLANG CSCI which includes several COTS plus custom software. This CSCI consists of a number of utility programs and server applications. The distinction between these two classes of programs is useful when describing the software design:

- Utilities are programs that perform well defined tasks and are invoked at the operator's requests. A utility is usually exited when the task is complete; the system does not depend on the utility being active at any given time. As an example, a mail editor is a common utility program, invoked whenever a user wishes to generate a mail message.
- Servers are programs that perform ongoing tasks and have to be active at all times for the subsystem to carry out its allocated functions. An example here is the mail daemon; the mail won't be delivered unless the daemon is running.

The allocation of the Planning subsystem capabilities to distinct applications accounts for the need to provide distinct interfaces for distinct activities, and to restrict access to planning functions to classes of users. The applications define an integrated set of tools with well-defined functions to supplement the COTS components. The allocation of capabilities to applications within the Planning CSCI also accounts for the need to permit independent development of the components as the capabilities of the ECS evolve. The purpose of each application is briefly outlined below.

PDPS Database

At the heart of the Planning CSCI is the Planning and Data Processing System (PDPS) Database (and the implied database server). This provides the persistent storage for data and facilitates the sharing of this data between the applications. The database also provides security, fault tolerance, and marshals requests for concurrent access to data. The objects which are built from the persistent data within the PDPS database are indicated within the object design. The schema for the database has been driven by the object model and is described in Section 9 of this specification.

6.1.1 Production Request Editor

This application allows the user to submit production requests that describe the data products to be produced. The application uses the PGE descriptions (profiles) entered during AI&T in order to work out the tasks - Data Processing Requests - that in sum meet the request. The application provides the capabilities to add, modify, and delete Production Requests, as well as review the resulting Data Processing Requests. The production request editor is identified as a distinct application and separate from the workbench in anticipation that defining production requests will be a discrete activity, unrelated to the “planning” of these events.

6.1.2 Production Planning Workbench

The application is used to prepare a schedule for the production at a site, and forecast the start and completion times of the activities within the schedule. These functions provided by the workbench include the following high-level activities:

- Candidate Plan Creation -- from the production requests prepared by the Production request Editor;
- Plan Activation -- activating a candidate plan;
- Updating the Active Plan -- feedback from the processing into the active plan; and
- Canceling/Modifying the Active Plan.

As described previously, activating a plan entails rolling a portion of a selected plan into the AutoSys COTS. This “schedule” is then managed within the Data Processing subsystem. The forecast times generated within the planner are used to set up operator alerts that would make the operator aware of gross departures from the predicted schedule. The production planning workbench can periodically update its predictions using feedback from the AutoSys.

6.1.3 Planning Subscription Editor

This application provides the capabilities required to submit subscriptions to the IMF for the insertion of data. Registration of a subscription is required for the Planning CSCI to receive notification when data arrive within the ECS. The submission of subscriptions is managed as an operator initiated activity.

6.1.4 Subscription Manager

The Subscription Manager is used to manage the receipt of a subscription notification. Subscription notification is used to notify Planning upon the arrival of input data required by a given PGE. The Subscription Notification contains Universal References (URs) which are pointers to the data objects stored in the IMF. The Subscription Manager updates the PDPS database to indicate when data become available. When all input data for a Data Processing Request is available, the job defined for that Data Processing Request is released within the Data Processing subsystem.

6.2 Subsystem Context

Figure 6.2-1 illustrates the subsystem interfaces within the Planning Subsystem.

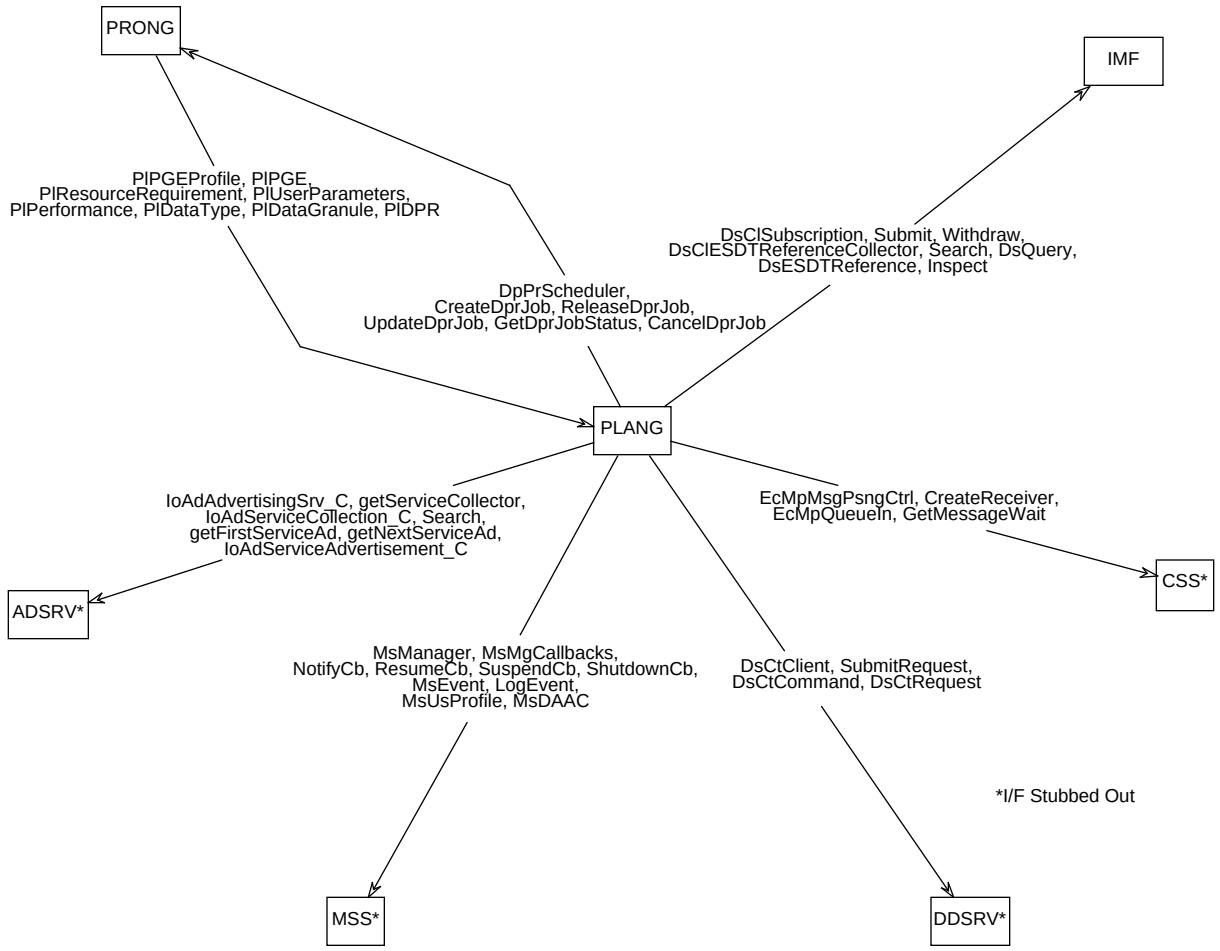


Figure 6.2-1. CSCI Event Flow Context Diagram

6.3 Subsystem Object Model

The CSCI object model is shown in a number of diagrams. Each diagram captures a different portion of the CSCI capabilities.

This page intentionally left blank.

6.3.1 Planning_Core_Library Class Category

6.3.1.1 Overview

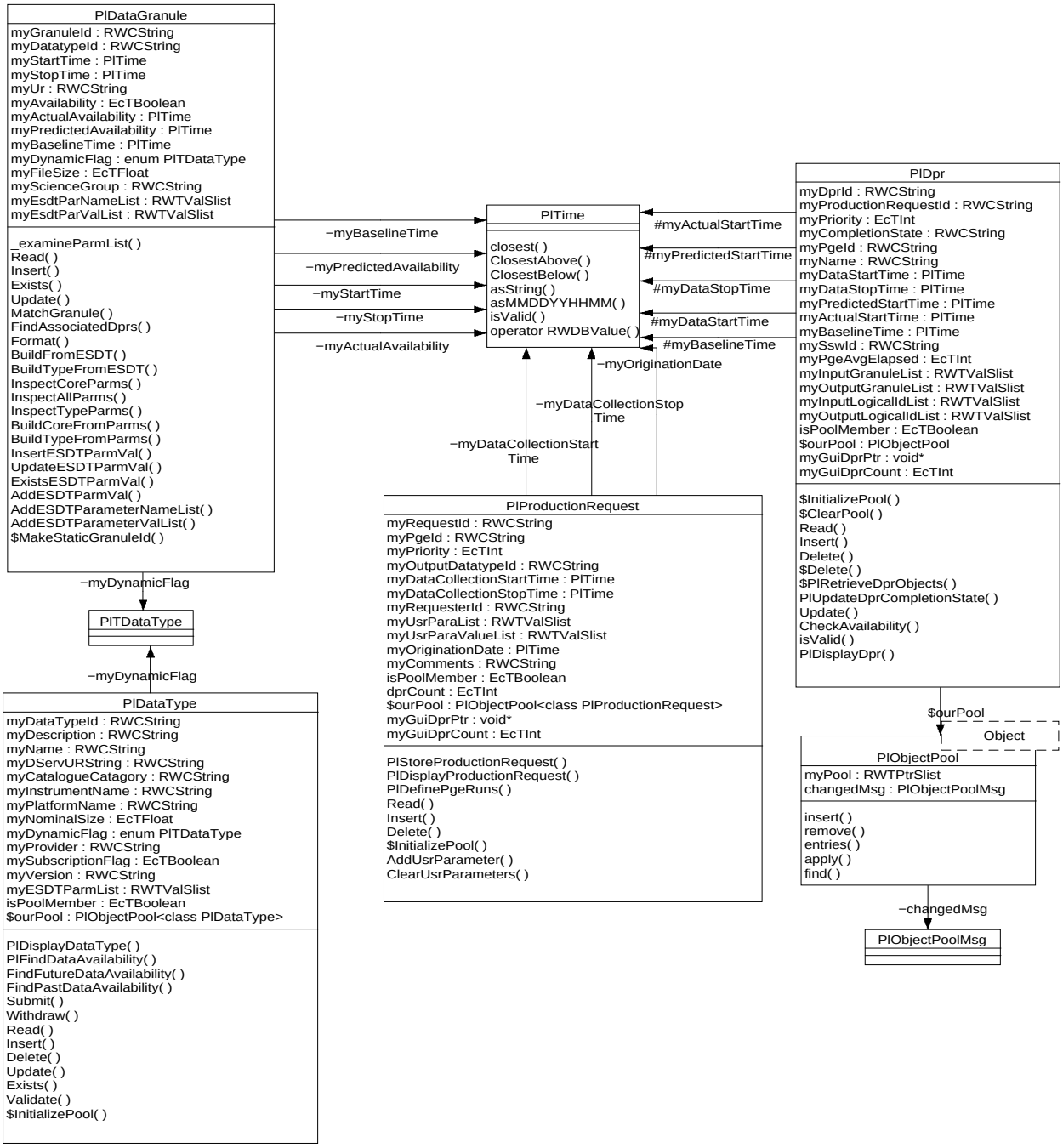


Figure 6.3.1.1-1 Core_Library_Part_1

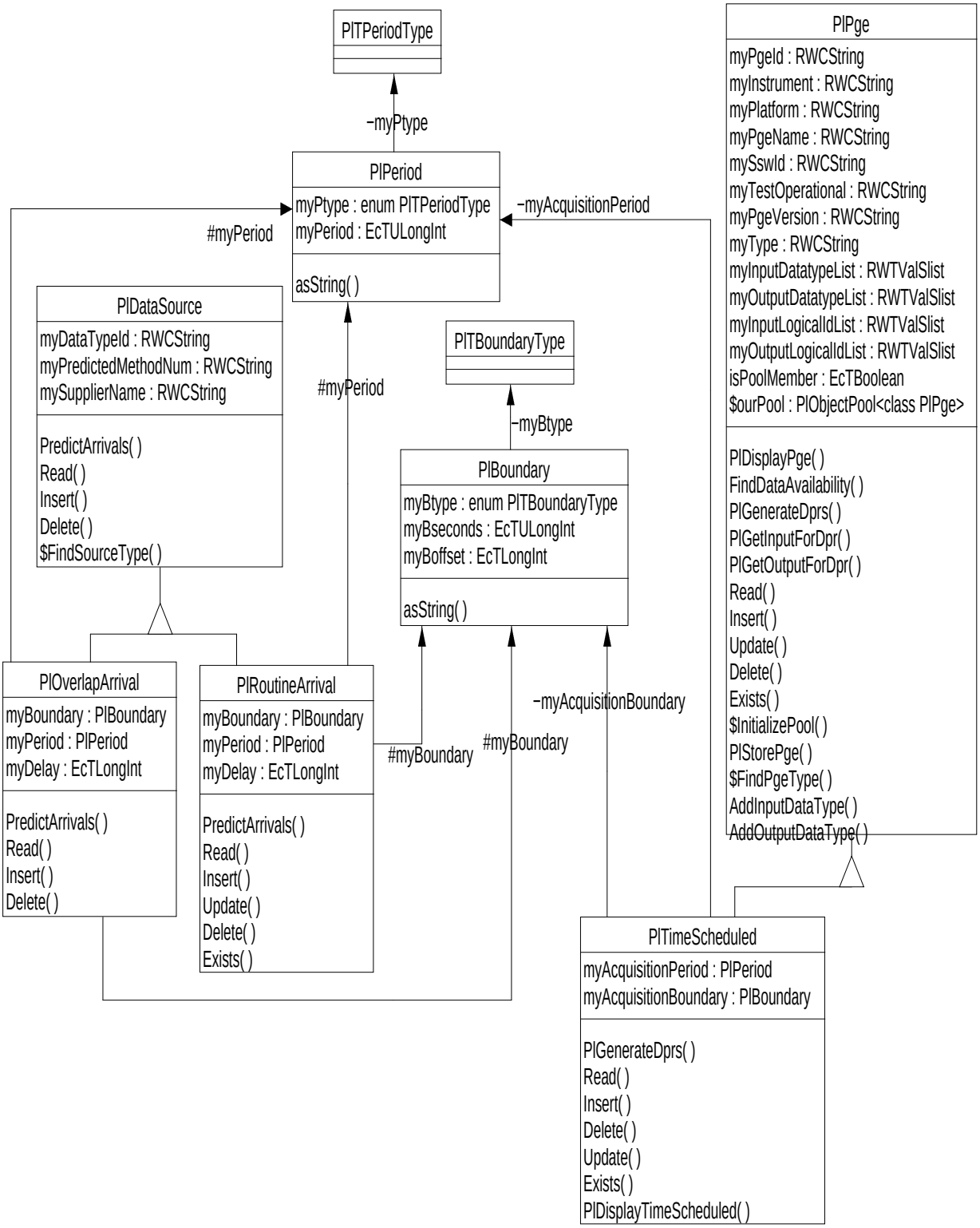


Figure 6.3.1.1-2 Core_Library_Part_2

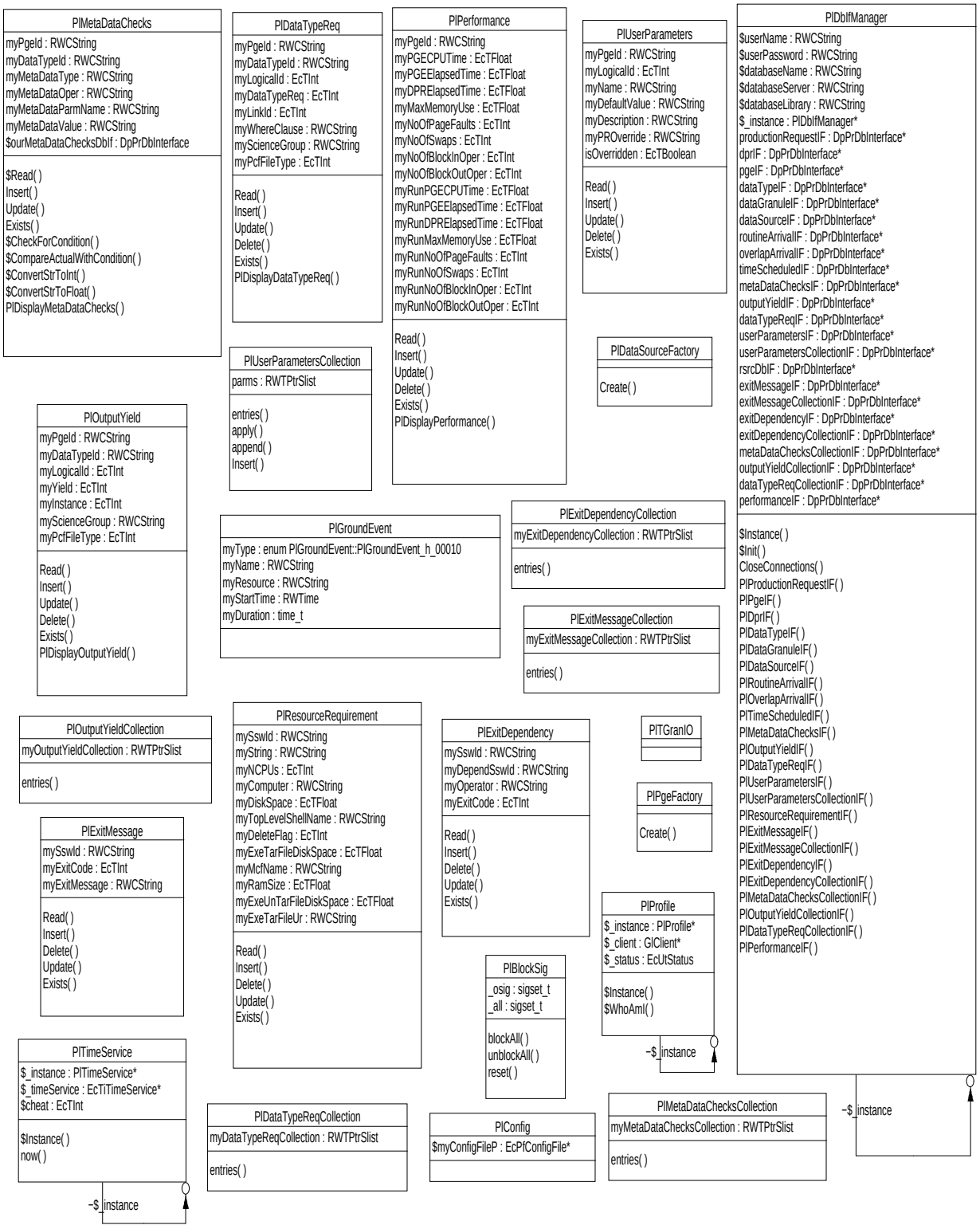


Figure 6.3.1.1-3 Core_Library_Part_3

6.3.1.2 Planning_Core_Library Classes

6.3.1.3 PlBlockSig Class

Overview:

POSIX signal blocking functions

Export Control: Public

Inheritance Relationships:

Attributes:

```
_all: sigset_t
```

```
_osig: sigset_t
```

POSIX signal blocking functions

Constructors and Destructor:

```
PlBlockSig();
```

```
PlBlockSig(const sigset_t* s);
```

```
virtual  ~PlBlockSig();
```

Operations:

- blockAll

```
void blockAll();
```

- reset

```
void reset();
```

- unblockAll

```
void unblockAll();
```

6.3.1.3.1 PlBoundary Class

Overview:

This class is used to describe boundaries for data acquisition and data processing within PGEs. Processing frequently has to occur organized to some natural boundary (e.g. start of hour, start of day, start of month). This class describes these boundaries and is typically used in conjunction with PlPeriod to describe the processing time needs of PGEs. Notice: this class has been designed to permit the specification of offsets, however these are not implemented fully in Release A.

Export Control: Public

Inheritance Relationships:

Attributes:

`myBoffset: EcTLongInt`

myBoffset: it may also be necessary to specify some offset from the natural boundary, such as 1 hour into the day. NCR 5305 14-Feb-97 TWA Change from unsigned to signed to support negative offsets

`myBseconds: EcTULongInt`

myBseconds: it is easy to calculate the next occurance of a boundary from by finding the next time measured in UTC seconds which is an integer number of myBseconds, e.g. a start of hour boundary would be some integral number times 3600 from UTC(0). The exception here is when the boundary is something such as a month which is a non constant number of seconds, these cases are managed by exception from the norm

`myBtype: enum PlTBoundaryType`

myBtype: enumerated list of boundary types (start of hour etc.)

Constructors and Destructor:

`PlBoundary();`

Default Constructor

`PlBoundary(const RWCString& , EcUtStatus& );`

Constructor from formatted string, the string specifies the boundary (START_OF_SEC, START_OF_MIN, START_OF_6HOUR, START_OF_DAY, START_OF_WEK, START_OF_MONTH) and may optionally specify some offset +/- NO_OF_SECS

```
virtual  ~PlBoundary();
```

Destructor

Operations:

- asString

```
RWCString asString() const;
```

Reformat the boundary as a string, which may be stored in the PDPS database

- GetBoffset

```
const EcTLongInt& GetBoffset();
```

- GetBtype

```
const enum PlTBoundaryType& GetBtype();
```

NCR 5305 14-Feb-97 TWA Add Get functions to support WHERE_CLAUSE

6.3.1.3.2 PlConfig Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

```
myConfigFileP: EcPfConfigFile*
```

Constructors and Destructor:

```
PlConfig();
```

The default constructor, destructor of the interface manager is protected as per singleton design pattern

```
~PlConfig();
```

Operations:

- GetParm

  `static EcUtStatus GetParm(const RWCString& , RWCString& );`

  Retrieve parameter values from the startup configuration file

- SetParm

  `static EcUtStatus SetParm(const RWCString& , const RWCString& );`

  Establish additional parameter=value pairs beyond those which are part of the startup configuration file

6.3.1.3.3 PlDataGranule Class

Overview:

This class describes an individual instance of an ESDT or data file within the PDPS, it may describe either predicted Data Granules, or Granules that are known to exist within the Science Data Server so that they may be This class has a persistent representation in the PDPS as part of the PGE profile, and therefore has the methods to insert and retreive itself from the database.

Export Control: Public

Inheritance Relationships:

Attributes:

`myActualAvailability: PlTime`

myActualAvailablility: time at which the Data Granule was logged as available, recorded as the time at which the PLS subscripiton manager received notification of the Granule

`myAvailability: EcTBoolean`

myAvailability: flag to indicate whether this Data Granule is just predicted to become available at some point, or whether the Granule is actually currently available

`myBaselineTime: PlTime`

myBaselineTime: baselined time for availability of Data Granule

`myDatatypeId: RWCString`

myDataTypeId: reference to the PlDataType class to which this Data Granule belongs

`myDynamicFlag: enum PlTDataType`

myDynamicFlag: type of Data Granule (static, dynamic)

`myEsdtParNameList: RWTValSlist`

myEsdtParmNameList: Granule metadata may vary per Data Type in general the PDPS production system does not need to interigate this Type specific data, however this list allows none-Core data to be accessed for QA checks etc.

`myEsdtParValList: RWTValSlist`

myEsdtParmValList: Granule metadata may vary per Data Type in general the PDPS production system does not need to interigate this Type specific data, however this list allows none-Core data to be accessed for QA checks etc.

`myFileSize: EcTFloat`

myFileSize: file size of the Data Granule (in MBytes) predicted size until the Data is registered to be available

`myGranuleId: RWCString`

myGranuleId: internal id of the Granule that is used as an index to the Granule within the PDPS database, this attibite is usually constructed from the Data Type name and time period of the Data Granule

`myPredictedAvailability: PlTime`

myActualAvailablility: time at which the Data Granule was predicted to be available.

`myScienceGroup: RWCString`

myScienceGroup: attribute describing the nature of the granule, whether it is science data, browse, qa etc. this attribute is used to help build the insert command for the Science Data Server

`myStartTime: PlTime`

myStartTime: describes the data acquisition start time of the Data Granule

`myStopTime: PlTime`

myStopTime: describes the data acquisition end time of the Data Granule

`myUr: RWCString`

myUr: stringified representation of the Universal Reference pointing to the Data Granule within the Science Data Server

Constructors and Destructor:

`PlDataGranule(const RWCString& granuleId);`

Construct Data Granule for given granule ID

`PlDataGranule(const RWCString& , EcUtStatus& );`

Constructor which retrieves the Data Granule from the database given the Granule id.

`PlDataGranule();`

Default constructor

```
~PlDataGranule();
```

Destructor

Operations:

- AddESDTParameterNameList

```
EcTVoid AddESDTParameterNameList(const RWCString& );
```

Add ESDT Parameter to the list

- AddESDTParameterValList

```
EcTVoid AddESDTParameterValList(const RWCString& );
```

Add ESDT Value to the list

- AddESDTParmVal

```
EcUtStatus AddESDTParmVal(const RWCString& , const RWCString& );
```

Add an ESDT parameter and value pair in the PDPS database

- BuildCoreFromParms

```
EcUtStatus BuildCoreFromParms(GlParameterList& parmList);
```

Helper method to extract into the Data Granule attributes from the Science Data Server abstraction of the granule

- BuildFromESDT

```
EcUtStatus BuildFromESDT(DsClESDTReference* dsGranule);
```

Extract into the Data Granule attributes from the Science Data Server abstraction of the granule

- BuildTypeFromESDT

```
EcUtStatus BuildTypeFromESDT(PlDataType& dataType,
DsClESDTReference* dsGranule);
```

Helper method to extract into the Data Granule attributes from the Science Data Server abstraction of the granule

- BuildTypeFromParms

```
EcUtStatus BuildTypeFromParms(PlDataType& dataType,
GlParameterList& parmList);
```

Helper method to extract into the Data Granule attributes from the Science Data Server abstraction of the granule

- ExistsESDTParmVal

```
EcUtStatus ExistsESDTParmVal(const RWCString& , EcTBoolean& );
```

See if exists an ESDT parameter and value pair in the PDPS database

- Exists

```
EcTBoolean Exists();
```

See if granule already exists in the database

- FindAssociatedDprs

```
EcUtStatus FindAssociatedDprs(<any> ...);
```

Find associated Data Processing Requests that require this granule as input

- Format

```
EcTVoid Format(RWCString& , PlTime& , PlTime& );
```

Construct granule id attributes from DataType and data time

- GetActualAvailability

```
const RWTime& GetActualAvailability();
```

- GetAvailability

```
static EcUtStatus GetAvailability(const RWCString& granId,
EcTBoolean& result);
```

Determines whether the Granule as specified by the Granule id is currently registered as available

- GetBaselineTime

```
const RWTime& GetBaselineTime();
```

- GetDatatypeId

```
const RWCString& GetDatatypeId();
```

- GetDynamicFlag

```
const enum PlTDataType GetDynamicFlag();
```

- GetDynamicGranuleIds

```
static RWTValSlist* GetDynamicGranuleIds(PlTime& start, PlTime& stop, RWCString& dataTypeId, RWCString& whereClause, EcUtStatus& returnStatus);
```

Returns a list of granule IDs which fall in the given time range, for the given Data Type, also allows specialization of the query to select these granules by supplying the whereClause

- GetESDTParameterNameList

```
const RWTValSlist& GetESDTParameterNameList();
```

- GetESDTParameterValList

```
const RWTValSlist& GetESDTParameterValList();
```

- GetESDTParmVal

```
EcUtStatus GetESDTParmVal(const RWCString& , RWCString& );
```

Retreive an ESDT parameter and value pair from the PDPS database

- GetFileSize

```
const EcTFloat GetFileSize();
```

- GetGranuleIds

```
static RWTValSlist* GetGranuleIds(PlTime& start, PlTime& stop, RWCString& dataTypeId, RWCString& whereClause, EcUtStatus& returnStatus);
```

Common code from GetDynamicGranules and GetStaticGranules, this method does a token replacement of two fields within the whereClause to specify a query for Data Granules of tha given time. These fields are $DPRSTART and $DPRSTOP and are replaced with the start time and stop time of the processing period. The defult whereClause used is STARTTIME >= $DPRSTART and STOPTIME <= $DPRSTOP however this could be modified to be any SQL permitted syntax to specify some overlap portion etc.

- GetGranuleId

```
const RWCString& GetGranuleId();
```

- GetPredictedAvailability

```
const RWTime& GetPredictedAvailability();
```

- GetScienceGroup

```
const RWCString GetScienceGroup();
```

- GetStartTime

```
const RWTime& GetStartTime();
```

- GetStaticGranuleIds

```
static RWTValSlist* GetStaticGranuleIds(RWCString& dataTypeId,
RWCString& pgeId, RWCString& scienceGroup, RWCString&
whereClause, EcUtStatus& returnStatus);
```

  Returns a list of static granule IDs for the given Data Type and science group, also allows specialization of the query to select these granules by supplying the whereClause NCR 5754 4-Mar-97 TWA Add PgeId to make DATA_TYPE_INSTANCE_ID unique

- GetStopTime

```
const RWTime& GetStopTime();
```

- GetUr

```
const RWCString& GetUr();
```

- InsertESDTParmVal

```
EcUtStatus InsertESDTParmVal(const RWCString& , const RWCString&
);
```

  Insert an ESDT parameter and value pair into the PDPS database

- Insert

```
EcUtStatus Insert();
```

  Insert myself into database.

- InspectAllParms

```
EcUtStatus InspectAllParms(PlDataType& dataType,
DsClESDTReference* dsGranule, GlParameterList& parmList);
```

  Helper method to extract into the Data Granule attributes from the Science Data Server abstraction of the granule

- InspectCoreParms

```
EcUtStatus InspectCoreParms(DsClESDTReference* dsGranule,
GlParameterList& parmList);
```

Helper method to extract into the Data Granule attributes from the Science Data Server abstraction of the granule

- InspectTypeParms

```
EcUtStatus InspectTypeParms(PlDataType& dataType,
DsClESDTReference* dsGranule, GlParameterList& parmList);
```

Helper method to extract into the Data Granule attributes from the Science Data Server abstraction of the granule

- MakeStaticGranuleId

```
static RWCString MakeStaticGranuleId(RWCString& dataTypeId,
RWCString& pgeId, RWCString& scienceGroup);
```

NCR 5754 4-Mar-97 TWA Add PgeId to make DATA_TYPE_INSTANCE_ID unique

- MatchGranule

```
EcUtStatus MatchGranule(EcTBoolean& match, EcTInt& num);
```

- MatchGranule

```
EcUtStatus MatchGranule(EcTBoolean& );
```

Match this granule (as built from UR contained within some subscription notification) to some predicted granule, then update the attributes of the predicted granule

- Read

```
EcUtStatus Read();
```

read the granule identified by the myGranuleId

- SetActualAvailability

```
EcTVoid SetActualAvailability(const RWTime& actualTime);
```

- SetAvailability

```
EcTVoid SetAvailability(const EcTBoolean avail);
```

- SetBaselineTime

```
EcTVoid SetBaselineTime(const RWTime& baselineTime);
```

- SetDatatypeId

```
EcTVoid SetDatatypeId(const RWCString& dataTypeId);
```

- SetDynamicFlag

```
EcTVoid SetDynamicFlag(const enum PlTDataType dynamicFlag);
```

- SetFileSize

```
EcTVoid SetFileSize(const EcTFloat& size);
```

- SetGranuleId

```
EcTVoid SetGranuleId(const RWCString& granuleId);
```

- SetPredictedAvailability

```
EcTVoid SetPredictedAvailability(const RWTime& predictTime);
```

- SetScienceGroup

```
EcTVoid SetScienceGroup(const RWCString& sci);
```

- SetStartTime

```
EcTVoid SetStartTime(const RWTime& startTime);
```

- SetStopTime

```
EcTVoid SetStopTime(const RWTime& stopTime);
```

- SetUr

```
EcTVoid SetUr(const RWCString& Ur);
```

- UpdateESDTParmVal

```
EcUtStatus UpdateESDTParmVal(const RWCString& , const RWCString&
);
```

Update an ESDT parameter and value pair in the PDPS database

- Update

```
EcUtStatus Update();
```

Update myself into the database

- _examineParmList

```
void _examineParmList(const RWTPtrOrderedVector& inspectAttr,
const GlParameterList& parmList) const;
```

helper method examines results of DsClESDTReference::Inspect

6.3.1.3.4 PlDataSourceFactory Class

Overview:

This class provides simple methods to construct the correct type of Data Source object from the persistent data in the database

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
PlDataSourceFactory();
```

Default constructor

```
~PlDataSourceFactory();
```

Destructor method

Operations:

- Create

```
PlDataSource* Create(const RWCString& , EcUtStatus& );
```

Factory method to create the correct subclass of PlDataSource for the Data Type corresponding to the given Data Type ID.

6.3.1.3.5 PlDataSource Class

Overview:

This is a base class in a class heirachy used to describe external data sources of data to enable the PDPS software to predict the arrival times and characteristics of data that will be required as input to PGEs This class has a persistent representation in the PDPS as part of the PGE profile, and therefore has the methods to insert and retreive itself from the database.

Export Control: Public

Inheritance Relationships:

Attributes:

`myDataTypeId: RWCString`

Reference back to the Data Type for which this class decribes the data source of

`myPredictedMethodNum: RWCString`

The specialization type of the Data Source

`mySupplierName: RWCString`

An optional name for the data supplier (e.g. SDPF, EDOS, FDF etc)

Constructors and Destructor:

`PlDataSource();`

Default constructor

`PlDataSource(const RWCString& , EcUtStatus& );`

Constructor which retrieves a data source from the database

`~PlDataSource();`

Destructor

Operations:

- Delete

`EcUtStatus Delete();`

delete the data source from the database

- FindSourceType

```
static RWCString FindSourceType(const RWCString& data_type_id,
EcUtStatus& status);
```

Find for source type of some Data Type from database

- GetDataTypeId

```
const RWCString& GetDataTypeId();
```

- GetPredictedMethodNum

```
const RWCString& GetPredictedMethodNum();
```

- GetSupplierName

```
const RWCString& GetSupplierName();
```

- Insert

```
EcUtStatus Insert();
```

insert the data source into the database

- PredictArrivals

```
virtual EcUtStatus PredictArrivals(PlTime& StartTime, PlTime&
StopTime) = 0;
```

pure virtual method that provides the signature definition for any class to present in order to preduct the data arrivals from the source

- Read

```
EcUtStatus Read(const RWCString& );
```

Read the data source from the database

- SetDataTypeId

```
EcTVoid SetDataTypeId(const RWCString& id);
```

- SetPredictedMethodNum

```
EcTVoid SetPredictedMethodNum(const RWCString& meth);
```

- SetSupplierName

```
EcTVoid SetSupplierName(const RWCString& name);
```

6.3.1.3.6 PlDataTypeReqCollection Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

```
myDataTypeReqCollection: RWTPtrSlist
```

Constructors and Destructor:

```
PlDataTypeReqCollection();
```

Default constructor

```
PlDataTypeReqCollection(RWCString& , EcUtStatus& );
```

construct the collection of parameters for the specific PgeId

```
virtual  ~PlDataTypeReqCollection();
```

destructor method

Operations:

- entries

```
EcTInt entries() const;
```

return the number of entries within the pool

- operator []

```
PlDataTypeReq* operator [](EcTInt i);
```

return the parameter at the indicated index

6.3.1.3.7 PlDataTypeReq Class

Overview:

This class captures the attributes that describe the input relationship between a PGE and a Data Type The class is used when selecting input data for the PGE from the predictions of Data Granules. This data describes helps describe a many to many relationship (many PGEs may input a Data Type, and many Data Types are input into a PGE). This class has a persistent representation in the PDPS as part of the PGE profile, and therefore has the methods to insert and retreive itself from the database.

Export Control: Public

Inheritance Relationships:

Attributes:

`myDataTypeId: RWCString`

myDataTypeId: reference to the Data Type

`myDataTypeReq: EcTInt`

myDataTypeReq: the minimum no. of granules of this input type that is required by the PGE when forming a DPR

`myLinkId: EcTInt`

myLinkId: place holder attribute unused

`myLogicalId: EcTInt`

myLogicalId: the SDP toolkit logical identifier through which the Data Type is referenced

`myPcfFileType: EcTInt`

myPCFFileType: provides information to Data Processing Subsystem for which portion of the Process Control File the data file is referenced from in the SDP toolkit =0, SYSTEM RUNTIME PARAMETERS =1, PRODUCT INPUT FILES =2, PRODUCT OUTPUT FILES =3, SUPPORT INPUT FILES =4, SUPPORT OUTPUT FILES =5, USER DEFINED RUNTIME PARAMETERS =6, SUPPORT INPUT FILES =7, SUPPORT OUTPUT FILES =8, TEMPORARY I/O

`myPgeId: RWCString`

myPgeId: reference to the PGE

`myScienceGroup: RWCString`

myScienceGroup: describes the data group that the input file is associated with NCR 4811 4-Feb-97 TWA Add SCIENCE_GROUP to input file section

myWhereClause: RWCString

myWhereClause: the most common way for a DPR to select it's input data is a straight selection of all the Data Granules of a given type within the data processing period of the Data Processing Request, however, this attribute allows the specification of an override to this selection by specification of an alternative "where clause" for the data. See PlDataGranule::GetGranuleIds for more details on the use of this attribute.

Constructors and Destructor:

PlDataTypeReq();

Default constructor

PlDataTypeReq(const RWCString& pgeId, const RWCString& dataTypeId, const EcTInt& logicalId, EcUtStatus& status);

Construct from database

virtual ~PlDataTypeReq();

Destructor

Operations:

- Delete

EcUtStatus Delete();

Delete

- Exists

EcTBoolean Exists();

See if the record exists in the database

- GetDataTypeId

const RWCString& GetDataTypeId();

- GetDataTypeReq

const EcTInt& GetDataTypeReq();

- GetLogicalId

const EcTInt& GetLogicalId();

- GetPcfFileType

```
const EcTInt& GetPcfFileType();
```

- GetPgeId

```
const RWCString& GetPgeId();
```

- GetScienceGroup

```
const RWCString& GetScienceGroup();
```

NCR 4811 4-Feb-97 TWA Add SCIENCE_GROUP to input file section

- GetWhereClause

```
const RWCString& GetWhereClause();
```

- Insert

```
EcUtStatus Insert();
```

Insert

- PlDisplayDataTypeReq

```
EcTVoid PlDisplayDataTypeReq();
```

Debug method to display the object

- Read

```
EcUtStatus Read(const RWCString& pgeId, const RWCString&
dataTypeId, const EcTInt& logicalId);
```

Read from database

- SetDataTypeId

```
EcTVoid SetDataTypeId(const RWCString& dataTypeId);
```

- SetDataTypeReq

```
EcTVoid SetDataTypeReq(const EcTInt& dataTypeReq);
```

- SetLogicalId

```
EcTVoid SetLogicalId(const EcTInt logicalId);
```

- SetPcfFileType

```
EcTVoid SetPcfFileType(const EcTInt pcfFileType);
```

- SetPgeId

```
EcTVoid SetPgeId(const RWCString& pgeId);
```

- SetScienceGroup

```
EcTVoid SetScienceGroup(const RWCString& group);
```

- SetWhereClause

```
EcTVoid SetWhereClause(const RWCString& whereClause);
```

- Update

```
EcUtStatus Update();
```

Update

6.3.1.3.8 PlDataType Class

Overview:

This class describes a Data Type known to the PDPS and is used to describe the input or output data of a PGE. This class is an abstraction or proxy that describes one of the Science Data Server's ESDTs. The class captures data and operations that are required to subscribe to the ESDT within the Science Data Server, as well as methods used in defining Production Requests and their subsequent explosion into Data Procesing Requests. This class has a persistent representation in the PDPS as part of the PGE profile, and therefore has the methods to insert and retreive itself from the database.

Export Control: Public

Inheritance Relationships:

Attributes:

```
isPoolMember: EcTBoolean
```

isPoolMember: signifies whether this object is a memeber of the pool (in memory resident set) of objects, and if so signifies that when the object is destroyed then it should also be removed from the pool list

`myCatalogueCatagory: RWCString`

myCatalogueCatagory: may be used to

`myDataTypeId: RWCString`

myDataTypeId: the internal id of the datatype within the PDPS database used as a key to retreive the Data Type from the database, this attibute is defined within SSIT and is usually defined as the ESDT shortName

`myDescription: RWCString`

myDescription: comment attribute to describe the usage of the Data Type

`myDServURString: RWCString`

myDServURString: the Universal reference of the Science Data Server that provides the services for the datatype

`myDynamicFlag: enum PlTDataType`

myDynamicFlag: describes whether the Data Type describes dynamic (input or output) data or static data (static Data Types are for calibration files etc, the flag also has to distinguish external dynamic types which have alternative methods to predicting when data are available as opposed to the internal dynamic types, ie. those that are produced by PGEs within a site

`myESDTParmList: RWTValSlist`

myESDTParmList: list used within the inspect to the Data Server, to retreive the DataType specific metadata associated to a newly arrived instance of the Data Type.

`myInstrumentName: RWCString`

myInstrumentName: describes the instrument/s to which the Data Type is associated

`myName: RWCString`

myName: maps the datatype to the ESDT, this is the shortName of the ESDT within the Science Data Server

`myNominalSize: EcTFloat`

myNominalSize: provides a nominal size of the granules of this Data Type for resource size estimates

`myPlatformName: RWCString`

myPlatformName: describes the platform/s (satellite) to which the Data Type is associated

`myProvider: RWCString`

myProvider: describes which DAAC / centre maintains the Data Type used in order to access the appropriate advertisement to the ESDT

`mySubscriptionFlag: EcTBoolean`

mySubscriptionFlag: describes whether a subscription has been set up for the Data Type on behalf of the PDPS production system

`myVersion: RWCString`

myVersion: describes the version of the Data Type

`ourPool: PlObjectPool<class PlDataType>`

ourPool: static list of Data Types provides a standard mechanism to access in memory resident copy of object

Constructors and Destructor:

`PlDataType(const RWCString& , EcUtStatus& );`

Constructor which retrieves a Data Type from the database, given the Data Type id.

`PlDataType(const PlDataType& );`

Copy constructor

`PlDataType();`

Default constructor

`virtual  ~PlDataType();`

Destructor

Operations:

- Delete

`EcUtStatus Delete();`

Delete the Data Type from the database

- Exists

`EcTBoolean Exists();`

See if the Data Type exists in the database

- FindFutureDataAvailability

`EcUtStatus FindFutureDataAvailability(const PlTime& StartTime, const PlTime& StopTime);`

Predict the Data Granules of this Data Type that will be available to service a Production Request

- FindPastDataAvailability

  ```
  EcUtStatus FindPastDataAvailability(const PlTime& StartTime,
  const PlTime& StopTime, PlTime& LatestTime);
  ```

  Query the Data Server to determine what Granule of this Data Type are available to service a Production Request

- GetCatalogueCatagory

  ```
  const RWCString& GetCatalogueCatagory();
  ```

- GetDataTypeId

  ```
  const RWCString& GetDataTypeId();
  ```

- GetDescription

  ```
  const RWCString& GetDescription();
  ```

- GetDServURString

  ```
  const RWCString& GetDServURString();
  ```

- GetDynamicFlag

  ```
  const enum PlTDataType GetDynamicFlag();
  ```

- GetESDTParmList

  ```
  const RWTValSlist& GetESDTParmList();
  ```

- GetInstrumentName

  ```
  const RWCString& GetInstrumentName();
  ```

- GetName

  ```
  const RWCString& GetName();
  ```

- GetNominalSize

```
const EcTFloat GetNominalSize();
```

- GetPlatformName

```
const RWCString& GetPlatformName();
```

- GetProvider

```
const RWCString& GetProvider();
```

- GetSubscriptionFlag

```
const EcTBoolean GetSubscriptionFlag();
```

- InitializePool

```
static EcUtStatus InitializePool();
```

Initialise the pool of objects from the database

- Insert

```
EcUtStatus Insert();
```

Insert the Data Type into the database

- operator ==

```
EcTBoolean operator ==(const PlDataType& );
```

equality operator

- PlDisplayDataType

```
EcTVoid PlDisplayDataType();
```

Debug method displays the Data Type object.

- PlFindDataAvailability

```
EcUtStatus PlFindDataAvailability(const PlTime& StartTime, const
PlTime& StopTime);
```

Finds Data Granules of this Data Type needed by a production request, ensures that either the appropriate method to predict the availability of data has been run, or that the Data Server has been queried to see what historic data exists for this Data Type

- Read

```
EcUtStatus Read(const RWCString& );
```

Read the Data Type from the database

- SetCatalogueCatagory

```
EcTVoid SetCatalogueCatagory(const RWCString& cat);
```

- SetDataTypeId

```
EcTVoid SetDataTypeId(const RWCString& id);
```

- SetDescription

```
EcTVoid SetDescription(const RWCString& desc);
```

- SetDServURString

```
EcTVoid SetDServURString(const RWCString& str);
```

- SetDynamicFlag

```
EcTVoid SetDynamicFlag(const enum PlTDataType flag);
```

- SetInstrumentName

```
EcTVoid SetInstrumentName(const RWCString& instr);
```

- SetName

```
EcTVoid SetName(const RWCString& name);
```

- SetNominalSize

```
EcTVoid SetNominalSize(const EcTFloat size);
```

- SetPlatformName

```
EcTVoid SetPlatformName(const RWCString& name);
```

- SetProvider

```
EcTVoid SetProvider(const RWCString& prov);
```

- SetSubscriptionFlag

```
EcTVoid SetSubscriptionFlag(const EcTBoolean flag);
```

- SetVersion

```
EcTVoid SetVersion(const RWCString& vers);
```

- Submit

```
EcUtStatus Submit(const GlClient& , const EcTBoolean , const
RWCString& serviceName, const RWCString& actionString);
```

Submit a subscription for this Data Type, basic submit is on half of the PDPS production system for notification of insert events within the Data Server, but the method also allows the specification of some other service name and action string.

- Update

```
EcUtStatus Update();
```

Update the Data Type in the database

- Validate

```
EcUtStatus Validate();
```

Validate this Data Type against the Advertising database this method uses the Provider and Name to determine the Universal Reference of the Science Data Server

- Withdraw

```
EcUtStatus Withdraw(const GlClient& , const EcTBoolean , const
RWCString& serviceName);
```

Withdraw the subscription for this Data Type

6.3.1.3.9 PlDbIfManager Class

Overview:

This class provides a simple helper methods to simplify access to the database interface classes. The class follows the singleton design pattern to ensure that the initialization of the class is performed once, when first needed. The database interface classes are specialized per object type, this class provides a standard way of accessing the interface for each class without worying about initialization methods.

Export Control: Public

Inheritance Relationships:

Attributes:

`databaseLibrary: RWCString`

databaseLibrary: database access library (as required by RW dbtools) obtained from process framework config file

`databaseName: RWCString`

databaseName: database name obtained from process framework config file and adjusted according to mode management instructions

`databaseServer: RWCString`

databaseServer: database server name, obtained from process framework config file

`dataGranuleIF: DpPrDbInterface*`

`dataSourceIF: DpPrDbInterface*`

`dataTypeIF: DpPrDbInterface*`

`dataTypeReqCollectionIF: DpPrDbInterface*`

`dataTypeReqIF: DpPrDbInterface*`

`dprIF: DpPrDbInterface*`

`exitDependencyCollectionIF: DpPrDbInterface*`

`exitDependencyIF: DpPrDbInterface*`

`exitMessageCollectionIF: DpPrDbInterface*`

exitMessageIF: DpPrDbInterface*

metaDataChecksCollectionIF: DpPrDbInterface*

metaDataChecksIF: DpPrDbInterface*

outputYieldCollectionIF: DpPrDbInterface*

outputYieldIF: DpPrDbInterface*

overlapArrivalIF: DpPrDbInterface*

performanceIF: DpPrDbInterface*

pgeIF: DpPrDbInterface*

productionRequestIF: DpPrDbInterface*

keep a pointer for all the IF classes

routineArrivalIF: DpPrDbInterface*

rsrcDbIF: DpPrDbInterface*

timeScheduledIF: DpPrDbInterface*

userName: RWCString

userName: user name to be obtained from the process framework

userParametersCollectionIF: DpPrDbInterface*

userParametersIF: DpPrDbInterface*

`userPassword: RWCString`

userPassword: password to be obtained from the process framework

`_instance: PlDbIfManager*`

following the singleton design pattern there should be only one instance of the PlDbIfManager class

Constructors and Destructor:

`PlDbIfManager();`

Default Constructor, protected as per singleton design pattern

`virtual  ~PlDbIfManager();`

Destructor, protected as per singleton design pattern

Operations:

- CloseConnections

  `EcTVoid CloseConnections();`

  Close all the connections made from this class

- GetDatabaseLibrary

  `const RWCString& GetDatabaseLibrary();`

- GetDatabaseName

  `const RWCString& GetDatabaseName();`

- GetDatabaseServer

  `const RWCString& GetDatabaseServer();`

- GetUserName

  `const RWCString& GetUserName();`

- GetUserPassword

  `const RWCString& GetUserPassword();`

- Init

  ```
  static PlDbIfManager* Init(EcUtStatus& , const RWCString&
  dbsKey, const RWCString& srvKey, const RWCString& libKey, const
  EcTLongInt appSeed);
  ```

  The Init method initializes the database information from a configuration file, it uses the process framework config file to retreive the name for the database, server and access library it uses the process framework methods for retrieving secure username and passwords, it adjusts the database name according to to mode management conventions.

- Instance

  ```
  static PlDbIfManager* Instance();
  ```

  A single instance of the class is all that is required within any application, therefore the class follows the singleton design pattern with the following method to construct and return a pointer to the class

- PlDataGranuleIF

  ```
  DpPrDbInterface* PlDataGranuleIF(EcUtStatus& );
  ```

- PlDataSourceIF

  ```
  DpPrDbInterface* PlDataSourceIF(EcUtStatus& );
  ```

- PlDataTypeIF

  ```
  DpPrDbInterface* PlDataTypeIF(EcUtStatus& );
  ```

- PlDataTypeReqCollectionIF

  ```
  DpPrDbInterface* PlDataTypeReqCollectionIF(EcUtStatus& );
  ```

- PlDataTypeReqIF

  ```
  DpPrDbInterface* PlDataTypeReqIF(EcUtStatus& );
  ```

- PlDprIF

  ```
  DpPrDbInterface* PlDprIF(EcUtStatus& );
  ```

- PlExitDependencyCollectionIF

```
DpPrDbInterface* PlExitDependencyCollectionIF(EcUtStatus& );
```

- PlExitDependencyIF

```
DpPrDbInterface* PlExitDependencyIF(EcUtStatus& );
```

- PlExitMessageCollectionIF

```
DpPrDbInterface* PlExitMessageCollectionIF(EcUtStatus& );
```

- PlExitMessageIF

```
DpPrDbInterface* PlExitMessageIF(EcUtStatus& );
```

- PlMetaDataChecksCollectionIF

```
DpPrDbInterface* PlMetaDataChecksCollectionIF(EcUtStatus& );
```

- PlMetaDataChecksIF

```
DpPrDbInterface* PlMetaDataChecksIF(EcUtStatus& );
```

- PlOutputYieldCollectionIF

```
DpPrDbInterface* PlOutputYieldCollectionIF(EcUtStatus& );
```

- PlOutputYieldIF

```
DpPrDbInterface* PlOutputYieldIF(EcUtStatus& );
```

- PlOverlapArrivalIF

```
DpPrDbInterface* PlOverlapArrivalIF(EcUtStatus& );
```

- PlPerformanceIF

```
DpPrDbInterface* PlPerformanceIF(EcUtStatus& );
```

- PlPgeIF

```
DpPrDbInterface* PlPgeIF(EcUtStatus& );
```

- PlProductionRequestIF

```
DpPrDbInterface* PlProductionRequestIF(EcUtStatus& );
```

The following member functions return a database interface class for the appropriate application class type

- PlResourceRequirementIF

```
DpPrDbInterface* PlResourceRequirementIF(EcUtStatus& );
```

- PlRoutineArrivalIF

```
DpPrDbInterface* PlRoutineArrivalIF(EcUtStatus& );
```

- PlTimeScheduledIF

```
DpPrDbInterface* PlTimeScheduledIF(EcUtStatus& );
```

- PlUserParametersCollectionIF

```
DpPrDbInterface* PlUserParametersCollectionIF(EcUtStatus& );
```

- PlUserParametersIF

```
DpPrDbInterface* PlUserParametersIF(EcUtStatus& );
```

6.3.1.3.10 PlDpr Class

Overview:

This class describes a Data Processing Request. A Data Processing Request describes an individual run of a PGE. Data Processing Requests are generated from Production Requests. This class has a persistent representation in the PDPS database so that the Data Processing Request information may be accessed at different times in it's life. A DPR may be planned and scheduled and is used to define the Process Control Information that is used to execute the PGE when that PGE is launched on one of the science processing computers

Export Control: Public

Inheritance Relationships:

Attributes:

`isPoolMember: EcTBoolean`

isPoolMember: signifies whether this object is a memeber of the pool (in memory resident set) of objects, and if so signifies that when the object is destroyed then it should also be removed from the pool list

`myActualStartTime: PlTime`

myActualStartTime: describes the actual start time of the Data Processing Request, ie. the time at which the DPR was executed

`myBaselineTime: PlTime`

myBaselineTime: when a Plan containing a DPR is updated or replanned to account for progress against the schedule the predicted start time will change, it is therefore necessary to baseline the plan at some point in order that the performance of the plan (planned baseline vs actual) provide some meaningful statistic of performance of the planning software

`myCompletionState: RWCString`

myCompletionState: the completion state defines the status of the Data Processing Request within the Data Processing Subsytem

`myDataStartTime: PlTime`

myDataStartTime: defines the start of the acquisistion time of the data that is processed within this Data Processing Request

`myDataStopTime: PlTime`

myDataStopTime: defines the end of the acquisistion time of the data that is processed within this Data Processing Request

`myDprId: RWCString`

myDprId: the id of the Data Processing Request within the PDPS database this attribute is used as a the key to retreive the DPR from the PDPS database, the id is also used to name the job within the Data Processing Subsystem scheduler AutoSys to support a readable description of the job the id is formatted from the PGE id and the period of data that the PGE is processing (time formatted as MMDDYYHHmm)

`myGuiDprCount: EcTInt`

myGuiDprCount: attribute used to quickly index the DPR within a GUI

`myGuiDprPtr: void*`

myGuiDprPtr: memory saving attribute that allows GUI data to be attached to the DPR to avoid having to recalculate this each time a GUI list is built

myInputGranuleList: RWTValSlist

myInputGranuleList: reference to the input Data Granules required as inputs to the Data Processing Request

myInputLogicalIdList: RWTValSlist

myInputLogicalIdList: describes the toolkit logical id on which the input granule will be accessed (is used to build the process control information that is supplied to the PGE at execution time)

myName: RWCString

myName: same as Data Processing Request Id

myOutputGranuleList: RWTValSlist

myOutputGranuleList: reference to the output Data Granules predicted to be output from the Data Processing Request

myOutputLogicalIdList: RWTValSlist

myInputLogicalIdList: describes the toolkit logical id on which the output granule will be accessed (is used to build the process control information that is supplied to the PGE at execution time)

myPgeAvgElapsed: EcTInt

myPgeAvgElapsed: this attribute will be set by the planning workbench to define the average elapsed time that the PGE specified within the PGE takes to run. This is subsequently used by the scheduler to set alarms if the PGE runs over this allottted time. Time measured in seconds

myPgeId: RWCString

myPgeId: reference back to the PGE which is executed within this Data Processing Request

myPredictedStartTime: PlTime

myPredictedStartTime: describes the predicted start time of of the Data Processing Request, ie. the time at which the DPR was planned to be executed

myPriority: EcTInt

myPriority: the priority by which the DPR is scheduled within the Data Processing Subsystem scheduler AutoSys

myProductionRequestId: RWCString

myProductionRequestId: reference back to the Production Request from which this Data Processing Request was defined

mySswId: RWCString

mySswId: reference back to the science software package from which the PGE was built

`ourPool: PlObjectPool`

ourPool: static list of Data Types provides a standard mechanism to access in memory resident copy of object

Constructors and Destructor:

`PlDpr();`

Default constructor

`PlDpr(const RWCString& , EcUtStatus& );`

Constructor which retrieves the Data Processing Request from the database given the DPR id.

`virtual  ~PlDpr();`

Destructor

Operations:

- AddInputGranule

  `EcTVoid AddInputGranule(const RWCString& , const EcTInt& );`

  Add an input granule to the list of Data Granules processed by this Data Processing Request

- AddOutputGranule

  `EcTVoid AddOutputGranule(const RWCString& , const EcTInt& );`

  Add an output granule to the list of Data Granules predicted to be produced by this Data Processing Request

- CheckAvailability

  `EcUtStatus CheckAvailability(const RWCString& , EcTBoolean& );`

  Checks if all input granules for this Data Processing Request are available or not.

- ClearPool

  `static EcUtStatus ClearPool();`

  Clear the pool of Data Procesing Requests

- Delete

  `static EcUtStatus Delete(const RWCString& );`

  Delete all the Data Processing Requests associated to the specified Production Request from the database

- Delete

```
EcUtStatus Delete();
```

Delete the Data Processing Request from the database

- GetActualStartTime

```
const RWTime& GetActualStartTime();
```

- GetBaselineTime

```
const RWTime& GetBaselineTime();
```

- GetCompletionState

```
const RWCString& GetCompletionState();
```

- GetDataStartTime

```
const RWTime& GetDataStartTime();
```

- GetDataStopTime

```
const RWTime& GetDataStopTime();
```

- GetDprId

```
const RWCString& GetDprId();
```

- GetInputGranuleList

```
const RWTValSlist& GetInputGranuleList() const;
```

- GetInputLogicalIdList

```
const RWTValSlist& GetInputLogicalIdList() const;
```

- GetName

```
const RWCString& GetName();
```

- GetOutputGranuleList

```
const RWTValSlist& GetOutputGranuleList() const;
```

- GetOutputLogicalIdList

```
const RWTValSlist& GetOutputLogicalIdList() const;
```

- GetPgeAvgElapsed

```
const EcTInt GetPgeAvgElapsed();
```

- GetPgeId

```
const RWCString& GetPgeId();
```

- GetPredictedStartTime

```
const RWTime& GetPredictedStartTime();
```

- GetPriority

```
const EcTInt GetPriority();
```

- GetProductionRequestId

```
const RWCString& GetProductionRequestId();
```

- GetSswId

```
const RWCString& GetSswId();
```

- InitializePool

```
static EcUtStatus InitializePool();
```

Initialise the pool of objects the database

- InitializePool

```
static EcUtStatus InitializePool(RWCString& prodReqId);
```

Initialise the pool from the database with a reduced set of Data Processing Requests only those associated with the specified Production Request

- Insert

  `EcUtStatus Insert();`

  Insert the Data Processing Request into the database

- isValid

  `EcTBoolean isValid();`

  Verifies that a valid DPR id is assigned and that the priority is reasonable from AutoSys perspective

- operator ==

  `EcTBoolean operator ==(const PlDpr& );`

  Equality operator

- PlDisplayDpr

  `EcTVoid PlDisplayDpr();`

  Debug method to display the Data Processing Request.

- PlRetrieveDprObjects

  `static EcUtStatus PlRetrieveDprObjects(const RWCString& , <any> ...);`

  Retreive all the Data Processing Requests which need this Data Granule as INPUT. This method is used within the subscription manager to process incoming subscription notifications

- PlStoreDpr

  `EcUtStatus PlStoreDpr();`

  Both inserts the Data Processing Request to the database and to the memory resident pool of DPRs

- PlUpdateDprCompletionState

  `EcUtStatus PlUpdateDprCompletionState(const RWCString& );`

  Finds the DPR and updates the completion state database.

- Read

  `EcUtStatus Read(const RWCString& );`

  Read the Data Processing Request from the database

- SetActualStartTime

  `EcTVoid SetActualStartTime(const RWTime& stime);`

- SetBaselineTime

```
EcTVoid SetBaselineTime(const RWTime& base);
```

- SetCompletionState

```
EcTVoid SetCompletionState(const RWCString& state);
```

- SetDataStartTime

```
EcTVoid SetDataStartTime(const RWTime& stime);
```

- SetDataStopTime

```
EcTVoid SetDataStopTime(const RWTime& stime);
```

- SetDprId

```
EcTVoid SetDprId(const RWCString& dprId);
```

- SetName

```
EcTVoid SetName(const RWCString& name);
```

- SetPgeAvgElapsed

```
EcTVoid SetPgeAvgElapsed(const EcTInt elapsed);
```

- SetPgeId

```
EcTVoid SetPgeId(const RWCString& pgeId);
```

- SetPredictedStartTime

```
EcTVoid SetPredictedStartTime(const RWTime& stime);
```

- SetPriority

```
EcTVoid SetPriority(const EcTInt priority);
```

- SetProductionRequestId

  ```
  EcTVoid SetProductionRequestId(const RWCString& prId);
  ```

- SetSswId

  ```
  EcTVoid SetSswId(const RWCString& sswId);
  ```

- Update

  ```
  EcUtStatus Update();
  ```

  Update the Data Processing Request into the database note that this operation does not modify the input and output Data Granule lists, just the basic metadata attributes of the DPR

6.3.1.3.11 PlExitDependencyCollection Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

```
myExitDependencyCollection: RWTPtrSlist
```

Constructors and Destructor:

```
PlExitDependencyCollection();
```

Default constructor

```
PlExitDependencyCollection(RWCString& , EcUtStatus& );
```

construct the collection of parameters for the specific SswId

```
virtual  ~PlExitDependencyCollection();
```

destructor method

Operations:

- entries

```
EcTInt entries() const;
```

return the number of entries within the pool

- operator []

```
PlExitDependency* operator [](EcTInt i);
```

return the parameter at the indicated index

6.3.1.3.12 PlExitDependency Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

```
myDependSswId: RWCString
```

```
myExitCode: EcTInt
```

```
myOperator: RWCString
```

```
mySswId: RWCString
```

Constructors and Destructor:

```
PlExitDependency(const PlExitDependency& exitDependency);
```

Methods needed if exit messages are going to behave well as a member of the ExitDependency collection and RWTValSlists

Copy constructor

```
PlExitDependency(const RWCString& , const RWCString& , const
EcTInt& , EcUtStatus& );
```

Constructor which retrieves from the database

```
PlExitDependency();
```

methods

Default constructor

```
~PlExitDependency();
```

Destructor

Operations:

- Delete

```
EcUtStatus Delete();
```

Delete the exit dependency data from the database

- Exists

```
EcTBoolean Exists();
```

See if the record exists in the database

- GetDependSswId

```
const RWCString& GetDependSswId();
```

- GetExitCode

```
const EcTInt& GetExitCode();
```

- GetOperator

```
const RWCString& GetOperator();
```

- GetSswId

```
const RWCString& GetSswId();
```

public methods to access persistent attributes

- Insert

```
EcUtStatus Insert();
```

Insert the exit dependency data into the database (new ssw)

- operator ==

```
RWBoolean operator ==(const PlExitDependency& );
```

Equality operator returns true if the objects have the same SswId and DependSswId

- operator =

  ```
  PlExitDependency& operator =(const PlExitDependency& );
  ```

  Assignment operator

- Read

  ```
  EcUtStatus Read(const RWCString& sswId, const RWCString&
  dependSswId, const EcTInt& exitCode);
  ```

  Database methods

  Read an exit dependency record from the database

- SetDependSswId

  ```
  EcTVoid SetDependSswId(const RWCString& dependSswId);
  ```

- SetExitCode

  ```
  EcTVoid SetExitCode(const EcTInt& exitCode);
  ```

- SetOperator

  ```
  EcTVoid SetOperator(const RWCString& oper);
  ```

- SetSswId

  ```
  EcTVoid SetSswId(const RWCString& sswId);
  ```

- Update

  ```
  EcUtStatus Update();
  ```

  Update the exit dependency data in the database (existing ssw)

6.3.1.3.13 PlExitMessageCollection Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

```
myExitMessageCollection: RWTPtrSlist
```

Constructors and Destructor:

```
PlExitMessageCollection();
```

Default constructor

```
PlExitMessageCollection(RWCString& , EcUtStatus& );
```

construct the collection of parameters for the specific SswId

```
virtual  ~PlExitMessageCollection();
```

destructor method

Operations:

- entries

```
EcTInt entries() const;
```

return the number of entries within the pool

- operator []

```
PlExitMessage* operator [](EcTInt i);
```

return the parameter at the indicated index

6.3.1.3.14 PlExitMessage Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

```
myExitCode: EcTInt
```

```
myExitMessage: RWCString
```

```
mySswId: RWCString
```

Constructors and Destructor:

```
PlExitMessage(const PlExitMessage& exitMessage);
```

Methods needed if exit messages are going to behave well as a member of the ExitMessage collection and RWTValSlists

Copy constructor

```
PlExitMessage(const RWCString& , const EcTInt& , EcUtStatus& );
```

Constructor which retrieves from the database

```
PlExitMessage();
```

methods

Default constructor

```
~PlExitMessage();
```

Destructor

Operations:

- Delete

```
EcUtStatus Delete();
```

Delete the exit dependency data from the database

- Exists

```
EcTBoolean Exists();
```

See if the record exists in the database

- GetExitCode

```
const EcTInt& GetExitCode();
```

- GetExitMessage

```
const RWCString& GetExitMessage();
```

- GetSswId

```
const RWCString& GetSswId();
```

public methods to access persistent attributes

- Insert

  `EcUtStatus Insert();`

  Insert the exit dependency data into the database (new ssw)

- operator ==

  `RWBoolean operator ==(const PlExitMessage& );`

  Equality operator returns true if the objects have the same PgeId and ExitCode

- operator =

  `PlExitMessage& operator =(const PlExitMessage& );`

  Assignment operator

- Read

  `EcUtStatus Read(const RWCString& sswId, const EcTInt& exitCode);`

  Database methods

  Read an exit message record from the database

- SetExitCode

  `EcTVoid SetExitCode(const EcTInt& exitCode);`

- SetExitMessage

  `EcTVoid SetExitMessage(const RWCString& exitMessage);`

- SetSswId

  `EcTVoid SetSswId(const RWCString& sswId);`

- Update

  `EcUtStatus Update();`

  Update the exit dependency data in the database (existing ssw)

6.3.1.3.15 PlMetaDataChecksCollection Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

```
myMetaDataChecksCollection: RWTPtrSlist
```

Constructors and Destructor:

```
PlMetaDataChecksCollection();
```

Default constructor

```
PlMetaDataChecksCollection(RWCString& , EcUtStatus& );
```

construct the collection of parameters for the specific PgeId

```
virtual  ~PlMetaDataChecksCollection();
```

destructor method

Operations:

- entries

```
EcTInt entries() const;
```

return the number of entries within the pool

- operator []

```
PlMetaDataChecks* operator [](EcTInt i);
```

return the parameter at the indicated index

6.3.1.3.16 PlMetaDataChecks Class

Overview:

This class manages any QA checking that must be performed by the PDPS on behalf of a PGE when notified of arrival of a new Data Granule This class has a persistent representation in the PDPS as part of the PGE profile, and therefore has the methods to insert and retreive itself from the database.

Export Control: Public

Inheritance Relationships:

Attributes:

`myDataTypeId: RWCString`

Reference to the Data Type for which the QA is to be performed

`myMetaDataOper: RWCString`

Describes the operation to be perfomed within the comparison of metadata attribute with it's threshold value (<, >, <=, >=, ==, !=)

`myMetaDataParmName: RWCString`

Describes the metadata parameter name that has to be checked (this is one of the inspect'able attributes of the ESDT within the science data server)

`myMetaDataType: RWCString`

Describes the type of comparison that is performed on the metadata attribute (FLOAT, INT, STR)

`myMetaDataValue: RWCString`

Describes the threshold value that is checked against

`myPgeId: RWCString`

Reference to the PGE for which the QA is to be performed

`ourMetaDataChecksDbIf: DpPrDbInterface`

database interface object for this class

Constructors and Destructor:

`PlMetaDataChecks();`

Default constructor

`PlMetaDataChecks(const RWCString& PgeId, const RWCString& DataTypeId, const RWCString& MetaDataType, const RWCString& MetaDataOper, const RWCString& MetaDataParmName, const RWCString& MetaDataValue);`

Construct by specifying all attributes

`~PlMetaDataChecks();`

Destructor

Operations:

- CheckForCondition

```
static EcUtStatus CheckForCondition(RWCString pgeId,
PlDataGranule& dataGranObj, EcTBoolean& testResult);
```

This function will check if any QA condition exists to be checked and if it exists, does the comparison. If it passes, returns EcDTrue(for testResult); otherwise returns EcDFalse. If condition does not exist, returns EcDTrue.

- CompareActualWithCondition

```
static EcTBoolean CompareActualWithCondition(const RWCString&
actual, const RWCString& condition, RWCString& operStr,
EcUtStatus& returnStatus);
```

Perform a check of the metadata attrubute with the threshold value, return EcDTrue for pass, EcDFalse for fail

- CompareActualWithCondition

```
static EcTBoolean CompareActualWithCondition(EcTFloat actual,
EcTFloat condition, RWCString& operStr, EcUtStatus&
returnStatus);
```

Perform a check of the metadata attrubute with the threshold value, return EcDTrue for pass, EcDFalse for fail

- CompareActualWithCondition

```
static EcTBoolean CompareActualWithCondition(EcTInt actual,
EcTInt condition, RWCString& operStr, EcUtStatus& returnStatus);
```

Perform a check of the metadata attrubute with the threshold value, return EcDTrue for pass, EcDFalse for fail

- ConvertStrToFloat

```
static EcTFloat ConvertStrToFloat(const RWCString& str);
```

conversion routine - string to float

- ConvertStrToInt

```
static EcTInt ConvertStrToInt(const RWCString& str);
```

conversion routine - string to int

- Exists

```
EcTBoolean Exists();
```

See if the record exists in the database

- GetDataTypeId

```
const RWCString& GetDataTypeId();
```

- GetMetaDataOper

```
const RWCString& GetMetaDataOper();
```

- GetMetaDataParmName

```
const RWCString& GetMetaDataParmName();
```

- GetMetaDataType

```
const RWCString& GetMetaDataType();
```

- GetMetaDataValue

```
const RWCString& GetMetaDataValue();
```

- GetPgeId

```
const RWCString& GetPgeId();
```

- Insert

```
EcUtStatus Insert();
```

Insert class data into PDPS database.

- operator ==

```
RWBoolean operator ==(const PlMetaDataChecks& );
```

equality operator returns true if the objects have same PgeId, DataTypeId and MetaDataParmName

- PlDisplayMetaDataChecks

```
EcTVoid PlDisplayMetaDataChecks();
```

Debug method to display class data

- Read

```
static EcUtStatus Read(const RWCString& pgeId, const RWCString&
dataTypeId, <any> ...);
```

Read the class data for the given PGE and Data Type from the PDPS database

- SetDataTypeId

```
EcTVoid SetDataTypeId(const RWCString& );
```

- SetMetaDataOper

```
EcTVoid SetMetaDataOper(const RWCString& metaDataOper);
```

- SetMetaDataParmName

```
EcTVoid SetMetaDataParmName(const RWCString& metaDataParmName);
```

- SetMetaDataType

```
EcTVoid SetMetaDataType(const RWCString& metaDataType);
```

- SetMetaDataValue

```
EcTVoid SetMetaDataValue(const RWCString& metaDataValue);
```

- SetPgeId

```
EcTVoid SetPgeId(const RWCString& );
```

- Update

```
EcUtStatus Update();
```

Updates database.

6.3.1.3.17 PlObjectPoolMsg Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

`object: EcTVoid*`

`reason: enum PlObjectPoolMsg::PlMsReason`

Constructors and Destructor:

Operations:

6.3.1.3.18 PlObjectPool ParameterizedClass

Overview:
class definition

Export Control: Public

Inheritance Relationships:

Attributes:

`changedMsg: PlObjectPoolMsg`
Define a messaging structure to inform a user of the reason of a change in the pool

`myPool: RWTPtrSlist`
The pool is implemented as singly linked ptr list

Constructors and Destructor:

`PlObjectPool();`
Default constructor

`virtual ~PlObjectPool();`
Destructor method

Operations:

- apply

```
EcTVoid apply(EcTVoid (*) (class _Object*, class EcTVoid*)
applyFun, EcTVoid* d);
```

Apply the supplied function to all the members of the pool

- entries

```
EcTInt entries() const;
```

Return the number of entries within the pool

- find

```
_Object* find(const _Object* );
```

Find the specified object in the pool

- insert

```
EcTVoid insert(_Object* );
```

Insert an object into the pool

- operator []

```
_Object*& operator [](EcTInt i);
```

Dereference the nth element of the pool

- remove

```
EcTVoid remove(_Object* );
```

Remove an object from the pool

6.3.1.3.19 PlOutputYieldCollection Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

```
myOutputYieldCollection: RWTPtrSlist
```

Constructors and Destructor:

`PlOutputYieldCollection();`

Default constructor

`PlOutputYieldCollection(RWCString& , EcUtStatus& );`

construct the collection of parameters for the specific PgeId

`virtual ~PlOutputYieldCollection();`

destructor method

Operations:

- entries

`EcTInt entries() const;`

return the number of entries within the pool

- operator []

`PlOutputYield* operator [](EcTInt i);`

return the parameter at the indicated index

6.3.1.3.20 PlOutputYield Class

Overview:

This class captures the attributes that describe the output relationship between a PGE and a Data Type The class is used when predicting the output data for the PGE that will be produced for a given Data Processing Request. This class has a persistent representation in the PDPS as part of the PGE profile, and therefore has the methods to insert and retreive itself from the database.

Export Control: Public

Inheritance Relationships:

Attributes:

`myDataTypeId: RWCString`

myDataTypeId: reference to the Data Type

`myInstance: EcTInt`

myInstance: if many granules produced of the same type but on different Logical id's this attribute describes which instance of the data is to be assigned to the id (assuming that each

instance contains the same temporal coverage, which is given by the period of the Data Processing Request divided by the yield)

`myLogicalId: EcTInt`

myLogicalId: the SDP toolkit logical identifier through which the Data Type is referenced

`myPcfFileType: EcTInt`

myPCFFileType: provides information to Data Processing Subsystem for which portion of the Process Control File the data file is referenced from in the SDP toolkit =0, SYSTEM RUNTIME PARAMETERS =1, PRODUCT INPUT FILES =2, PRODUCT OUTPUT FILES =3, SUPPORT INPUT FILES =4, SUPPORT OUTPUT FILES =5, USER DEFINED RUNTIME PARAMETERS =6, SUPPORT INPUT FILES =7, SUPPORT OUTPUT FILES =8, TEMPORARY I/O

`myPgeId: RWCString`

myPgeId: reference to the PGE

`myScienceGroup: RWCString`

myScienceGroup: describes the data group that the output file is associated to, used to build the data server insert command

`myYield: EcTInt`

myYield: the number of output granules that will be produced for this PGE, Data Type, Logical Id combination

Constructors and Destructor:

`PlOutputYield();`

Default constructor

`PlOutputYield(const RWCString& PgeId, const RWCString& DataTypeId, const EcTInt& logicalId, EcUtStatus& status);`

Construct from database

`virtual ~PlOutputYield();`

Destructor

Operations:

- Delete

`EcUtStatus Delete();`

Delete the class data from the PDPS database

- Exists

```
EcTBoolean Exists();
```

See if the record exists in the database

- GetDataTypeId

```
const RWCString& GetDataTypeId();
```

- GetInstance

```
const EcTInt& GetInstance();
```

- GetLogicalId

```
const EcTInt& GetLogicalId();
```

- GetPcfFileType

```
const EcTInt& GetPcfFileType();
```

- GetPgeId

```
const RWCString& GetPgeId();
```

- GetScienceGroup

```
const RWCString& GetScienceGroup();
```

- GetYield

```
const EcTInt& GetYield();
```

- Insert

```
EcUtStatus Insert();
```

Insert class data to PDPS database

- PlDisplayOutputYield

```
EcTVoid PlDisplayOutputYield();
```

Debug method to display class data

- Read

```
EcUtStatus Read(const RWCString& pgeId, const RWCString&
dataTypeId, const EcTInt& logicalId);
```

Read the class data from the PDPS database

- SetDataTypeId

```
EcTVoid SetDataTypeId(const RWCString& dataTypeId);
```

- SetInstance

```
EcTVoid SetInstance(const EcTInt& inst);
```

- SetLogicalId

```
EcTVoid SetLogicalId(const EcTInt& logicalId);
```

- SetPcfFileType

```
EcTVoid SetPcfFileType(const EcTInt pcfFileType);
```

- SetPgeId

```
EcTVoid SetPgeId(const RWCString& pgeId);
```

- SetScienceGroup

```
EcTVoid SetScienceGroup(const RWCString& group);
```

- SetYield

```
EcTVoid SetYield(const EcTInt& yield);
```

- Update

```
EcUtStatus Update();
```

Update class data in the PDPS database

6.3.1.3.21 PlOverlapArrival Class

Overview:

This class is a specialization of the Data Source class used to describe periodic routine arrival of data from an external data provider such as FDF. The FDF is different in that the The data arrives daily and covers the current day as well as a prediction for the following n days.

Export Control: Public

Inheritance Relationships:

Inherits from PlDataSource

Attributes:

`myBoundary: PlBoundary`

myBoundary: The natural boundary to which the data is alligned (e.g. start of day)

`myDelay: EcTLongInt`

myDelay: The delay (seconds) between end of acquisistion of the data and the anticipated arrival of the data in the ECS

`myPeriod: PlPeriod`

myPeriod: The period which the data covers from the boundary (e.g. 24 hours)

Constructors and Destructor:

`PlOverlapArrival();`

Default constructor

`PlOverlapArrival(const RWCString& , EcUtStatus& );`

constructor which retrieves clas data from the database

`~PlOverlapArrival();`

Destructor

Operations:

- Delete

`EcUtStatus Delete();`

Delete the class attributes from the database

- GetBoundary

`const PlBoundary& GetBoundary();`

- GetDataTypeId

```
const RWCString& GetDataTypeId();
```

- GetDelay

```
const EcTLongInt& GetDelay();
```

- GetPeriod

```
const PlPeriod& GetPeriod();
```

- Insert

```
EcUtStatus Insert();
```

Insert the class attributes into the database

- PredictArrivals

```
EcUtStatus PredictArrivals(PlTime& StartTime, PlTime& StopTime);
```

Predict arrivals of Data Granules between the given start and stop time

- Read

```
EcUtStatus Read(const RWCString& );
```

Read the class attributes from the database

- SetBoundary

```
EcTVoid SetBoundary(const PlBoundary& boundary);
```

- SetDataTypeId

```
EcTVoid SetDataTypeId(const RWCString& id);
```

- SetDelay

```
EcTVoid SetDelay(const EcTLongInt& delay);
```

- SetPeriod

```
EcTVoid SetPeriod(const PlPeriod& period);
```

6.3.1.3.22 PlPerformance Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

```
myDPRElapsedTime: EcTFloat

myMaxMemoryUse: EcTFloat

myNoOfBlockInOper: EcTInt

myNoOfBlockOutOper: EcTInt

myNoOfPageFaults: EcTInt

myNoOfSwaps: EcTInt

myPGECPUTime: EcTFloat

myPGEElapsedTime: EcTFloat

myPgeId: RWCString

myRunDPRElapsedTime: EcTFloat
```

myRunMaxMemoryUse: EcTFloat

myRunNoOfBlockInOper: EcTInt

myRunNoOfBlockOutOper: EcTInt

myRunNoOfPageFaults: EcTInt

myRunNoOfSwaps: EcTInt

myRunPGECPUTime: EcTFloat

myRunPGEElapsedTime: EcTFloat

Constructors and Destructor:

PlPerformance();
default constructor

PlPerformance(const RWCString& PgeId, EcUtStatus& status);
construct from database

virtual ~PlPerformance();
destructor

Operations:

- Delete

  EcUtStatus Delete();
  database delete

- Exists

  EcTBoolean Exists();
  See if the record exists in the database

- GetDPRElapsedTime

```
const EcTFloat& GetDPRElapsedTime();
```

- GetMaxMemoryUse

```
const EcTFloat& GetMaxMemoryUse();
```

- GetNoOfBlockInOper

```
const EcTInt& GetNoOfBlockInOper();
```

- GetNoOfBlockOutOper

```
const EcTInt& GetNoOfBlockOutOper();
```

- GetNoOfPageFaults

```
const EcTInt& GetNoOfPageFaults();
```

- GetNoOfSwaps

```
const EcTInt& GetNoOfSwaps();
```

- GetPGECPUTime

```
const EcTFloat& GetPGECPUTime();
```

- GetPGEElapsedTime

```
const EcTFloat& GetPGEElapsedTime();
```

- GetPgeId

```
const RWCString& GetPgeId();
```

public methods to access persistent attributes

- GetRunDPRElapsedTime

```
const EcTFloat& GetRunDPRElapsedTime();
```

- GetRunMaxMemoryUse

```
const EcTFloat& GetRunMaxMemoryUse();
```

- GetRunNoOfBlockInOper

```
const EcTInt& GetRunNoOfBlockInOper();
```

- GetRunNoOfBlockOutOper

```
const EcTInt& GetRunNoOfBlockOutOper();
```

- GetRunNoOfPageFaults

```
const EcTInt& GetRunNoOfPageFaults();
```

- GetRunNoOfSwaps

```
const EcTInt& GetRunNoOfSwaps();
```

- GetRunPGEElapsedTime

```
const EcTFloat& GetRunPGEElapsedTime();
```

- Insert

```
EcUtStatus Insert();
```

database insert

- PlDisplayPerformance

```
EcTVoid PlDisplayPerformance();
```

print

- Read

```
EcUtStatus Read(const RWCString& pgeId);
```

database read

- SetDPRElapsedTime

```
EcTVoid SetDPRElapsedTime(const EcTFloat& DPRElapsedTime);
```

- SetMaxMemoryUse

```
EcTVoid SetMaxMemoryUse(const EcTFloat& MaxMemoryUse);
```

- SetNoOfBlockInOper

```
EcTVoid SetNoOfBlockInOper(const EcTInt& NoOfBlockInOper);
```

- SetNoOfBlockOutOper

```
EcTVoid SetNoOfBlockOutOper(const EcTInt& NoOfBlockOutOper);
```

- SetNoOfPageFaults

```
EcTVoid SetNoOfPageFaults(const EcTInt& NoOfPageFaults);
```

- SetNoOfSwaps

```
EcTVoid SetNoOfSwaps(const EcTInt& NoOfSwaps);
```

- SetPGECPUTime

```
EcTVoid SetPGECPUTime(const EcTFloat& PGECPUTime);
```

- SetPGEElapsedTime

```
EcTVoid SetPGEElapsedTime(const EcTFloat& PGEElapsedTime);
```

- SetPgeId

```
EcTVoid SetPgeId(const RWCString& pgeId);
```

- SetRunDPRElapsedTime

```
EcTVoid SetRunDPRElapsedTime(const EcTFloat& runDPRElapsedTime);
```

- SetRunMaxMemoryUse

```
EcTVoid SetRunMaxMemoryUse(const EcTFloat& runMaxMemoryUse);
```

- SetRunNoOfBlockInOper

```
EcTVoid SetRunNoOfBlockInOper(const EcTInt& runNoOfBlockInOper);
```

- SetRunNoOfBlockOutOper

```
EcTVoid SetRunNoOfBlockOutOper(const EcTInt&
runNoOfBlockOutOper);
```

- SetRunNoOfPageFaults

```
EcTVoid SetRunNoOfPageFaults(const EcTInt& runNoOfPageFaults);
```

- SetRunNoOfSwaps

```
EcTVoid SetRunNoOfSwaps(const EcTInt& runNoOfSwaps);
```

- SetRunPGECPUTime

```
EcTVoid SetRunPGECPUTime(const EcTFloat& runPGECPUTime);
```

- SetRunPGEElapsedTime

```
EcTVoid SetRunPGEElapsedTime(const EcTFloat& runPGEElapsedTime);
```

- Update

```
EcUtStatus Update();
```

update

6.3.1.3.23 PlPeriod Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

```
myPeriod: EcTULongInt

myPtype: enum PlTPeriodType
```

Constructors and Destructor:

```
PlPeriod(const RWCString& , EcUtStatus& );

PlPeriod(const EcTULongInt& );

PlPeriod();

 ~PlPeriod();
```

Operations:

- asString

```
RWCString asString() const;
```

6.3.1.3.24 PlPgeFactory Class

Overview:

This class provides simple methods to construct the correct type of PGE object from the persistent data in the database

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

`PlPgeFactory();`

Class constructor

`~PlPgeFactory();`

Destructor

Operations:

- Create

  `PlPge* Create(const RWCString& pgeId, EcUtStatus& status);`

  Factory method to create the appropriate subclass of PlPge for the PGE corresponding to the given PGE ID.

- Create

  `PlPge* Create(const RWCString& pgeId, const RWCString& pgeTypeString, EcUtStatus& status);`

  Factory method to create the correct subclass of PlPge for the PGE corresponding to the given PGE ID and type. used for efficiency reasons to reduce database access in certain circumstances

6.3.1.3.25 PlPge Class

Overview:

This class is the base class within a generalization heirachy that describes Product Generation Executives (PGEs) to the PDPS software, it defines the abstract operations required in order that Planning Subsystem can work out when a PGE requires to be scheduled. The class contains key data attributes required to define a PGE. This class has a persistent representation in the PDPS as part of the PGE profile, and therefore has the methods to insert and retreive itself from the database.

Export Control: Public

Inheritance Relationships:

Attributes:

`isPoolMember: EcTBoolean`

isPoolMember: signifies whether this object is a memeber of the pool (in memory resident set) of objects, and if so signifies that when the object is destroyed then it should also be removed from the pool list

`myInputDatatypeList: RWTValSlist`

myInputDatatypeList: list of references to the Data Types required as input to the PGE

myInputLogicalIdList: RWTValSlist

myInputLogicalIdList: SDP toolkit logical id that is used as by PGE to reference input of a Data Type

myInstrument: RWCString

myInstrument: describes the instrument/s to which the PGE is associated

myOutputDatatypeList: RWTValSlist

myOuputDatatypeList: list of references to the Data Types produced as output to the PGE

myOutputLogicalIdList: RWTValSlist

myOuputLogicalIdList: SDP toolkit logical id that is used as by PGE to reference input of a Data Type

myPgeId: RWCString

myPgeId: internal id of the PGE that is used as an index to the PGE within the PDPS database, this attibite is defined within SSIT and is usually constructed from the PGE name and version

myPgeName: RWCString

myPgeName: name associated to a PGE (may be many versions of the same PGE name), defined in SSIT

myPgeVersion: RWCString

myPgeVersion: version of PGE

myPlatform: RWCString

myPlatformName: describes the platform/s (satellite) to which the PGE is associated

mySswId: RWCString

mySswId: several PGEs may be associated with the same Science Software id, this would (for example) describe different versions of the PGE using different callibration files, but the same basic executable

myTestOperational: RWCString

myTestOperational: flag used within SSIT

myType: RWCString

myType: describes which "type" or specialization of PGE this is, Release A currently supports "TimeScheduled" PGEs, see sub-class for details

ourPool: PlObjectPool<class PlPge>

ourPool: static list of Data Types provides a standard mechanism to access in memory resident copy of object

Constructors and Destructor:

```
PlPge();
```

Default constructor

```
PlPge(const RWCString& , EcUtStatus& );
```

Constructor which retrieves a PGE from the database, given the PGE id.

```
virtual  ~PlPge();
```

Destructor

Operations:

- AddInputDataType

```
EcTVoid AddInputDataType(const RWCString& , const EcTInt& );
```

Add an association to an Input Data Type required by this PGE to be accessed on the specified SDP toolkit logical id

- AddOutputDataType

```
EcTVoid AddOutputDataType(const RWCString& , const EcTInt& );
```

Add an association to an Output Data Type produced by this PGE to be accessed on the specified SDP toolkit logical id

- Delete

```
virtual EcUtStatus Delete();
```

Delete the PGE from the database

- Exists

```
virtual EcTBoolean Exists();
```

Determine if the PGE exists in the database

- FindDataAvailability

```
virtual EcUtStatus FindDataAvailability(PlTime& StartTime,
PlTime& StopTime);
```

Find the Data Granule availability for this PGE by iterating over the input data type list, and invoking the FindDataAvaialability method for each input data type.

- FindPgeType

```
static RWCString FindPgeType(const RWCString& , EcUtStatus& );
```

Find out the type of a PGE from the database

- GetInputDataTypeList

```
const RWTValSlist& GetInputDataTypeList();
```

- GetInstrument

```
const RWCString& GetInstrument();
```

- GetOutputDataTypeList

```
const RWTValSlist& GetOutputDataTypeList();
```

- GetPgeId

```
const RWCString& GetPgeId();
```

- GetPgeName

```
const RWCString& GetPgeName();
```

- GetPgeVersion

```
const RWCString& GetPgeVersion();
```

- GetPlatform

```
const RWCString& GetPlatform();
```

- GetSswId

```
const RWCString& GetSswId();
```

- GetTestOperational

```
const RWCString& GetTestOperational();
```

- GetType

```
const RWCString& GetType();
```

- InitializePool

```
static EcUtStatus InitializePool();
```

Initialise the pool from the database

- Insert

```
virtual EcUtStatus Insert();
```

Insert the PGE into the database

- operator ==

```
EcTBoolean operator ==(const PlPge& );
```

Equality operator

- PlDisplayPge

```
EcTVoid PlDisplayPge();
```

Debug methid displays a PGE object.

- PlGenerateDprs

```
virtual EcUtStatus PlGenerateDprs(PlTime& startTime, PlTime&
stopTime, RWCString& prodReqId, EcTInt& dprCount);
```

Generate Data Processing Requests for this PGE based on the instructions given within the referenced Production Request.

- PlGetInputForDpr

```
virtual EcUtStatus PlGetInputForDpr(PlDpr& dpr, PlTime&
startTime, PlTime& stopTime);
```

Gets input Data Granules required by this PGE based on the specified period, and associate these to the Data Processing Request being generated.

- PlGetOutputForDpr

```
virtual EcUtStatus PlGetOutputForDpr(PlDpr& dpr, PlTime&
startTime, PlTime& stopTime);
```

Predicts output Data Granules produced by this PGE based on the specified period, and associate these to the Data Processing Request being generated

- PlStorePge

```
EcUtStatus PlStorePge();
```

Insert a PGE to the database and add to the pool

- Read

```
virtual EcUtStatus Read(const RWCString& );
```

Read the PGE from the database

- SetInstrument

```
EcTVoid SetInstrument(const RWCString& instrument);
```

- SetPgeId

```
EcTVoid SetPgeId(const RWCString& pgeId);
```

- SetPgeName

```
EcTVoid SetPgeName(const RWCString& name);
```

- SetPgeVersion

```
EcTVoid SetPgeVersion(const RWCString& vers);
```

- SetPlatform

```
EcTVoid SetPlatform(const RWCString& platform);
```

- SetSswId

```
EcTVoid SetSswId(const RWCString& id);
```

- SetTestOperational

```
EcTVoid SetTestOperational(const RWCString& test);
```

- SetType

```
EcTVoid SetType(const RWCString& type);
```

- Update

```
virtual EcUtStatus Update();
```

Update the PGE in the database

6.3.1.3.26 PlProductionRequest Class

Overview:

A Production Request describes the need for the scheduling of some PGE to produce a series of Data Granules. A Production Request typically describes a request for a PGE to be scheduled to process data for some extended period of time for example a month's worth of data, the Production Request provides the methods to "explode" the request into a set of Data Processing Requests, or distinct jobs to run PGEs on input Data Granules

Export Control: Public

Inheritance Relationships:

Attributes:

`dprCount: EcTInt`

`isPoolMember: EcTBoolean`

isPoolMember: signifies whether this object is a memeber of the pool (in memory resident set) of objects, and if so signifies that when the object is destroyed then it should also be removed from the pool list

`myComments: RWCString`

myComments: optional comments which may accompany the Production Request

`myDataCollectionStartTime: PlTime`

myDataCollectionStartTime: start date time for the data which is to be processed

`myDataCollectionStopTime: PlTime`

myDataCollectionStopTime: end date time for the data which is to be processed

`myGuiDprCount: EcTInt`

Reserved for Gui Use // PDURAO 4/26/96 myGuiDprCount: attribute used to quickly index the DPR within a GUI

`myGuiDprPtr: void*`

myGuiDprPtr: memory saving attribute that allows GUI data to be attached to the DPR to avoid having to recalculate this each time a GUI list is built

`myOriginationDate: PlTime`

list of parameter overrides myOriginationDate: date at which the Production Request was entered

myOutputDatatypeId: RWCString

myOutputDatatypeId: alternative to reference the PGE as to that which produces this Data Type, unused at Rel A

myPgeId: RWCString

myPgeId: reference to the PGE which has to be scheduled for this Production Request

myPriority: EcTInt

myPriority: the default priority at which the Production Request will be planned

myRequesterId: RWCString

myRequesterId: user identifier of the requestor who placed the Production Request

myRequestId: RWCString

myRequestId: provides a name against which the PR may be tracked within the Production System. No restrictions are placed on any naming convention for the Production Requests

myUsrParaList: RWTValSlist

myUsrParaList: optional specification of a SDP toolkit run time parameter id for which an override to the default parameter value is to be supplied to the PGE at execution time

myUsrParaValueList: RWTValSlist

list of parameter ids myUserParaValueList: optional specification of the value of the override for the SDP toolkit run time parameter

ourPool: PlObjectPool<class PlProductionRequest>

ourPool: static list of Data Types provides a standard mechanism to access in memory resident copy of object

Constructors and Destructor:

PlProductionRequest(const PlProductionRequest& productionRequestObj);

Copy constructor

PlProductionRequest(const RWCString& , EcUtStatus&);

Constructor which retrieves a production request from the database.

PlProductionRequest();

Default constructor

virtual ~PlProductionRequest();

Destructor

Operations:

- AddUsrParameter

  ```
  EcTVoid AddUsrParameter(EcTInt , const RWCString& );
  ```

  Add a parameter id and value to the list of overrides

- ClearUsrParameters

  ```
  EcTVoid ClearUsrParameters();
  ```

  Clear the list of User Parameter overrides

- Delete

  ```
  EcUtStatus Delete();
  ```

  Delete the production request from the database

- GetComment

  ```
  const RWCString& GetComment();
  ```

- GetDataCollectionStartTime

  ```
  const RWTime& GetDataCollectionStartTime();
  ```

- GetDataCollectionStopTime

  ```
  const RWTime& GetDataCollectionStopTime();
  ```

- GetDateOfReq

  ```
  const RWTime& GetDateOfReq();
  ```

- GetmyUsrParaList

  ```
  const RWTValSlist& GetmyUsrParaList();
  ```

- GetOutputDatatypeID

  ```
  const RWCString& GetOutputDatatypeID();
  ```

- GetPgeID

```
const RWCString& GetPgeID();
```

- GetPriority

```
const EcTInt GetPriority();
```

- GetRequesterID

```
const RWCString& GetRequesterID();
```

- GetRequestID

```
const RWCString& GetRequestID();
```

- GetUsrParaValist

```
const RWTValSlist& GetUsrParaValist();
```

- InitializePool

```
static EcUtStatus InitializePool();
```

Initialise the pool of objects from the database

- Insert

```
EcUtStatus Insert();
```

Insert the production request into the database

- operator ==

```
EcTBoolean operator ==(const PlProductionRequest& );
```

Equality operator

- PlDefinePgeRuns

```
EcUtStatus PlDefinePgeRuns();
```

Exploding the production request into corresponding Data Processing Requests (DPRs).

- PlDisplayProductionRequest

```
EcTVoid PlDisplayProductionRequest();
```

Debug method to display class atributes

- PlStoreProductionRequest

```
EcUtStatus PlStoreProductionRequest();
```

store a production request into the database and add to in memory pool

- Read

```
EcUtStatus Read(RWCString );
```

Read the production request from the database

- SetComment

```
EcTVoid SetComment(const RWCString& comment);
```

- SetDataCollectionStartTime

```
EcTVoid SetDataCollectionStartTime(const RWTime& stime);
```

- SetDataCollectionStopTime

```
EcTVoid SetDataCollectionStopTime(const RWTime& stime);
```

- SetDateOfReq

```
EcTVoid SetDateOfReq(const RWTime& stime);
```

- SetOutputDatatypeID

```
EcTVoid SetOutputDatatypeID(const RWCString& dtype);
```

- SetPgeID

```
EcTVoid SetPgeID(const RWCString& pgeId);
```

- SetPriority

```
EcTVoid SetPriority(const EcTInt prior);
```

- SetRequesterID

```
EcTVoid SetRequesterID(const RWCString& id);
```

- SetRequestID

```
EcTVoid SetRequestID(const RWCString& id);
```

6.3.1.3.27 PlProfile Class

Overview:

This is a simple class to create a single instance of the client information class required by DSS when using any of it's client interfaces following the Singleton design pattern there should be only one instance of the PlProfile class

Export Control: Public

Inheritance Relationships:

Attributes:

`_client: GlClient*`

Attribute to describe the client info

`_instance: PlProfile*`

Attribute to store pointer to the single instance of this class

`_status: EcUtStatus`

Status of construction of client info

Constructors and Destructor:

`PlProfile();`

The default constructor protected as per singleton design pattern

`~PlProfile();`

The destructor protected as per singleton design pattern

Operations:

- getProfile

```
const GlClient& getProfile(EcUtStatus& );
```

return the client info

- Instance

```
static PlProfile* Instance();
```

static method to retreive the single instance of the class, if the class hasn't been instantiated before then this method does so

- WhoAmI

  `static RWCString WhoAmI();`

  Find out who i am

6.3.1.3.28 PlResourceRequirement Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

`myComputer: RWCString`

`myDeleteFlag: EcTInt`

`myDiskSpace: EcTFloat`

`myExeTarFileDiskSpace: EcTFloat`

`myExeTarFileUr: RWCString`

`myExeUnTarFileDiskSpace: EcTFloat`

`myMcfName: RWCString`

`myNCPUs: EcTInt`

`myRamSize: EcTFloat`

```
mySswId: RWCString
```

```
myString: RWCString
```

```
myTopLevelShellName: RWCString
```

Constructors and Destructor:

```
PlResourceRequirement();
```

Constructor and Destructor methods.

```
PlResourceRequirement(const RWCString& sswId, EcUtStatus&
status);
```

Constructor takes SSW ID in order to resurrect the class from the Database.

```
~PlResourceRequirement();
```

Operations:

- Delete

```
EcUtStatus Delete();
```

delete the resource requirements from the database

- Exists

```
EcTBoolean Exists();
```

See if the record exists in the database

- GetComputer

```
const RWCString& GetComputer();
```

- GetDeleteFlag

```
EcTInt GetDeleteFlag();
```

- GetDiskSpace

```
EcTFloat GetDiskSpace();
```

- GetExeTarFileDiskSpace

```
EcTFloat GetExeTarFileDiskSpace();
```

- GetExeTarFileUr

```
const RWCString& GetExeTarFileUr();
```

- GetExeUnTarFileDiskSpace

```
EcTFloat GetExeUnTarFileDiskSpace();
```

- GetMcfName

```
const RWCString& GetMcfName();
```

- GetNCPUs

```
EcTInt GetNCPUs();
```

- GetRamSize

```
EcTFloat GetRamSize();
```

- GetSswId

```
const RWCString& GetSswId();
```

gets and sets

- GetString

```
const RWCString& GetString();
```

- GetTopLevelShellName

```
const RWCString& GetTopLevelShellName();
```

- Insert

```
EcUtStatus Insert();
```

insert the resource requirements into the database

- Read

```
EcUtStatus Read(const RWCString& sswId);
```

Read a resource requirement record from the database

- SetComputer

```
EcTVoid SetComputer(const RWCString& computer);
```

- SetDeleteFlag

```
EcTVoid SetDeleteFlag(const EcTInt& deleteFlag);
```

- SetDiskSpace

```
EcTVoid SetDiskSpace(const EcTFloat& diskSpace);
```

- SetExeTarFileDiskSpace

```
EcTVoid SetExeTarFileDiskSpace(const EcTFloat&
exeTarFileDiskSpace);
```

- SetExeTarFileUr

```
EcTVoid SetExeTarFileUr(const RWCString& exeTarFileUr);
```

- SetExeUnTarFileDiskSpace

```
EcTVoid SetExeUnTarFileDiskSpace(const EcTFloat&
exeUnTarFileDiskSpace);
```

- SetMcfName

```
EcTVoid SetMcfName(const RWCString& mcfName);
```

- SetNCPUs

```
EcTVoid SetNCPUs(const EcTInt& nCPUs);
```

- SetRamSize

```
EcTVoid SetRamSize(const EcTFloat& ramSize);
```

- SetSswId

```
EcTVoid SetSswId(const RWCString& id);
```

- SetString

```
EcTVoid SetString(const RWCString& string);
```

- SetTopLevelShellName

```
EcTVoid SetTopLevelShellName(const RWCString&
topLevelShellName);
```

- Update

```
EcUtStatus Update();
```

update the resource requirements in the database

6.3.1.3.29 PlRoutineArrival Class

Overview:

This class is a specialization of the Data Source class used to describe periodic routine arrival of data from an external data provider such as SDPF. The class predicts both the characteristics of the data (temporal coverage) and the arrival of the data

Export Control: Public

Inheritance Relationships:

Inherits from PlDataSource

Attributes:

```
myBoundary: PlBoundary
```

myBoundary: The natural boundary to which the data is alligned (e.g. start of day)

```
myDelay: EcTLongInt
```

myDelay: The delay (seconds) between end of acquisistion of the data and the anticipated arrival of the data in the ECS

`myPeriod: PlPeriod`

myPeriod: The period which the data covers from the boundary (e.g. 24 hours)

Constructors and Destructor:

`PlRoutineArrival();`

Default constructor

`PlRoutineArrival(const RWCString& , EcUtStatus& );`

Constructor which retrieves a routine arrival from the database

`~PlRoutineArrival();`

Destructor

Operations:

- Delete

  `EcUtStatus Delete();`

  Delete the routine arrival from the database

- Exists

  `EcTBoolean Exists();`

  See if the record exists in the database

- GetBoundary

  `const PlBoundary& GetBoundary();`

- GetDataTypeId

  `const RWCString& GetDataTypeId();`

- GetDelay

  `const EcTLongInt& GetDelay();`

- GetPeriod

  `const PlPeriod& GetPeriod();`

- Insert

  `EcUtStatus Insert();`

  Insert the routine arrival into the database

- PredictArrivals

  `EcUtStatus PredictArrivals(PlTime& StartTime, PlTime& StopTime);`

  Predict arrivals of Data Granules between the given start and stop time

- Read

  `EcUtStatus Read(const RWCString& );`

  Read the routine arrival from the database

- SetBoundary

  `EcTVoid SetBoundary(const PlBoundary& boundary);`

- SetDataTypeId

  `EcTVoid SetDataTypeId(const RWCString& id);`

- SetDelay

  `EcTVoid SetDelay(const EcTLongInt& delay);`

- SetPeriod

  `EcTVoid SetPeriod(const PlPeriod& period);`

- Update

  `EcUtStatus Update();`

  Update the routine arrival in the database

6.3.1.3.30 PlTBoundaryType Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

Operations:

6.3.1.3.31 PlTDataType Class

Overview:
class definition

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

Operations:

6.3.1.3.32 PlTGranIO Class

Overview:
forward definitions

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

Operations:

6.3.1.3.33 PlTimeScheduled Class

Overview:

This class is a specialization of the PlPge class and accounts for PGEs that are scheduled to process data for regular time period, this class accounts for all the PGEs that can be planned and scheduled in Release A. This class has a persistent representation in the PDPS as part of the PGE profile, and therefore has the methods to insert and retreive itself from the database.

Export Control: Public

Inheritance Relationships:

Inherits from PlPge

Attributes:

`myAcquisitionBoundary: PlBoundary`

myAcquisistionBoundary: describes the boundary on which the the period of data that is required within a Data Processing Request for this PGE (for more details see the PlBoundary class)

`myAcquisitionPeriod: PlPeriod`

myAcquisitionPeriod: descripes the period (duration) of the data that is requried within a Data Processing Request for this PGE (for more details see PlPeriod class)

Constructors and Destructor:

`PlTimeScheduled();`

Default constructor

`PlTimeScheduled(const RWCString& , EcUtStatus& );`

Constructor which retrieves a PGE from the database, given the PGE id.

```
virtual  ~PlTimeScheduled();
```

Destructor

Operations:

- Delete

```
virtual EcUtStatus Delete();
```

Database delete

- Exists

```
EcTBoolean Exists();
```

See if the record exists in the database

- GetAcquisitionBoundary

```
const PlBoundary& GetAcquisitionBoundary();
```

- GetAcquisitionPeriod

```
const PlPeriod& GetAcquisitionPeriod();
```

- Insert

```
virtual EcUtStatus Insert();
```

Database insert

- PlDisplayTimeScheduled

```
EcTVoid PlDisplayTimeScheduled();
```

Debug method to display the attributes

- PlGenerateDprs

```
virtual EcUtStatus PlGenerateDprs(PlTime& startTime, PlTime&
stopTime, RWCString& prodReqId, EcTInt& dprCount);
```

Provide a concrete implementation for the PGE method to explode Data Processing Requests from Production Requests

- Read

```
virtual EcUtStatus Read(const RWCString& );
```

Database read

- SetAcquisitionBoundary

  `EcTVoid SetAcquisitionBoundary(const PlBoundary& boundary);`

- SetAcquisitionPeriod

  `EcTVoid SetAcquisitionPeriod(const PlPeriod& period);`

- Update

  `virtual EcUtStatus Update();`

  Database update

6.3.1.3.34 PlTimeService Class

Overview:

This class simplifies the access to the EcTiTimeService time server for the PDPS for users of the Process Framework.

Export Control: Public

Inheritance Relationships:

Attributes:

`cheat: EcTInt`

If we're unable to instantiate the time service it could mean that we're running (testing) outside of DCE, therefore cheat and return the system time

`_instance: PlTimeService*`

following the Singleton design pattern there should be one instance of the PlTimeService class

`_timeService: EcTiTimeService*`

Just initialize one instance of the time service class

Constructors and Destructor:

`PlTimeService();`

The default constructor, protected as per singleton design pattern

`~PlTimeService();`

The destructor, protected as per singleton design pattern

Operations:

- Instance

  ```
  static PlTimeService* Instance();
  ```

  In order to create the class the following static member function has to be used as per Singleton design pattern

- now

  ```
  RWTime now();
  ```

  the reason d'etre for this class is to return current time

6.3.1.3.35 PlTime Class

Overview:

This class is a utility class for management of Time within the PDPS, it is derived from the standard Rogue Wave class but overrides it's behaviour to expect as a standard to be constructed from UTC, and to format to UTC, which is the standard that is used in specifying data acquisistion periods etc. The class also neatly presents utility methods required to format and construct time from other commonly used ECS formats.

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
PlTime(const RWDate& , EcTULongInt h, EcTULongInt m, EcTULongInt
s, const RWZone& );
```

Constructor that follows the RWTime standard

```
PlTime(const RWTime& );
```

Constructor from RWTime

```
PlTime(const tm* , const RWZone& );
```

Constructor from the struct tm defined in <time.h>

```
PlTime();
```

Default constructor

```
PlTime();
```

Constructor with a given number of seconds

```
~PlTime();
```

Destructor

Operations:

- asMMDDYYHHMM

```
RWCString asMMDDYYHHMM() const;
```

Formats the time as MMDDYYHHMM

- asString

```
RWCString asString(char , const RWZone& , const RWLocale& )
const;
```

Prints the time in UTC

- ClosestAbove

```
PlTime ClosestAbove(const PlBoundary& );
```

Closest time that is on a boundary after the given time

- ClosestBelow

```
PlTime ClosestBelow(const PlBoundary& );
```

Closest time that is on a boundary before the given time

- closest

```
PlTime closest(const PlTime& , const PlPeriod& );
```

Closest time that is a unit no of periods after the given time

- getDSSDateTime

```
EcTVoid getDSSDateTime(RWDate& , RWCString& ) const;
```

Converts the time into a format understood by SDSRV

- isValid

```
RWBoolean isValid();
```

Returns whether the time is a valid one

- operator +=

```
PlTime operator +=();
```

Overloaded + operator adds seconds to the time

- operator +=

  `PlTime operator +=(const PlPeriod& );`

  Overloaded + operator adds a specified period to the time

- operator -=

  `PlTime operator -=(const PlPeriod& );`

  Overloaded - operator subtracts a specified period from the time

- operator -=

  `PlTime operator -=();`

  Overloaded - operator subtracts seconds from the time

- operator =

  `PlTime operator =(const RWTime& );`

  Assignment operator

- operator =

  `PlTime operator =(const PlTime& );`

  Assignment operator

- operator RWDBValue

  `RWDBValue operator RWDBValue();`

  Converts to an RWDBValue

6.3.1.3.36 PlTPeriodType Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

Operations:

6.3.1.3.37 PlUserParametersCollection Class

Overview:

This class defines methods and operations to allow access to all of the User Parameters associated to a particular PGE

Export Control: Public

Inheritance Relationships:

Attributes:

`parms: RWTPtrSlist`

Collector for the User Parameters

Constructors and Destructor:

`PlUserParametersCollection();`

Default constructor

`PlUserParametersCollection(RWCString& , EcUtStatus& );`

Construct the collection of parameters for the specific PGE

`virtual ~PlUserParametersCollection();`

Destructor method

Operations:

- append

  `void append(PlUserParameters& );`

  Append a parameter to the collection

- apply

  `EcTVoid apply(EcTVoid (*) (class PlUserParameters*, class EcTVoid*) applyFun, EcTVoid* d);`

  Apply a function to all the members of the collection

- entries

`EcTInt entries() const;`

Return the number of entries within the collection

- GetOverridesFromPR

`EcUtStatus GetOverridesFromPR(PlProductionRequest& );`

Set the override values from the production request

- Insert

`EcUtStatus Insert();`

Insert the whole list of parameters into the database

- operator []

`PlUserParameters* operator [](EcTInt i);`

Return the parameter at the indicated index

- SetOverridesIntoPR

`EcUtStatus SetOverridesIntoPR(PlProductionRequest& );`

Get the override values from the production reques

6.3.1.3.38 PlUserParameters Class

Overview:

This class allows for manipulation of the run time parameters that may be defined for a PGE. These parameters are registered at SSIT, the overrides to a parameter are captured within Production Requests.

Export Control: Public

Inheritance Relationships:

Attributes:

`isOverridden: EcTBoolean`

usOverridden: flag to indicate that the default has been overriden

`myDefaultValue: RWCString`

myDefaultValue: The default value for the parameter (configured at SSIT)

`myDescription: RWCString`

myDescription: a comment that describes the usage of the parameter may describe valid ranges for the parameter

`myLogicalId: EcTInt`

myLogicalId: SDP toolkit identifier for the parameter

`myName: RWCString`

myName: A more descriptive name for the parameter

`myPgeId: RWCString`

myPgeId: reference to the PGE for which this parameter is defined

`myPROverride: RWCString`

myPROverride: a override to the default of the Parameter set up from the Production Request

Constructors and Destructor:

`PlUserParameters(const RWCString& pgeId, const EcTInt& logicalId, const RWCString& name, const RWCString& defaultValue, const RWCString& description);`

Constructor taking all persistent data items

`PlUserParameters(const RWCString& , EcTInt , EcUtStatus& );`

Constructor which retrieves the User Parameter from the database given the Parameter Name.

`PlUserParameters(const PlUserParameters& );`

Copy Constructor

`PlUserParameters();`

Default constructor

`virtual  ~PlUserParameters();`

Destructor

Operations:

- Delete

`EcUtStatus Delete();`

Delete class data from database

- Exists

`EcTBoolean Exists();`

Check to see if class data in database

- GetDefaultValue

```
const RWCString& GetDefaultValue();
```

- GetDescription

```
const RWCString& GetDescription();
```

- GetLogicalId

```
const EcTInt& GetLogicalId();
```

- GetName

```
const RWCString& GetName();
```

- GetOverride

```
EcTBoolean GetOverride();
```

- GetPgeId

```
const RWCString& GetPgeId();
```

- GetValue

```
const RWCString& GetValue();
```

- Insert

```
EcUtStatus Insert();
```

Insert class data into database

- operator ==

```
RWBoolean operator ==(const PlUserParameters& );
```

equality operator returns true if the objects have same PgeParameterId

- operator =

```
PlUserParameters& operator =(const PlUserParameters& );
```

assignment operator

- Read

  EcUtStatus Read(RWCString , EcTInt);

  Read class data from database

- SetDefaultValue

  EcTVoid SetDefaultValue(const RWCString&);

- SetDescription

  EcTVoid SetDescription(const RWCString&);

- SetLogicalId

  EcTVoid SetLogicalId(const EcTInt&);

- SetName

  EcTVoid SetName(const RWCString&);

- SetOverride

  EcTVoid SetOverride(const RWCString&);

- SetPgeId

  EcTVoid SetPgeId(const RWCString&);

- SetPgeParameterId

  EcTVoid SetPgeParameterId(const RWCString&);

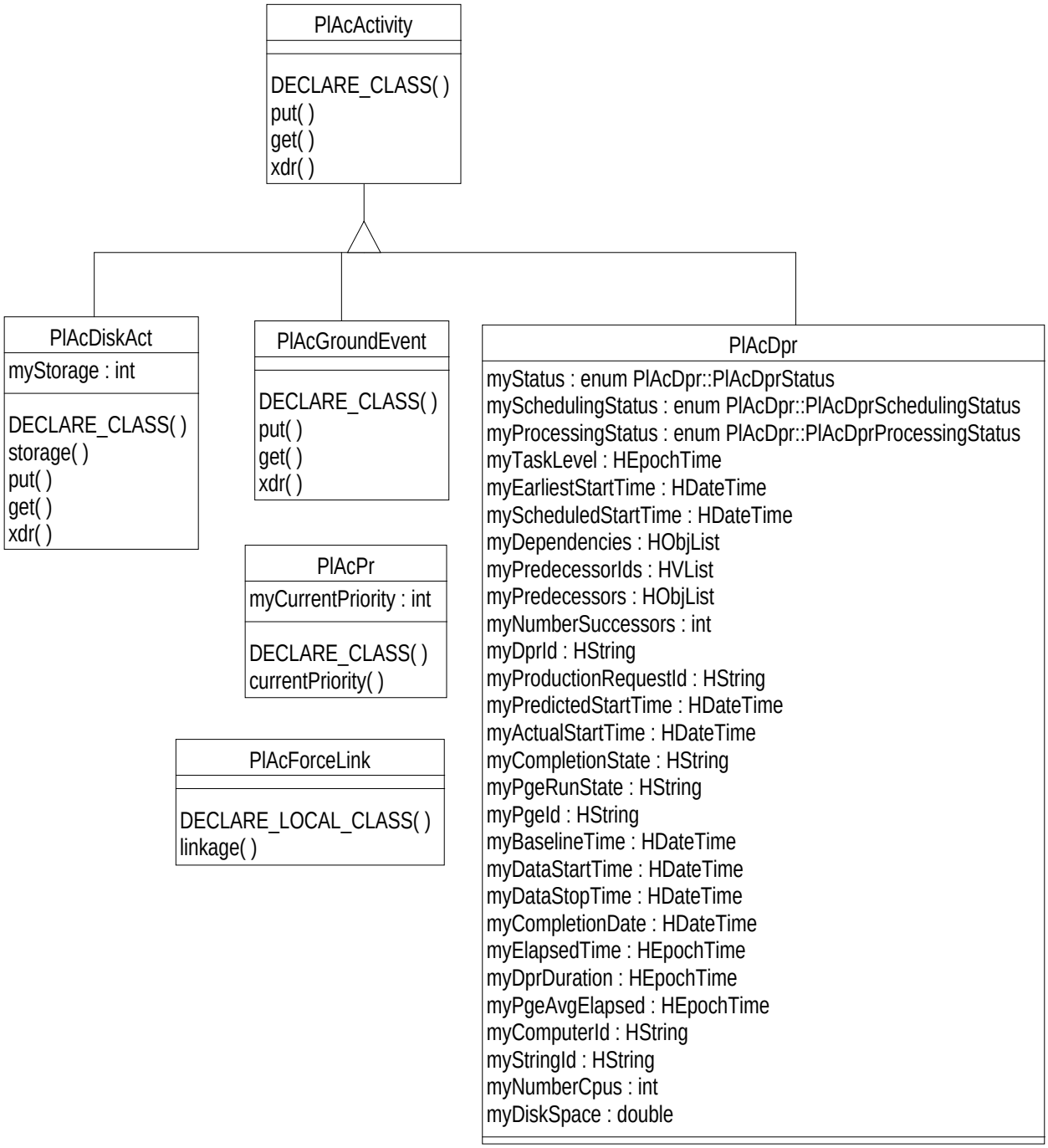
- Update

  EcUtStatus Update();

  Update class data into database

6.3.2 Planning_Workbench_Ac Class Category

6.3.2.1 Overview



6.3.2.2 Planning_Workbench_Ac Classes

6.3.2.3 PlAcActivity Class

Overview:

class PlAcActivity (-)

An instance of a PlAcActivity represents a request to change the configuration of a resource.

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
PlAcActivity();
```

Default constructor

```
PlAcActivity(const PlAcActivity& orig);
```

Copy constructor

```
virtual  ~PlAcActivity();
```

Destructor

Operations:

- DECLARE_CLASS

```
int DECLARE_CLASS(PlAcActivity );
```

class PlAcActivity (-)

An instance of a PlAcActivity represents a request to change the configuration of a resource.

- get

```
virtual int get(istream& istr);
```

Get myself from a stream

- operator =

```
PlAcActivity& operator =(const PlAcActivity& orig);
```

Assignment operator

- put

  `virtual int put(ostream& ostr) const;`

  Put myself onto a stream

- xdr

  `virtual int xdr(XDR* xdrs);`

  Get/put myself onto an XDR stream

6.3.2.3.1 PlAcDiskAct Class

Overview:

class PlAcDiskAct

Instances of PlAcDiskAct are activities that are directed specifically at the disk resource.

Export Control: Public

Inheritance Relationships:

Inherits from PlAcActivity

Attributes:

`myStorage: int`

My storage requirement

Constructors and Destructor:

`PlAcDiskAct();`

Default constructor

`PlAcDiskAct(const PlAcDiskAct& orig);`

Copy constructor

`virtual ~PlAcDiskAct();`

Destructor

Operations:

- DECLARE_CLASS

  `int DECLARE_CLASS(PlAcDiskAct );`

  class PlAcDiskAct

Instances of PlAcDiskAct are activities that are directed specifically at the disk resource.

- get

  `int get(istream& );`

  Get myself from a stream

- operator =

  `PlAcDiskAct& operator =(const PlAcDiskAct& orig);`

  Assignment operator

- put

  `int put(ostream& ) const;`

  Put myself onto a stream

- storage

  `virtual int storage() const;`

  Get my storage requirement

- storage

  `virtual int storage(int );`

  Set my storage requirement

- xdr

  `int xdr(XDR* );`

  Get/put myself onto an XDR stream

6.3.2.3.2 PlAcDpr Class

Overview:

class PlAcDpr (-)

Instances of PlAcDpr are activities which represent Data Processing Requests.

Export Control: Public

Inheritance Relationships:

Inherits from PlAcActivity

Attributes:

`myActualStartTime: HDateTime`

Actual start time

myBaselineTime: HDateTime

Plan baseline time

myCompletionDate: HDateTime

Completion Date

myCompletionState: HString

Completion state

myComputerId: HString

Computer ID

myDataStartTime: HDateTime

Data start time

myDataStopTime: HDateTime

Data stop time

myDependencies: HObjList

Data dependency granule IDs (strings)

myDiskSpace: double

Disk Space Required

myDprDuration: HEpochTime

DPR duration

myDprId: HString

Key identifier

myEarliestStartTime: HDateTime

Earliest start time. This should be calculated from the arrival times of external data.

myElapsedTime: HEpochTime

DPR elasped time

myNumberCpus: int

Number of CPUs

myNumberSuccessors: int

Number of successor DPRs.

myPgeAvgElapsed: HEpochTime

PGE average elapsed time in seconds

`myPgeId: HString`

PGE id

`myPgeRunState: HString`

PGE run state

`myPredecessorIds: HVList`

Predecessor DPR activitity IDs. The list contains unique activity IDs.

`myPredecessors: HObjList`

Predecessor DPR activitities. The list should contains unique pointers.

`myPredictedStartTime: HDateTime`

Predicted start date/time. This should be set when the plan is rolled over to processing.

`myProcessingStatus: enum PlAcDpr::PlAcDprProcessingStatus`

My processing status

`myProductionRequestId: HString`

Parent production request identifier

`myScheduledStartTime: HDateTime`

Earliest scheduled time. This should be set by the scheduling algorithm for the currently selected plan.

`mySchedulingStatus: enum PlAcDpr::PlAcDprSchedulingStatus`

My scheduling status

`myStatus: enum PlAcDpr::PlAcDprStatus`

My overall status

`myStringId: HString`

String ID. This will be used if a computer ID is not provided.

`myTaskLevel: HEpochTime`

My task level: The level of a node in a task graph is defined as the length in time of the longest path from the node to the exit node.

Constructors and Destructor:

`PlAcDpr();`

Default constructor

`PlAcDpr(const PlAcDpr& orig);`

Copy constructor

```
virtual  ~PlAcDpr();
```

Destructor

Operations:

- actualStartTime

```
virtual const HDateTime& actualStartTime() const;
```

Get my actual start time

- actualStartTime

```
virtual int actualStartTime(const HDateTime& aActualStart);
```

Setmy actual start time

- addDependency

```
virtual int addDependency(const HString& granuleId);
```

Add a granule ID on which I depend to list of granules

- addPredecessor

```
virtual int addPredecessor(const HString& predecessorDprName);
```

Add DPR to list of predecessors by name

- addPredecessor

```
virtual int addPredecessor(PlAcDpr* predecessorDpr);
```

Add DPR to list of predecessors

- adjustEarliestStartTime

```
virtual int adjustEarliestStartTime(const HDateTime&
aStartTime);
```

Adjust my earliest start time. Only make an adjustment if the passed in start time is later than the current earliest start time.

- adjustScheduledStartTime

```
virtual int adjustScheduledStartTime(const HDateTime&
aStartTime);
```

Adjust my scheduled start time. Only make an adjustment if the passed in start time is later than the current scheduled start time.

- baselineTime

```
virtual int baselineTime(const HDateTime& aBaselineTime);
```

Set my baseline time

- baselineTime

```
virtual const HDateTime& baselineTime() const;
```

Get my baseline time

- cleanup

```
virtual void cleanup();
```

Delete member data

- compare

```
virtual int compare(const PlAcDpr& dprAct) const;
```

Compare me with another PlAcDpr

- compare

```
virtual int compare(const HObject& obj) const;
```

Compare me with an HObject

- compare

```
virtual int compare(const RActivity& ract) const;
```

Compare me with an RActivity

- completionDate

```
virtual const HDateTime& completionDate() const;
```

Get my completion date

- completionDate

```
virtual int completionDate(const HDateTime& aCompletionDate);
```

Set my completion date

- completionState

```
virtual int completionState(const char* aCompletionState);
```

Set my completion state

- completionState

```
virtual const HString& completionState() const;
```

Get my completion state

- computerId

```
virtual int computerId(const char* aComputerId);
```

Set my Computer ID

- computerId

```
virtual const HString& computerId() const;
```

Get my Computer ID

- copyOver

```
virtual int copyOver(const PlAcDpr& dpr);
```

Copy internals of source PlAcDpr

- copyOver

```
virtual int copyOver(const RSimpleAct& act);
```

Copy internals of source RSimpleAct

- dataStartTime

```
virtual int dataStartTime(const HDateTime& aDataStart);
```

Set my data start time

- dataStartTime

```
virtual const HDateTime& dataStartTime() const;
```

Get my data start time

- dataStopTime

```
virtual const HDateTime& dataStopTime() const;
```

Get my data stop time

- dataStopTime

```
virtual int dataStopTime(const HDateTime& aDataStop);
```

Set my data stop time

- DECLARE_CLASS

```
int DECLARE_CLASS(PlAcDpr );
```

class PlAcDpr (-)

Instances of PlAcDpr are activities which represent Data Processing Requests.

- dependencies

```
virtual HObjIter* dependencies();
```

Get an iterator over my list of input granules

- diskSpace

  ```
  virtual int diskSpace(double amount);
  ```

  Set my disk space requirement

- diskSpace

  ```
  virtual double diskSpace() const;
  ```

  Get my disk space requirement

- dprDuration

  ```
  virtual int dprDuration(double duration);
  ```

  Set my DPR duration

- dprDuration

  ```
  virtual double dprDuration() const;
  ```

  Get my DPR duration

- dprId

  ```
  virtual int dprId(const char* aDprId);
  ```

  Set my DPR identifier

- dprId

  ```
  virtual const HString& dprId() const;
  ```

  Get my DPR identifier

- earliestStartTime

  ```
  virtual int earliestStartTime(const HDateTime& aStartTime);
  ```

  Set my earliest start time. This should be calculated from the predicted arrival of external data.

- earliestStartTime

  ```
  virtual const HDateTime& earliestStartTime() const;
  ```

  Get my earliest start time.

- elapsedTime

  ```
  virtual double elapsedTime() const;
  ```

  Get my DPR elapsed time

- elapsedTime

  ```
  virtual int elapsedTime(double aElapsedTime);
  ```

  Set my DPR elapsed time

- get

  ```
  virtual int get(istream& istr);
  ```

  Get myself from a stream

- numberCpus

  ```
  virtual int numberCpus(int number);
  ```

  Set my required number of CPUs

- numberCpus

  ```
  virtual int numberCpus() const;
  ```

  Get my required number of CPUs

- numberSuccessors

  ```
  virtual int numberSuccessors(int nSuccessors);
  ```

  Set my number of successor DPRs

- numberSuccessors

  ```
  virtual int numberSuccessors() const;
  ```

  Get my number of successor DPRs

- operator =

  ```
  PlAcDpr& operator =(const PlAcDpr& orig);
  ```

  Assignment operator

- pgeAvgElapsed

  ```
  virtual int pgeAvgElapsed(double interval);
  ```

  Set my average PGE elapsed time

- pgeAvgElapsed

  ```
  virtual double pgeAvgElapsed() const;
  ```

  Get my average PGE elapsed time

- pgeId

  ```
  virtual const HString& pgeId() const;
  ```

  Get my PGE identifier

- pgeId

  ```
  virtual int pgeId(const char* aPgeId);
  ```

  Set my PGE identifier

- pgeRunState

  ```
  virtual const HString& pgeRunState() const;
  ```

  Get my PGE run state

- pgeRunState

  ```
  virtual int pgeRunState(const char* aPgeRunState);
  ```

  Set my PGE run state

- predecessors

  ```
  virtual HObjIter* predecessors();
  ```

  Get an iterator over my list of predecessor DPR activities.

- predictedStartTime

  ```
  virtual int predictedStartTime(const HDateTime&
  aPredictedStartTime);
  ```

  Set my predicted start time. This should be set when the plan is rolled over to processing.

- predictedStartTime

  ```
  virtual const HDateTime& predictedStartTime() const;
  ```

  Get my predicted start time

- processingStatusAsString

  ```
  virtual const char* processingStatusAsString() const;
  ```

  Return my processing status as a character string

- processingStatus

  ```
  virtual int processingStatus(enum
  PlAcDpr::PlAcDprProcessingStatus );
  ```

  Set my processing status

- processingStatus

  ```
  virtual int processingStatus(const char* aProcessingStatus);
  ```

  Set my status with a character string representing one of the possible enumerated vaules: inactive, on_queue, started, success, fail, canceled, qa_failure Return TRUE/FALSE upon success or failure.

- processingStatus

  ```
  virtual enum PlAcDpr::PlAcDprProcessingStatus processingStatus()
  const;
  ```

Get my processing status

- productionRequestId

```
virtual int productionRequestId(const char* aPrId);
```

Set my parent production request identifier

- productionRequestId

```
virtual const HString& productionRequestId() const;
```

Get my parent production request identifier

- put

```
virtual int put(ostream& ostr) const;
```

Put myself onto a stream

- resolvePredecessorIds

```
virtual void resolvePredecessorIds();
```

Try to resolve any predecessor DPR ids to DPRs from the activity pool

- scheduledStartTime

```
virtual int scheduledStartTime(const HDateTime& aStartTime);
```

Set my scheduled start time. This attribute is a place holder used by the scheduling algorithm for the currently selected plan.

- scheduledStartTime

```
virtual const HDateTime& scheduledStartTime() const;
```

Get my scheduled start time

- schedulingStatusAsString

```
virtual const char* schedulingStatusAsString() const;
```

Return my scheduling status as a character string

- schedulingStatus

```
virtual enum PlAcDpr::PlAcDprSchedulingStatus schedulingStatus()
const;
```

Get my scheduling status

- schedulingStatus

```
virtual int schedulingStatus(enum
PlAcDpr::PlAcDprSchedulingStatus );
```

Set my scheduling status

- schedulingStatus

  ```
  virtual int schedulingStatus(const char* aSchedulingStatus);
  ```

  Set my status with a character string of SCHEDUABLE or UNSCHEDUABLE. Return TRUE/FALSE upon success or failure.

- statusAsString

  ```
  virtual const char* statusAsString() const;
  ```

  Return my status as a character string.

- status

  ```
  virtual enum PlAcDpr::PlAcDprStatus status() const;
  ```

  Get my overall status

- status

  ```
  virtual int status(enum PlAcDpr::PlAcDprStatus );
  ```

  Set my overall status

- status

  ```
  virtual int status(const char* aStatus);
  ```

  Set my status with a character string of VALID or INVALID. Return TRUE/FALSE upon success or failure.

- stringId

  ```
  virtual int stringId(const char* aStringId);
  ```

  Set my String ID

- stringId

  ```
  virtual const HString& stringId() const;
  ```

  Get my String ID

- taskLevel

  ```
  virtual int taskLevel(const HEpochTime& aTaskLevel);
  ```

  Set my task level

- taskLevel

  ```
  virtual const HEpochTime& taskLevel() const;
  ```

  Get my task level

- xdr

```
virtual int xdr(XDR* xdrs);
```

Get/put myself onto an XDR stream

6.3.2.3.3 PlAcForceLink Class

Overview:

class PlAcForceLink (container class for dummy class instantiation)

Instances of PlAcForceLink are dummy classes, used to trick the linker into linking classes which are usually not known at compile time. These are objects which might be dynamically loaded off of a stream.

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
PlAcForceLink();
```

Default constructor

```
PlAcForceLink(const PlAcForceLink& orig);
```

Copy constructor

```
virtual  ~PlAcForceLink();
```

Destructor

Operations:

- DECLARE_LOCAL_CLASS

  ```
  int DECLARE_LOCAL_CLASS(PlAcForceLink );
  ```

  class PlAcForceLink (container class for dummy class instantiation)

  Instances of PlAcForceLink are dummy classes, used to trick the linker into linking classes which are usually not known at compile time. These are objects which might be dynamically loaded off of a stream.

- linkage

  ```
  virtual void linkage();
  ```

Protocol for forcing linkage of class objects not known at run time. Create dummy objects of all the types which you desire to force into the link. This behavior is never actually called at run time, just here to force the compiler to link classes in.

- operator =

  ```
  PlAcForceLink& operator =(const PlAcForceLink& orig);
  ```

  Assignment operator

6.3.2.3.4 PlAcGroundEvent Class

Overview:

class PlAcGroundEvent (-)

Instances of PlAcGroundEvent are activities which model ground events on resources.

Export Control: Public

Inheritance Relationships:

Inherits from PlAcActivity

Attributes:

Constructors and Destructor:

```
PlAcGroundEvent();
```

Default constructor

```
PlAcGroundEvent(const PlAcGroundEvent& orig);
```

Copy constructor

```
virtual  ~PlAcGroundEvent();
```

Destructor

Operations:

- DECLARE_CLASS

  ```
  int DECLARE_CLASS(PlAcGroundEvent );
  ```

  class PlAcGroundEvent (-)

  Instances of PlAcGroundEvent are activities which model ground events on resources.

- get

```
virtual int get(istream& istr);
```

Get myself from a stream

- operator =

```
PlAcGroundEvent& operator =(const PlAcGroundEvent& orig);
```

Assignment operator

- put

```
virtual int put(ostream& ostr) const;
```

Put myself onto a stream

- xdr

```
virtual int xdr(XDR* xdrs);
```

Get/put myself onto an XDR stream

6.3.2.3.5 PlAcPr Class

Overview:

class PlAcPr (Production Request Activity)

Instances of PlAcPr are complex activities which represent Production Requests. PlAcPrs maintain a list of associated DPRs (PlAcDpr)

Export Control: Public

Inheritance Relationships:

Attributes:

```
myCurrentPriority: int
```

My current priority

Constructors and Destructor:

```
PlAcPr();
```

Default constructor

```
PlAcPr(const PlAcPr& orig);
```

Copy constructor

```
virtual  ~PlAcPr();
```

Destructor

Operations:

- currentPriority

  ```
  virtual int currentPriority(int val);
  ```

  Set my current priority

- currentPriority

  ```
  virtual int currentPriority() const;
  ```

  Get my current priority

- DECLARE_CLASS

  ```
  int DECLARE_CLASS(PlAcPr );
  ```

  class PlAcPr (Production Request Activity)

  Instances of PlAcPr are complex activities which represent Production Requests. PlAcPrs maintain a list of associated DPRs (PlAcDpr)

- get

  ```
  virtual int get(istream& istr);
  ```

  Get myself from a stream

- operator =

  ```
  PlAcPr& operator =(const PlAcPr& orig);
  ```

  Assignment operator

- put

  ```
  virtual int put(ostream& ostr) const;
  ```

  Put myself onto a stream

- xdr

  ```
  virtual int xdr(XDR* xdrs);
  ```

  Get/put myself onto an XDR stream

6.3.3 Planning_Workbench_Cl Class Category

6.3.3.1 Overview

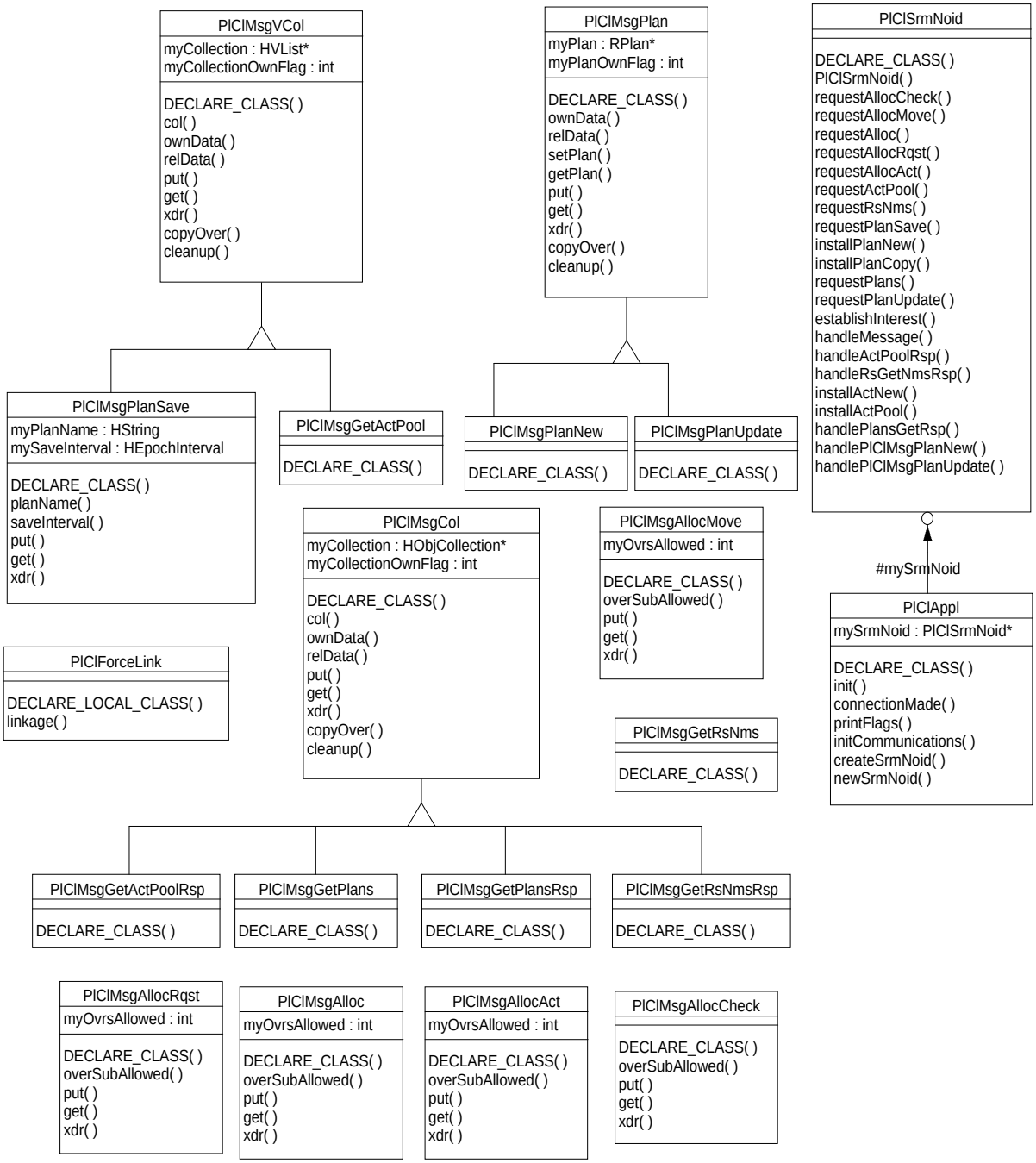


Figure 6.3.3.1-1 Planning_Workbench_Cl

6.3.3.2 Planning_Workbench_Cl Classes

6.3.3.3 PlClAppl Class

Overview:

PlClAppl (-)

A PlClAppl is a resource model client application class. It is responsible for setting up communication to a resource model and initializing the entire process. PlClAppl contains an SRM noid for sending messages to the srm.

Export Control: Public

Inheritance Relationships:

Inherits from PlMiAppl

Attributes:

```
mySrmNoid: PlClSrmNoid*
```

My SRM noid. It is owned by the agent and is therefore cleaned up there.

Constructors and Destructor:

```
PlClAppl();
```

Default Constructor

```
PlClAppl(const PlClAppl& orig);
```

Copy Constructor

```
virtual  ~PlClAppl();
```

Destructor

Operations:

- connectionMade

```
virtual int connectionMade();
```

This member is called when the srm noid has made a connection to the resource model. Derived classes should override for any behavior that is to occur after a connection is made to the resource model.

- createSrmNoid

```
virtual int createSrmNoid();
```

Create the SRM noid. This should be overridden in the derived class.

- DECLARE_CLASS

  ```
  int DECLARE_CLASS(PlClAppl );
  ```

  PlClAppl (-)

  A PlClAppl is a resource model client application class. It is responsible for setting up communication to a resource model and initializing the entire process. PlClAppl contains an SRM noid for sending messages to the srm.

- initCommunications

  ```
  virtual int initCommunications();
  ```

  Initializes the SRM noid, and connects to the SRM

- init

  ```
  virtual int init();
  ```

  This function provides protocol for initialization done here.

- newSrmNoid

  ```
  virtual PlClSrmNoid* newSrmNoid();
  ```

  Instantiate an SRM noid. This should be overridden in derived classes to instantiate a derived type of PlClSrmNoid.

- operator =

  ```
  PlClAppl& operator =(const PlClAppl& orig);
  ```

  Assignment Operator

- printFlags

  ```
  virtual void printFlags();
  ```

  Dump a description of what flags are availible and required to standard output.

6.3.3.3.1 PlClForceLink Class

Overview:

class PlClForceLink (force link for ECS resource model classes)

Instances of PlClForceLink are used to force linkage of resource model message classes which may not be present at link time.

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
PlClForceLink();
```

Default constructor

```
PlClForceLink(const PlClForceLink& orig);
```

Copy constructor

```
virtual  ~PlClForceLink();
```

Destructor

Operations:

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlClForceLink );
```

class PlClForceLink (force link for ECS resource model classes)

Instances of PlClForceLink are used to force linkage of resource model message classes which may not be present at link time.

- linkage

```
void linkage();
```

Protocol for forcing linkage of class objects not known at run time. Create dummy objects of all the types which you desire to force into the link. This behavior is never actually called at run time, just here to force the compiler to link classes in.

- operator =

```
PlClForceLink& operator =(const PlClForceLink& orig);
```

Assignment operator

6.3.3.3.2 PlClMsgAllocAct Class

Overview:

class PlClMsgAllocAct (allocation message with activity, oversubscription).

PlClMsgAllocAct is a request to allocate a specific resource to a specific activity on a specific plan over a specific interval. Constraints will be checked to determine if the allocation is feasible. The activity will be added to the RActPool. Oversubscription may be specified.

Export Control: Public

Inheritance Relationships:

Attributes:

`myOvrsAllowed: int`

Specifies whether or not oversubscription is allowed

Constructors and Destructor:

`PlClMsgAllocAct(const PlClMsgAllocAct& orig);`

Copy constructor.

`PlClMsgAllocAct(const char* planName, const HEpochInterval& intvl, int rsId, RActivity* act, int ovrsAllowed);`

Constructor taking member data. I do not assume ownership of the provided RActivity.

`PlClMsgAllocAct();`

Default constructor

`virtual  ~PlClMsgAllocAct();`

Destructor. I will delete any data that I own.

Operations:

- DECLARE_CLASS

  `int DECLARE_CLASS(PlClMsgAllocAct );`

  class PlClMsgAllocAct (allocation message with activity, oversubscription).

  PlClMsgAllocAct is a request to allocate a specific resource to a specific activity on a specific plan over a specific interval. Constraints will be checked to determine if the allocation is feasible. The activity will be added to the RActPool. Oversubscription may be specified.

- get

  `virtual int get(istream& istr);`

  Get myself from a stream

- operator =

  `PlClMsgAllocAct& operator =(const PlClMsgAllocAct& orig);`

  Assignment operator. The data in the source message is cloned and owned.

- overSubAllowed

  `virtual int overSubAllowed(int yesNo);`

Set whether oversubscription is allowed.

- overSubAllowed

```
virtual int overSubAllowed() const;
```

Get whether oversubscription is allowed.

- put

```
virtual int put(ostream& ostr) const;
```

Put myself onto a stream

- xdr

```
virtual int xdr(XDR* xdrs);
```

Get/put myself onto an XDR stream

6.3.3.3.3 PlClMsgAllocCheck Class

Overview:

Class PlClMsgAllocCheck (ecs check allocation message)

PlClMsgAllocCheck is a SMsgAllocCheck request to check a specific resource to see if a specific activity on a specific plan over a specific interval can be allocated on that resource. This is determined by constraint-checking. The message also holds state to determine if oversubscription is allowed.

Export Control: Public

Inheritance Relationships:

Attributes:

```
myOvrsAllowed: int
```

Specifies whether or not oversubscription is allowed

Constructors and Destructor:

```
PlClMsgAllocCheck(const PlClMsgAllocCheck& orig);
```

Copy constructor.

```
PlClMsgAllocCheck(const char* planName, const HEpochInterval&
intvl, int rsId, int actId, int ovrsAllowed);
```

Constructor taking member data.

```
PlClMsgAllocCheck();
```

Default constructor.

```
virtual  ~PlClMsgAllocCheck();
```

Destructor.

Operations:

- DECLARE_CLASS

```
int DECLARE_CLASS(PlClMsgAllocCheck );
```

Class PlClMsgAllocCheck (ecs check allocation message)

PlClMsgAllocCheck is a SMsgAllocCheck request to check a specific resource to see if a specific activity on a specific plan over a specific interval can be allocated on that resource. This is determined by constraint-checking. The message also holds state to determine if oversubscription is allowed.

- get

```
virtual int get(istream& istr);
```

Get myself from a stream

- operator =

```
PlClMsgAllocCheck& operator =(const PlClMsgAllocCheck& orig);
```

Assignment operator

- overSubAllowed

```
virtual int overSubAllowed(int yesNo);
```

Set whether oversubscription is allowed or not

- overSubAllowed

```
virtual int overSubAllowed() const;
```

Get whether oversubscription is allowed or not

- put

```
virtual int put(ostream& ostr) const;
```

Put myself onto a stream

- xdr

```
virtual int xdr(XDR* xdrs);
```

Get/put myself onto an XDR stream

6.3.3.3.4 PlClMsgAllocMove Class

Overview:

Class PlClMsgAllocMove (force allocation message)

PlClMsgAllocMove is a request to allocate a specific resource to a specific activity on a specific plan over a specific interval. The allocation is forced - no constraints are checked. Information is added to specify if oversubscription is permitted or not.

Export Control: Public

Inheritance Relationships:

Attributes:

`myOvrsAllowed: int`

Specifies whether or not oversubscription is allowed

Constructors and Destructor:

`PlClMsgAllocMove(const PlClMsgAllocMove& orig);`

Copy constructor.

`PlClMsgAllocMove(int actId, int sourceRsId, int destRsId, const char* sourcePlanName, const char* destPlanName, const HEpochInterval& oldIntvl, const HEpochInterval& newIntvl, int ovrsAllowed, int impactFlag);`

Constructor taking member data.

`PlClMsgAllocMove();`

Default constructor.

`virtual ~PlClMsgAllocMove();`

Destructor.

Operations:

- DECLARE_CLASS

`int DECLARE_CLASS(PlClMsgAllocMove );`

Class PlClMsgAllocMove (force allocation message)

PlClMsgAllocMove is a request to allocate a specific resource to a specific activity on a specific plan over a specific interval. The allocation is forced - no constraints are checked. Information is added to specify if oversubscription is permitted or not.

- get

  ```
  virtual int get(istream& istr);
  ```

  Get myself from a stream

- operator =

  ```
  PlClMsgAllocMove& operator =(const PlClMsgAllocMove& orig);
  ```

  Assignment operator.

- overSubAllowed

  ```
  virtual int overSubAllowed(int yesNo);
  ```

  Set whether oversubscription is allowed or not

- overSubAllowed

  ```
  virtual int overSubAllowed() const;
  ```

  Get whether oversubscription is allowed or not

- put

  ```
  virtual int put(ostream& ostr) const;
  ```

  Put myself onto a stream

- xdr

  ```
  virtual int xdr(XDR* xdrs);
  ```

  Get/put myself onto an XDR stream

6.3.3.3.5 PlClMsgAllocRqst Class

Overview:

class PlClMsgAllocRqst (allocation message with activity, oversubscription).

PlClMsgAllocRqst is a request to allocate a specific resource to a specific activity on a specific plan over a specific interval. Constraints will be checked to determine if the allocation is feasible. The activity will be added to the RActPool. Oversubscription may be specified.

Export Control: Public

Inheritance Relationships:

Attributes:

`myOvrsAllowed: int`

Specifies whether or not oversubscription is allowed

Constructors and Destructor:

`PlClMsgAllocRqst(const PlClMsgAllocRqst& orig);`

Copy constructor.

`PlClMsgAllocRqst(const char* planName, HObjCollection* reqsToSch, HObjCollection* schRqsts, HObjCollection* notSchRqsts, int ovrsAllowed);`

Constructor taking member data.

`PlClMsgAllocRqst();`

Default constructor

`virtual  ~PlClMsgAllocRqst();`

Destructor. I will delete any data that I own.

Operations:

- DECLARE_CLASS

  `int DECLARE_CLASS(PlClMsgAllocRqst );`

  class PlClMsgAllocRqst (allocation message with activity, oversubscription).

  PlClMsgAllocRqst is a request to allocate a specific resource to a specific activity on a specific plan over a specific interval. Constraints will be checked to determine if the allocation is feasible. The activity will be added to the RActPool. Oversubscription may be specified.

- get

  `virtual int get(istream& istr);`

  Get myself from a stream

- operator =

  `PlClMsgAllocRqst& operator =(const PlClMsgAllocRqst& orig);`

  Assignment operator. The data in the source message is cloned and owned.

- overSubAllowed

  `virtual int overSubAllowed(int yesNo);`

  Set whether oversubscription is allowed

- overSubAllowed

  ```
  virtual int overSubAllowed() const;
  ```

  Get whether oversubscription is allowed

- put

  ```
  virtual int put(ostream& ostr) const;
  ```

  Put myself onto a stream

- xdr

  ```
  virtual int xdr(XDR* xdrs);
  ```

  Get/put myself onto an XDR stream

6.3.3.3.6 PlClMsgAlloc Class

Overview:

Class PlClMsgAlloc (allocation message).

PlClMsgAlloc is a request to allocate a specific resource to a specific activity on a specific plan over a specific interval. Constraints will be checked to determine if the allocation is feasible. The user may specify whether oversubscription is permitted or not.

Export Control: Public

Inheritance Relationships:

Attributes:

```
myOvrsAllowed: int
```

Specifies whether or not oversubscription is allowed

Constructors and Destructor:

```
PlClMsgAlloc(const PlClMsgAlloc& orig);
```

Copy constructor.

```
PlClMsgAlloc(const char* planName, const HEpochInterval& intvl,
int rsId, int actId, int ovrsAllowed);
```

Constructor taking member data.

```
PlClMsgAlloc();
```

Default constructor

```
virtual  ~PlClMsgAlloc();
```

Destructor.

Operations:

- DECLARE_CLASS

```
int DECLARE_CLASS(PlClMsgAlloc );
```

Class PlClMsgAlloc (allocation message).

PlClMsgAlloc is a request to allocate a specific resource to a specific activity on a specific plan over a specific interval. Constraints will be checked to determine if the allocation is feasible. The user may specify whether oversubscription is permitted or not.

- get

```
virtual int get(istream& istr);
```

Get myself from a stream

- operator =

```
PlClMsgAlloc& operator =(const PlClMsgAlloc& orig);
```

Assignment operator.

- overSubAllowed

```
virtual int overSubAllowed(int yesNo);
```

Set whether oversubscription is allowed or not

- overSubAllowed

```
virtual int overSubAllowed() const;
```

Get whether oversubscription is allowed or not

- put

```
virtual int put(ostream& ostr) const;
```

Put myself onto a stream

- xdr

```
virtual int xdr(XDR* xdrs);
```

Get/put myself onto an XDR stream

6.3.3.3.7 PlClMsgCol Class

Overview:

class PlClMsgCol (collection message with status)

Instances of PlClMsgCol are a message that carries collections from one agent to another. NOTE: copy and assignment will result in a DEEP COPY of the messages carried collection I.E. - the message copied to or assigned to will own its own copy of the data.

Export Control: Public

Inheritance Relationships:

Attributes:

`myCollectionOwnFlag: int`

Specifies whether or not I own myCollection

`myCollection: HObjCollection*`

The collection that I carry

Constructors and Destructor:

`PlClMsgCol(const PlClMsgCol& orig);`

Copy constructor

`PlClMsgCol(HObjCollection* col);`

Constructor taking a pointer to a collection. I do not assume ownership of the passed in collection.

`PlClMsgCol();`

Default constructor

`virtual  ~PlClMsgCol();`

Destructor

Operations:

- cleanup

  `void cleanup();`

  Clean up any data members

- col

  `virtual HObjCollection* col();`

  Get my collection.

- col

```
virtual int col(HObjCollection* col);
```

Set my collection. I do not assume ownership on the set. I will relinquish ownership on the get.

- copyOver

```
int copyOver(const PlClMsgCol& orig);
```

Copy over data from another PlClMsgCol

- DECLARE_CLASS

```
int DECLARE_CLASS(PlClMsgCol );
```

class PlClMsgCol (collection message with status)

Instances of PlClMsgCol are a message that carries collections from one agent to another. NOTE: copy and assignment will result in a DEEP COPY of the messages carried collection I.E. - the message copied to or assigned to will own its own copy of the data.

- get

```
virtual int get(istream& istr);
```

Get myself from a stream

- operator =

```
PlClMsgCol& operator =(const PlClMsgCol& orig);
```

Assignment operator

- ownData

```
virtual int ownData();
```

Ask me to own my data

- put

```
virtual int put(ostream& ostr) const;
```

Put myself onto a stream

- relData

```
virtual int relData();
```

Ask me to relinquish ownership of my data

- xdr

```
virtual int xdr(XDR* xdrs);
```

Get/put myself onto an XDR stream

6.3.3.3.8 PlClMsgGetActPoolRsp Class

Overview:

class PlClMsgGetActPoolRsp ()

Comprises a response to a request for activities, and carries a copy of all or part of the resource model's activity pool.

Export Control: Public

Inheritance Relationships:

Inherits from PlClMsgCol

Attributes:

Constructors and Destructor:

```
PlClMsgGetActPoolRsp(const PlClMsgGetActPoolRsp& orig);
```

Copy constructor

```
PlClMsgGetActPoolRsp(HObjCollection* actPool);
```

Alternate constructor. I will not assume ownership of the given collection.

```
PlClMsgGetActPoolRsp();
```

Default constructor

```
virtual  ~PlClMsgGetActPoolRsp();
```

Destructor

Operations:

- DECLARE_CLASS

```
int DECLARE_CLASS(PlClMsgGetActPoolRsp );
```

class PlClMsgGetActPoolRsp ()

Comprises a response to a request for activities, and carries a copy of all or part of the resource model's activity pool.

- operator =

```
PlClMsgGetActPoolRsp& operator =(const PlClMsgGetActPoolRsp&
orig);
```

Assignment operator

6.3.3.3.9 PlClMsgGetActPool Class

Overview:

class PlClMsgGetActPool (-)

Instances of PlClMsgGetActPool are messages requesting activities from the resource model. The carried HVList should be a list of integer activity ID's. The corresponding activity to each ID will be returned in the response. If no list is supplied, all activities known to the resource model will be returned.

Export Control: Public

Inheritance Relationships:

Inherits from PlClMsgVCol

Attributes:

Constructors and Destructor:

```
PlClMsgGetActPool(const PlClMsgGetActPool& orig);
```

Copy constructor

```
PlClMsgGetActPool(HVList* actIdList);
```

Constructor taking member data

```
PlClMsgGetActPool();
```

Default constructor

```
virtual  ~PlClMsgGetActPool();
```

Destructor

Operations:

- DECLARE_CLASS

  ```
  int DECLARE_CLASS(PlClMsgGetActPool );
  ```

  class PlClMsgGetActPool (-)

  Instances of PlClMsgGetActPool are messages requesting activities from the resource model. The carried HVList should be a list of integer activity ID's. The corresponding activity to each ID will be returned in the response. If no list is supplied, all activities known to the resource model will be returned.

- operator =

```
PlClMsgGetActPool& operator =(const PlClMsgGetActPool& orig);
```

Assignment operator

6.3.3.3.10 PlClMsgGetPlansRsp Class

Overview:

class PlClMsgGetPlansRsp (Plans message)

Instances of PlClMsgGetPlansRsp are a response to a request sent by an PlClMsgGetPlans.

PlClMsgGetPlansRsp carries a collection of plans corresponding to the plan names carried in the PlClMsgGetPlans. If no names were specified, then the response contains all plans in the resource model.

Export Control: Public

Inheritance Relationships:

Inherits from PlClMsgCol

Attributes:

Constructors and Destructor:

```
PlClMsgGetPlansRsp(const PlClMsgGetPlansRsp& orig);
```

Copy constructor

```
PlClMsgGetPlansRsp(HObjCollection* planList);
```

Constructor taking a collection of plans I do not assume ownership of the passed in collection.

```
PlClMsgGetPlansRsp();
```

Default constructor

```
virtual  ~PlClMsgGetPlansRsp();
```

Destructor

Operations:

- DECLARE_CLASS

```
int DECLARE_CLASS(PlClMsgGetPlansRsp );
```

class PlClMsgGetPlansRsp (Plans message)

Instances of PlClMsgGetPlansRsp are a response to a request sent by an PlClMsgGetPlans.

PlClMsgGetPlansRsp carries a collection of plans corresponding to the plan names carried in the PlClMsgGetPlans. If no names were specified, then the response contains all plans in the resource model.

- operator =

  `PlClMsgGetPlansRsp& operator =(const PlClMsgGetPlansRsp& orig);`

  Assignment operator

6.3.3.3.11 PlClMsgGetPlans Class

Overview:

class PlClMsgGetPlans (Get plans message)

Instances of PlClMsgGetPlans are a request to get a subset of plans from the resource model. All plans with names matching those given in the collection will be returned. If no names are specified, then all plans will be returned.

Export Control: Public

Inheritance Relationships:

Inherits from PlClMsgCol

Attributes:

Constructors and Destructor:

`PlClMsgGetPlans(const PlClMsgGetPlans& orig);`

Copy constructor

`PlClMsgGetPlans(HObjCollection* planNames);`

Constructor taking a pointer to a collection of plan names. I do not assume ownership of the passed in collection.

`PlClMsgGetPlans();`

Default constructor

`virtual ~PlClMsgGetPlans();`

destructor

Operations:

- DECLARE_CLASS

```
int DECLARE_CLASS(PlClMsgGetPlans );
```

class PlClMsgGetPlans (Get plans message)

Instances of PlClMsgGetPlans are a request to get a subset of plans from the resource model. All plans with names matching those given in the collection will be returned. If no names are specified, then all plans will be returned.

- operator =

```
PlClMsgGetPlans& operator =(const PlClMsgGetPlans& orig);
```

Assignment operator

6.3.3.3.12 PlClMsgGetRsNmsRsp Class

Overview:

Class PlClMsgGetRsNmsRsp (resource names message)

PlClMsgGetRsNmsRsp contains the names of all resources known by the resource model. It is sent in response to a request for resource names (PlClMsgRsNms).

Export Control: Public

Inheritance Relationships:

Inherits from PlClMsgCol

Attributes:

Constructors and Destructor:

```
PlClMsgGetRsNmsRsp(const PlClMsgGetRsNmsRsp& orig);
```

Copy constructor. The data in the source message is cloned and owned.

```
PlClMsgGetRsNmsRsp(HObjCollection* nameList);
```

Constructor taking a list of resource names. I do not assume ownership of the list.

```
PlClMsgGetRsNmsRsp();
```

Default constructor.

```
virtual  ~PlClMsgGetRsNmsRsp();
```

Destructor. I will delete any data that I own.

Operations:

- DECLARE_CLASS

```
int DECLARE_CLASS(PlClMsgGetRsNmsRsp );
```

Class PlClMsgGetRsNmsRsp (resource names message)

PlClMsgGetRsNmsRsp contains the names of all resources known by the resource model. It is sent in response to a request for resource names (PlClMsgRsNms).

- operator =

```
PlClMsgGetRsNmsRsp& operator =(const PlClMsgGetRsNmsRsp& orig);
```

Assignment operator. The data in the source message is cloned and owned.

6.3.3.3.13 PlClMsgGetRsNms Class

Overview:

class PlClMsgGetRsNms (get resource names message)

PlClMsgGetRsNms is a request to get the names of all resources known by the resource model.

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
PlClMsgGetRsNms();
```

Default constructor.

```
PlClMsgGetRsNms(const PlClMsgGetRsNms& orig);
```

Copy constructor.

```
virtual  ~PlClMsgGetRsNms();
```

Destructor.

Operations:

- DECLARE_CLASS

```
int DECLARE_CLASS(PlClMsgGetRsNms );
```

class PlClMsgGetRsNms (get resource names message)

PlClMsgGetRsNms is a request to get the names of all resources known by the resource model.

- operator =

  ```
  PlClMsgGetRsNms& operator =(const PlClMsgGetRsNms& orig);
  ```

  Assignment operator.

6.3.3.3.14 PlClMsgPlanNew Class

Overview:

class PlClMsgPlanNew (New Plan message)

Instances of PlClMsgPlanNew are a message that tells the resource model about a new plan that has been created.

PlClMsgPlanNew is a request to add the new plan it carries to some or all resources, as well as to place the plan in the resource model.

Export Control: Public

Inheritance Relationships:

Inherits from PlClMsgPlan

Attributes:

Constructors and Destructor:

```
PlClMsgPlanNew(const PlClMsgPlanNew& orig);
```

Copy constructor

```
PlClMsgPlanNew(RPlan* );
```

Constructor taking a pointer to a plan.

```
PlClMsgPlanNew();
```

Default constructor

```
virtual  ~PlClMsgPlanNew();
```

Destructor

Operations:

- DECLARE_CLASS

  ```
  int DECLARE_CLASS(PlClMsgPlanNew );
  ```

  class PlClMsgPlanNew (New Plan message)

Instances of PlClMsgPlanNew are a message that tells the resource model about a new plan that has been created.

PlClMsgPlanNew is a request to add the new plan it carries to some or all resources, as well as to place the plan in the resource model.

- operator =

  `PlClMsgPlanNew& operator =(const PlClMsgPlanNew& orig);`

  Assignment operator

6.3.3.3.15 PlClMsgPlanSave Class

Overview:

class PlClMsgPlanSave (save a plan to a file)

PlClMsgPlanSave is a request to save a plan to a file whose name is specified at the command line. If a list of resource Ids is given, then only those resources will be saved, otherwise all of the resources on the specified plan will be saved. In addition, an interval may be specified over which to save. If no interval is specified, then the plan will be saved over all time.

Export Control: Public

Inheritance Relationships:

Inherits from PlClMsgVCol

Attributes:

`myPlanName: HString`

The name of the plan that is to be saved

`mySaveInterval: HEpochInterval`

The interval over which the plan should be saved (all time will be saved if the interval is not specified)

Constructors and Destructor:

`PlClMsgPlanSave(const PlClMsgPlanSave& orig);`

Copy constructor

`PlClMsgPlanSave(HVList* rsIdList);`

Constructor taking member data

`PlClMsgPlanSave();`

Default constructor

```
virtual  ~PlClMsgPlanSave();
```

Destructor

Operations:

- DECLARE_CLASS

```
int DECLARE_CLASS(PlClMsgPlanSave );
```

class PlClMsgPlanSave (save a plan to a file)

PlClMsgPlanSave is a request to save a plan to a file whose name is specified at the command line. If a list of resource Ids is given, then only those resources will be saved, otherwise all of the resources on the specified plan will be saved. In addition, an interval may be specified over which to save. If no interval is specified, then the plan will be saved over all time.

- get

```
virtual int get(istream& istr);
```

Get myself from a stream

- operator =

```
PlClMsgPlanSave& operator =(const PlClMsgPlanSave& orig);
```

Assignment operator

- planName

```
const HString& planName() const;
```

Get my plan name

- planName

```
int planName(const char* name);
```

Set my plan name

- put

```
virtual int put(ostream& ostr) const;
```

Put myself onto a stream

- saveInterval

```
int saveInterval(const HEpochInterval& );
```

Set my save interval

- saveInterval

```
const HEpochInterval& saveInterval() const;
```

Get my save interval

- xdr

```
virtual int xdr(XDR* xdrs);
```

Get/put myself onto an XDR stream

6.3.3.3.16 PlClMsgPlanUpdate Class

Overview:

class PlClMsgPlanUpdate (New Plan message)

Instances of PlClMsgPlanUpdate are messages that inform the resource model that a plan has been updated.

Export Control: Public

Inheritance Relationships:

Inherits from PlClMsgPlan

Attributes:

Constructors and Destructor:

```
PlClMsgPlanUpdate(const PlClMsgPlanUpdate& orig);
```

Copy constructor

```
PlClMsgPlanUpdate(RPlan* );
```

Constructor taking a pointer to a plan.

```
PlClMsgPlanUpdate();
```

Default constructor

```
virtual  ~PlClMsgPlanUpdate();
```

Destructor

Operations:

- DECLARE_CLASS

```
int DECLARE_CLASS(PlClMsgPlanUpdate );
```

class PlClMsgPlanUpdate (New Plan message)

Instances of PlClMsgPlanUpdate are messages that inform the resource model that a plan has been updated.

- operator =

  `PlClMsgPlanUpdate& operator =(const PlClMsgPlanUpdate& orig);`

  Assignment operator

6.3.3.3.17 PlClMsgPlan Class

Overview:

class PlClMsgPlan (Message carrying a plan)

Instances of PlClMsgPlan are messages that have an associated plan. derived classes can add any additional information necessary to do something with the plan, such as add a new plan, or update an existing plan.

Export Control: Public

Inheritance Relationships:

Attributes:

`myPlanOwnFlag: int`

Flag indicating plan ownership

`myPlan: RPlan*`

My plan

Constructors and Destructor:

`PlClMsgPlan(const PlClMsgPlan& orig);`

Copy constructor The data in the source message is cloned and owned.

`PlClMsgPlan(RPlan* );`

Constructor taking a pointer to a plan. I do not assume ownership of the passed in plan.

`PlClMsgPlan();`

Default constructor

`virtual ~PlClMsgPlan();`

Destructor I will delete any data that I own.

Operations:

- cleanup

```
virtual void cleanup();
```

Clean up any data that I own.

- copyOver

```
int copyOver(const PlClMsgPlan& orig);
```

Clone and own the data in the source message.

- DECLARE_CLASS

```
int DECLARE_CLASS(PlClMsgPlan );
```

class PlClMsgPlan (Message carrying a plan)

Instances of PlClMsgPlan are messages that have an associated plan. derived classes can add any additional information necessary to do something with the plan, such as add a new plan, or update an existing plan.

- getPlan

```
virtual RPlan* getPlan();
```

Get my plan.

- get

```
virtual int get(istream& istr);
```

Get myself from a stream

- operator =

```
PlClMsgPlan& operator =(const PlClMsgPlan& orig);
```

Assignment operator The data in the source message is cloned and owned.

- ownData

```
virtual int ownData();
```

Ask me to own my data

- put

```
virtual int put(ostream& ostr) const;
```

Put myself onto a stream

- relData

```
virtual int relData();
```

Ask me to relinquish ownership of my data

- setPlan

```
virtual int setPlan(RPlan* );
```

Set my plan. I do not assume ownership on the set. I will relinquish ownership on the get.

- xdr

```
virtual int xdr(XDR* xdrs);
```

Get/put myself onto an XDR stream

6.3.3.3.18 PlClMsgVCol Class

Overview:

class PlClMsgVCol (collection message with status)

Instances of PlClMsgVCol are messaged that carry void collections from one agent to another. These void collections are ONLY to be used for sending lists of integer values. Any pointers in the list will be treated as an integer, and what they point at will not be sent. NOTE: copy and assignment will result in a DEEP COPY of the messages carried collection I.E. - the message copied to or assigned to will own its own copy of the data.

Export Control: Public

Inheritance Relationships:

Attributes:

```
myCollectionOwnFlag: int
```

Specifies whether or not I own myCollection

```
myCollection: HVList*
```

The collection of integers I am to carry

Constructors and Destructor:

```
PlClMsgVCol(const PlClMsgVCol& orig);
```

Copy constructor

```
PlClMsgVCol(HVList* list);
```

Constructor taking a pointer to a collection. I do not assume ownership of the passed in collection.

```
PlClMsgVCol();
```

Default constructor

```
virtual  ~PlClMsgVCol();
```

Destructor

Operations:

- cleanup

  ```
  void cleanup();
  ```

  Clean up my data

- col

  ```
  virtual HVList* col();
  ```

  Get my collection.

- col

  ```
  virtual int col(HVList* list);
  ```

  Set my collection. I do not assume ownership on the set. I will relinquish ownership on the get.

- copyOver

  ```
  int copyOver(const PlClMsgVCol& orig);
  ```

  Copy over data from the source PlClMsgVCol

- DECLARE_CLASS

  ```
  int DECLARE_CLASS(PlClMsgVCol );
  ```

  class PlClMsgVCol (collection message with status)

  Instances of PlClMsgVCol are messaged that carry void collections from one agent to another. These void collections are ONLY to be used for sending lists of integer values. Any pointers in the list will be treated as an integer, and what they point at will not be sent. NOTE: copy and assignment will result in a DEEP COPY of the messages carried collection I.E. - the message copied to or assigned to will own its own copy of the data.

- get

  ```
  virtual int get(istream& istr);
  ```

  Get myself from a stream

- operator =

  ```
  PlClMsgVCol& operator =(const PlClMsgVCol& orig);
  ```

  Assignment operator

- ownData

  ```
  virtual int ownData();
  ```

  Ask me to own my data

- put

  ```
  virtual int put(ostream& ostr) const;
  ```

  Put myself onto a stream

- relData

  ```
  virtual int relData();
  ```

  Ask me to relinquish ownership of my data

- xdr

  ```
  virtual int xdr(XDR* xdrs);
  ```

  Get/put myself onto an XDR stream

6.3.3.3.19 PlClSrmNoid Class

Overview:

class PlClSrmNoid (ecs resource model client)

Instances of PlClSrmNoid are HNoids that manage the finding, requests to, and receiving information from the system SRM. They add additional behavior over the base SrmCliNoid, as well as adding additional information having to do with requesting allocations that may be oversubscribed.

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
PlClSrmNoid();
```

Default constructor

```
PlClSrmNoid(const PlClSrmNoid& orig);
```

Copy constructor

```
virtual  ~PlClSrmNoid();
```

Destructor.

Operations:

- DECLARE_CLASS

  ```
  int DECLARE_CLASS(PlClSrmNoid );
  ```

  class PlClSrmNoid (ecs resource model client)

  Instances of PlClSrmNoid are HNoids that manage the finding, requests to, and receiving information from the system SRM. They add additional behavior over the base SrmCliNoid, as well as adding additional information having to do with requesting allocations that may be oversubscribed.

- establishInterest

  ```
  virtual int establishInterest();
  ```

  Establish interest with my agent in the get Activity pool response

- handleActPoolRsp

  ```
  virtual int handleActPoolRsp(HMsgConn* , HAddress* ,
  PlClMsgGetActPoolRsp* msg);
  ```

  Method to handle incoming PlClMsgGetActPoolRsp messages. Sets this processes' Activity Pool to be the same as the incoming messages' pool

- handleMessage

  ```
  virtual int handleMessage(HMsgConn* curCon, HAddress* curAdr,
  HMessage* curMsg);
  ```

  Method to handle current inbound messages

- handlePlansGetRsp

  ```
  virtual int handlePlansGetRsp(HMsgConn* , HAddress* ,
  PlClMsgGetPlansRsp* msg);
  ```

  Handle response message to request to get plans.

- handlePlClMsgPlanNew

  ```
  virtual int handlePlClMsgPlanNew(HMsgConn* , HAddress* ,
  PlClMsgPlanNew* msg);
  ```

  Handle derived message for new plan.

- handlePlClMsgPlanUpdate

  ```
  virtual int handlePlClMsgPlanUpdate(HMsgConn* , HAddress* ,
  PlClMsgPlanUpdate* msg);
  ```

  Handle derived message for updated plan.

- handleRsGetNmsRsp

```
virtual int handleRsGetNmsRsp(HMsgConn* , HAddress* ,
PlClMsgGetRsNmsRsp* msg);
```

Handle a message containing the names of all the resources known by the SRM. Default behavior is to just see if the request worked.

- installActNew

```
virtual int installActNew(RActivity* act);
```

Overridden from base class to install this activity into my local activity pool, if there is one and if the activity has a parent id, the activity will be added to the parent as well as the pool. This routine assumes ownership of the passed in RActivity. The default behavior is to add this RActivity to my RActPool, and return TRUE. If there is no RActPool or the RActivity cannot be added to the pool or their is already an RActivity with the given activity id, then delete the RActivity and return FALSE.

- installActPool

```
virtual int installActPool(HObjCollection* col);
```

install the given collection into the local activity pool. Returns FALSE if the addition of any activity to the activity pool fails.

- installPlanCopy

```
virtual int installPlanCopy(const char* srcPlanName, const char*
destPlanName, HEpochInterval* intvl);
```

Overide base class behavior. Do a plan copy from a source plan name to a destination plan name accross some optional interval. I do not assume ownership of the passed in HEpochInterval. Returns TRUE/FALSE as to success/failure.

- installPlanNew

```
virtual int installPlanNew(const char* name, HVList* rsIds);
```

Install a new, empty plan given a plan name and an optional list of resource IDs. If no resource IDs are supplied, then the plan is installed against all RResources in the RRsPool. If a given RResource already has a plan of this name, then that RResource is skipped. This behavior does not presume ownership of the passed in HVList.

- operator =

```
PlClSrmNoid& operator =(const PlClSrmNoid& orig);
```

Assignment operator

- requestActPool

```
virtual int requestActPool(HVList* theCol);
```

Request to receive a copy of the SRM's activity pool

- requestAllocAct

  ```
  virtual int requestAllocAct(const char* planName, const HEpochInterval& intvl, int rsId, RActivity* act, const char* allocatorType);
  ```

  Request that an allocation be performed, with contraint-checking. against an RActivity object. This is overridden from my base class to presume that oversubscription is not allowed.

- requestAllocAct

  ```
  virtual int requestAllocAct(const char* planName, const HEpochInterval& intvl, int rsId, RActivity* act, int ovrsAllowed, const char* allocatorType);
  ```

  Request that an allocation be performed, with contraint-checking. against an RActivity object. This is overloaded from my base class to permit the caller to specify if oversubscription is allowed or not.

- requestAllocCheck

  ```
  virtual int requestAllocCheck(const char* planName, const HEpochInterval& intvl, int rsId, int actId);
  ```

  Request that an allocation be checked. This will determine if the allocation COULD be performed. This defaults to oversubscription not allowed.

- requestAllocCheck

  ```
  virtual int requestAllocCheck(const char* planName, const HEpochInterval& intvl, int rsId, int actId, int ovrsAllowed);
  ```

  Request that an allocation be checked. This will determine if the allocation COULD be performed. Whether oversubscription is permitted may may be specified.

- requestAllocMove

  ```
  virtual int requestAllocMove(int actId, int sourceRsId, int destRsId, const char* sourcePlanName, const char* destPlanName, const HEpochInterval& oldIntvl, const HEpochInterval& newIntvl, int impactFlag);
  ```

  Request that an allocation be moved. The default is for oversubscription to be not allowed.

- requestAllocMove

  ```
  virtual int requestAllocMove(int actId, int sourceRsId, int destRsId, int ovrsAllowed, const char* sourcePlanName, const char* destPlanName, const HEpochInterval& oldIntvl, const HEpochInterval& newIntvl, int impactFlag);
  ```

  Request that an allocation be moved. Whether oversubscription is permitted may may be specified.

- requestAllocRqst

  ```
  virtual int requestAllocRqst(const char* planName,
  HObjCollection& reqsToSch, HObjCollection& schRqsts,
  HObjCollection& notSchRqsts, int ovrsAllowed, const char*
  allocatorType);
  ```

  Request that an allocation be performed, with contraint-checking, against a provided collection of RSchRqst(s). This is overloaded from my base class to permit the caller to specifiy if oversubscription is allowed or not.

- requestAllocRqst

  ```
  virtual int requestAllocRqst(const char* planName,
  HObjCollection& reqsToSch, HObjCollection& schRqsts,
  HObjCollection& notSchRqsts, const char* allocatorType);
  ```

  Request that an allocation be performed, with contraint-checking, against a provided collection of RSchRqst(s). This is overridden from my base class to presume that oversubscription is not allowed.

- requestAlloc

  ```
  virtual int requestAlloc(const char* planName, const
  HEpochInterval& intvl, int rsId, int actId, const char*
  allocatorType);
  ```

  Request that an allocation be performed, with contraint-checking. This is overridden from my base class to presume that oversubscription is not allowed.

- requestAlloc

  ```
  virtual int requestAlloc(const char* planName, const
  HEpochInterval& intvl, int rsId, int actId, int ovrsAllowed,
  const char* allocatorType);
  ```

  Request that an allocation be performed, with contraint-checking. Whether oversubscription is permitted may may be specified.

- requestPlanSave

  ```
  virtual int requestPlanSave();
  ```

  Request to save a plan in a particular file.

- requestPlans

  ```
  virtual int requestPlans(HObjList* plans);
  ```

  Request plans.

- requestPlanUpdate

  ```
  int requestPlanUpdate(const char* planName);
  ```

Notify the resource model that a plan has been updated.

- requestRsNms

  ```
  virtual int requestRsNms();
  ```

  Request to receive a list of names of all the resources in the SRM's resource pool

6.3.4 Planning_Workbench_Db Class Category

6.3.4.1 Overview

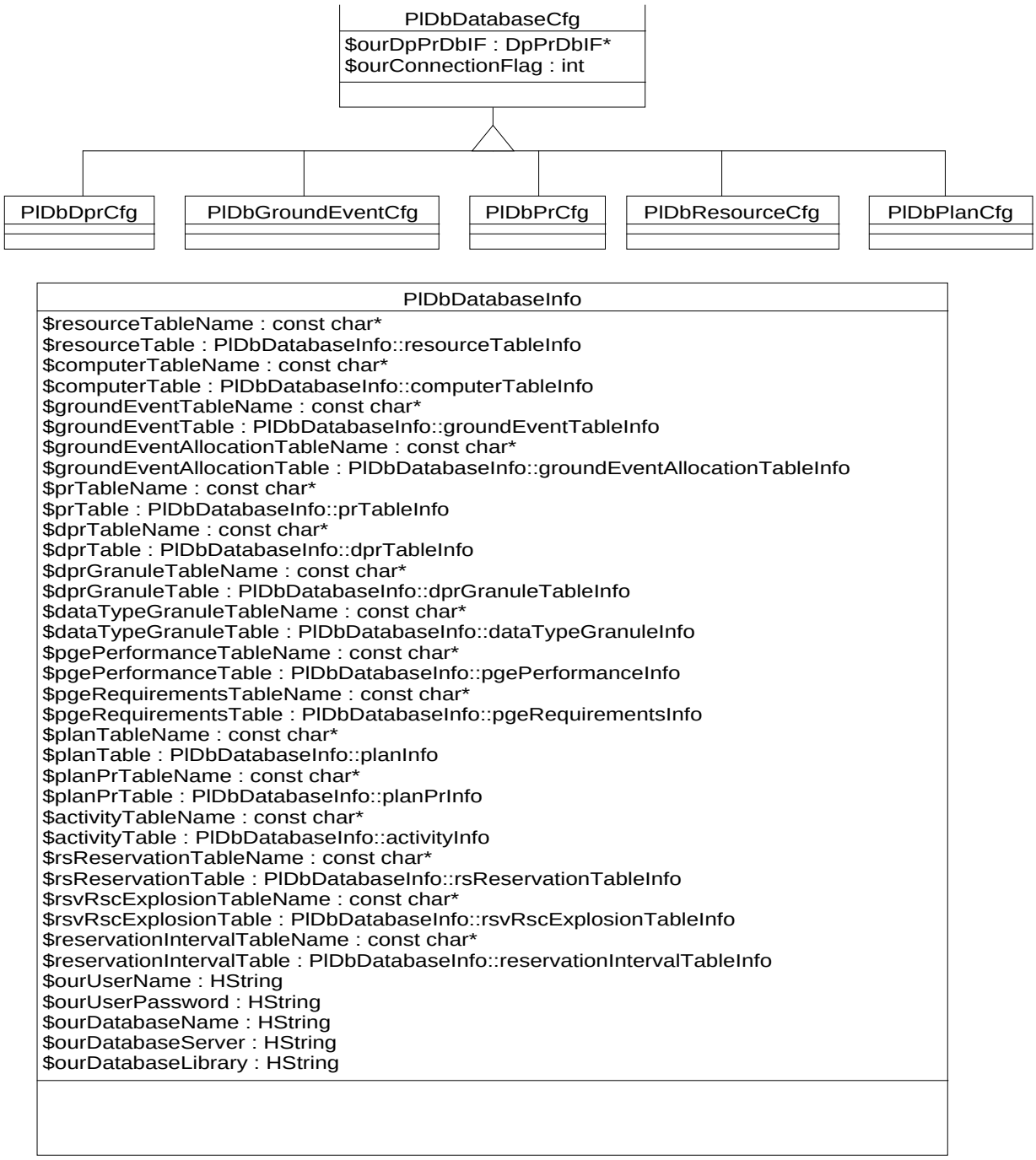


Figure 6.3.4.1-1 Planning_Workbench_Db

6.3.4.2 Planning_Workbench_Db Classes

6.3.4.3 PlDbDatabaseCfg Class

Overview:

class PlDbDatabaseCfg (-)

PlDbDatabaseCfg serves as a base class for all HCL database interface classes.

Export Control: Public

Inheritance Relationships:

Attributes:

`ourConnectionFlag: int`

Flag to indicate whether a database connection has been established.

`ourDpPrDbIF: DpPrDbIF*`

PDPS Database Interface

Constructors and Destructor:

`PlDbDatabaseCfg(const PlDbDatabaseCfg& orig);`

Copy constructor

`PlDbDatabaseCfg(const HString& userName, const HString& userPassword, const HString& databaseName, const HString& databaseServer, const HString& databaseLibrary);`

Constructor takes in the user id, database password, database name, database server, and database name.

`PlDbDatabaseCfg();`

Default constructor

`virtual ~PlDbDatabaseCfg();`

Destructor

Operations:

- databaseIF

`static DpPrDbIF* databaseIF();`

Get my DpPrDbIF

- databaseIF

  ```
  static DpPrDbIF* databaseIF(DpPrDbIF* pDpPrDbIF);
  ```

  Set my DpPrDbIF

- DECLARE_LOCAL_CLASS

  ```
  int DECLARE_LOCAL_CLASS(PlDbDatabaseCfg );
  ```

  class PlDbDatabaseCfg (-)

  PlDbDatabaseCfg serves as a base class for all HCL database interface classes.

- HDateTimeToRWDBDateTime

  ```
  int HDateTimeToRWDBDateTime(const HDateTime& hDateTime,
  RWDBDateTime& rwDBDateTime);
  ```

  Convert from HDateTime to RWDBDateTime and return status.

- init

  ```
  static int init();
  ```

  Initialize connection to database

- operator =

  ```
  PlDbDatabaseCfg& operator =(const PlDbDatabaseCfg& orig);
  ```

  Assignment operator

- queryAll

  ```
  RWTValSlist* queryAll(const HString& tableName) const;
  ```

  Query the database for all rows of specified table, returning the resulting table in an array of ColValLists.

- queryByValue

  ```
  RWTValSlist* queryByValue(const RWDBValue& value, const HString&
  columnName, const HString& table) const;
  ```

  Query table by value and return an array of resulting column/value lists.

- RWDBDateTimeToHDateTime

  ```
  int RWDBDateTimeToHDateTime(const RWDBDateTime& rwDBDateTime,
  HDateTime& hDateTime);
  ```

  Convert from RWDBDateTime to HDateTime and return status.

6.3.4.3.1 PlDbDatabaseInfo Class

Overview:

class PlDbDatabaseInfo (-)

PlDatabaseInfo serves as an encapuslation of the database schema and stores database login information.

Export Control: Public

Inheritance Relationships:

Attributes:

`activityTableName: const char*`

Activity table

`activityTable: PlDbDatabaseInfo::activityInfo`

`computerTableName: const char*`

computer table

`computerTable: PlDbDatabaseInfo::computerTableInfo`

`dataTypeGranuleTableName: const char*`

data type granule table

`dataTypeGranuleTable: PlDbDatabaseInfo::dataTypeGranuleInfo`

`dprGranuleTableName: const char*`

DPR/granule table

`dprGranuleTable: PlDbDatabaseInfo::dprGranuleTableInfo`

`dprTableName: const char*`

DPR master table

`dprTable: PlDbDatabaseInfo::dprTableInfo`

groundEventAllocationTableName: const char*

ground event allocation table

groundEventAllocationTable: PlDbDatabaseInfo::groundEventAllocationTableInfo

groundEventTableName: const char*

ground event master table

groundEventTable: PlDbDatabaseInfo::groundEventTableInfo

ourDatabaseLibrary: HString

RW Access Library

ourDatabaseName: HString

Database database name

ourDatabaseServer: HString

Database server name

ourUserName: HString

Database user Login Id

ourUserPassword: HString

Database user password

pgePerformanceTableName: const char*

PGE performance table

pgePerformanceTable: PlDbDatabaseInfo::pgePerformanceInfo

pgeRequirementsTableName: const char*

PGE resource requirements table

pgeRequirementsTable: PlDbDatabaseInfo::pgeRequirementsInfo

planPrTableName: const char*

Plan / Production request table.

planPrTable: PlDbDatabaseInfo::planPrInfo

planTableName: const char*

Plan table

planTable: PlDbDatabaseInfo::planInfo

prTableName: const char*

Production Request (PR) table

prTable: PlDbDatabaseInfo::prTableInfo

reservationIntervalTableName: const char*

Resource interval table

reservationIntervalTable: PlDbDatabaseInfo::reservationIntervalTableInfo

resourceTableName: const char*

Database table structures // resource master table

resourceTable: PlDbDatabaseInfo::resourceTableInfo

rsReservationTableName: const char*

Resource reservation table

rsReservationTable: PlDbDatabaseInfo::rsReservationTableInfo

rsvRscExplosionTableName: const char*

Resource reservation/ Resource explosion table

rsvRscExplosionTable: PlDbDatabaseInfo::rsvRscExplosionTableInfo

Constructors and Destructor:

PlDbDatabaseInfo(const PlDbDatabaseInfo& orig);

Copy constructor

```
PlDbDatabaseInfo(const HString& userName, const HString&
userPassword, const HString& databaseName, const HString&
databaseServer, const HString& databaseLibrary);
```

Constructor takes in the user id, database password, database name, database server, and database name.

```
PlDbDatabaseInfo();
```

Default constructor

```
virtual  ~PlDbDatabaseInfo();
```

Destructor

Operations:

- databaseLibrary

```
static int databaseLibrary(const HString& dbLibrary);
```

Set my access Library

- databaseLibrary

```
static const HString& databaseLibrary();
```

Get my access Library

- databaseName

```
static const HString& databaseName();
```

Get my database name

- databaseName

```
static int databaseName(const HString& dbName);
```

Set my database name

- databaseServer

```
static const HString& databaseServer();
```

Get my server name

- databaseServer

```
static int databaseServer(const HString& dbServer);
```

Set my server name

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlDbDatabaseInfo );
```

class PlDbDatabaseInfo (-)

PlDatabaseInfo serves as an encapuslation of the database schema and stores database login information.

- operator =

```
PlDbDatabaseInfo& operator =(const PlDbDatabaseInfo& orig);
```

Assignment operator

- userName

```
static int userName(const HString& uName);
```

Set my user login id

- userName

```
static const HString& userName();
```

Get my user login id

- userPassword

```
static const HString& userPassword();
```

Get my user password

- userPassword

```
static int userPassword(const HString& uPassword);
```

Set my user password

6.3.4.3.2 PlDbDprCfg Class

Overview:

class PlDbDprCfg (-)

PlDbDprCfg allows loading and saving data processing request (DPR) activities (PlAcDpr) from and to the database tables.

Export Control: Public

Inheritance Relationships:

Inherits from PlDbDatabaseCfg

Attributes:

Constructors and Destructor:

```
PlDbDprCfg();
```

Default constructor

```
PlDbDprCfg(const PlDbDprCfg& orig);
```

Copy constructor

```
virtual  ~PlDbDprCfg();
```

Destructor

Operations:

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlDbDprCfg );
```

class PlDbDprCfg (-)

PlDbDprCfg allows loading and saving data processing request (DPR) activities (PlAcDpr) from and to the database tables.

- extractAll

```
virtual int extractAll();
```

Retrieve all ground activities from their database tables

- extractPgeInformation

```
virtual int extractPgeInformation();
```

Extract PGE information for all DPRs from database.

- init

```
virtual int init();
```

Initialize by connecting to the database containing DPR information.

- load

```
virtual int load();
```

Load the DPR activities from the database tables.

- makeNewObject

```
virtual HObject* makeNewObject(const DpPrDbColValList&
thisColValList);
```

Create an instance of a DPR from a ColValList and associated detail tables.

- makeObjects

```
virtual int makeObjects(const RWTValSlist& resultsList);
```

Create new instances of DPRs from an array of ColValLists and add them to the activity pool.

- operator =

```
PlDbDprCfg& operator =(const PlDbDprCfg& orig);
```

Assignment operator

- resolveDprDependencies

```
virtual int resolveDprDependencies();
```

Create predecessor DPR lists

- resolveExternalDataDependencies

```
int resolveExternalDataDependencies();
```

Create external data dependency lists and determine earliest possible start times based on the latest arriving external data dependency.

- save

```
virtual int save();
```

Save the current DPR activity pool to the database tables.

- updateActivatedDprs

```
virtual int updateActivatedDprs(HObjCollection& dprCollection);
```

Update activated DPRs in the database

6.3.4.3.3 PlDbGroundEventCfg Class

Overview:

class PlDbGroundEventCfg (-)

PlDbGroundEventCfg allows loading and saving ground event activities (PlAcGroundEvent) from and to the database tables.

Export Control: Public

Inheritance Relationships:

Inherits from PlDbDatabaseCfg

Attributes:

Constructors and Destructor:

```
PlDbGroundEventCfg();
```

Default constructor

```
PlDbGroundEventCfg(const PlDbGroundEventCfg& orig);
```

Copy constructor

```
virtual  ~PlDbGroundEventCfg();
```

Destructor

Operations:

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlDbGroundEventCfg );
```

class PlDbGroundEventCfg (-)

PlDbGroundEventCfg allows loading and saving ground event activities (PlAcGroundEvent) from and to the database tables.

- extractAll

```
virtual int extractAll();
```

Retrieve all ground event activities from their database tables

- extract

```
PlAcGroundEvent* extract(PlAcGroundEvent* newGroundEvent);
```

Extract allocations to resources for the given ground event.

- init

```
virtual int init();
```

Initialize by connecting to the database containing ground event information.

- load

```
virtual int load();
```

Load the ground event activities from the database tables.

- makeNewObject

```
virtual HObject* makeNewObject(const DpPrDbColValList&
thisColValList);
```

Create an instance of a ground event from a ColValList and associated detail tables.

- makeObjects

```
virtual int makeObjects(const RWTValSlist& resultsList);
```

Create new instances of ground events from an array of ColValLists and add them to the activity pool.

- operator =

```
PlDbGroundEventCfg& operator =(const PlDbGroundEventCfg& orig);
```

Assignment operator

- save

```
virtual int save();
```

Save the current ground event activity pool off to the database tables.

6.3.4.3.4 PlDbPlanCfg Class

Overview:

class PlDbPlanCfg (-)

PlDbPlanCfg allows loading and saving of Plan information (PlPlPlan) from and to the database tables.

Export Control: Public

Inheritance Relationships:

Inherits from PlDbDatabaseCfg

Attributes:

Constructors and Destructor:

```
PlDbPlanCfg();
```

Default constructor

```
PlDbPlanCfg(const PlDbPlanCfg& orig);
```

Copy constructor

```
virtual  ~PlDbPlanCfg();
```

Destructor

Operations:

- baseline

```
virtual int baseline(const char* planName);
```

Baseline the given plan in the database.

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlDbPlanCfg );
```

class PlDbPlanCfg (-)

PlDbPlanCfg allows loading and saving of Plan information (PlPlPlan) from and to the database tables.

- deleteFromDatabase

```
virtual int deleteFromDatabase(const char* planName);
```

Delete a plan from the database. Compiler would not let us use the name "delete".

- extractAll

```
virtual int extractAll();
```

Retrieve all plans from their database tables

- extractPrs

```
virtual int extractPrs();
```

Retrieve production request information for all plans

- init

```
virtual int init();
```

Initialize by connecting to the database containing plan information.

- insertDprs

```
int insertDprs(PlPlPlan* plan);
```

Insert the DPRs for the given plan into the database.

- insertPrs

```
int insertPrs(PlPlPlan* plan);
```

Insert the production requests for the given plan into the database.

- insert

```
virtual int insert(PlPlPlan* plan);
```

Insert a new plan into the database.

- load

```
virtual int load();
```

Load the plans from the database tables.

- makeNewObject

```
virtual HObject* makeNewObject(const DpPrDbColValList&
thisColValList);
```

Create an instance of a plan from a ColValList and associated detail tables.

- makeObjects

```
virtual int makeObjects(const RWTValSlist& resultsList);
```

Create new instances of plans from an array of ColValLists and add them to the activity pool.

- operator =

```
PlDbPlanCfg& operator =(const PlDbPlanCfg& orig);
```

Assignment operator

- save

```
virtual int save(const char* planName);
```

Insert or update the given plan in the database.

- update

```
virtual int update(PlPlPlan* plan);
```

Update an old plan in the database.

6.3.4.3.5 PlDbPrCfg Class

Overview:

class PlDbPrCfg (-)

PlDbPrCfg allows loading and saving Production Request activities (PlAcPr) from and to the database tables.

Export Control: Public

Inheritance Relationships:

Inherits from PlDbDatabaseCfg

Attributes:

Constructors and Destructor:

```
PlDbPrCfg();
```

Default constructor

```
PlDbPrCfg(const PlDbPrCfg& orig);
```

Copy constructor

```
virtual  ~PlDbPrCfg();
```

Destructor

Operations:

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlDbPrCfg );
```

class PlDbPrCfg (-)

PlDbPrCfg allows loading and saving Production Request activities (PlAcPr) from and to the database tables.

- extractAll

```
virtual int extractAll();
```

Retrieve all production request activities from their database tables

- handleConnections

```
virtual int handleConnections();
```

Resolve connections with associated DPRs in activity pool. This assumes that all DPRs have already been read from the database into the pool.

- init

```
virtual int init();
```

Initialize by connecting to the database containing production request information.

- load

```
virtual int load();
```

Load the production request activities from the database tables.

- makeNewObject

```
virtual HObject* makeNewObject(const DpPrDbColValList&
thisColValList);
```

Create an instance of a production request from a ColValList and associated detail tables.

- makeObjects

```
virtual int makeObjects(const RWTValSlist& resultsList);
```

Create new instances of production requests from an array of ColValLists and add them to the activity pool.

- operator =

```
PlDbPrCfg& operator =(const PlDbPrCfg& orig);
```

Assignment operator

- save

```
virtual int save();
```

Save the current production request activity pool off to the database tables.

6.3.4.3.6 PlDbResourceCfg Class

Overview:

class PlDbResourceCfg (-)

PlDbResourceCfg allows loading the resource pool from the database tables and saving to the database tables from the resource pool. PlDbResourceCfg is also responsible for creating schedulable resources and adding them to the schedulable resource pool.

Export Control: Public

Inheritance Relationships:

Inherits from PlDbDatabaseCfg

Attributes:

Constructors and Destructor:

```
PlDbResourceCfg();
```

Default constructor

```
PlDbResourceCfg(const PlDbResourceCfg& orig);
```

Copy constructor

```
virtual  ~PlDbResourceCfg();
```

Destructor

Operations:

- createSchedulableResources

```
virtual int createSchedulableResources() const;
```

Create new schedulable resources (PlRcResource) and add them to the schedulable resource pool.

- DECLARE_LOCAL_CLASS

  `int DECLARE_LOCAL_CLASS(PlDbResourceCfg );`

  class PlDbResourceCfg (-)

  PlDbResourceCfg allows loading the resource pool from the database tables and saving to the database tables from the resource pool. PlDbResourceCfg is also responsible for creating schedulable resources and adding them to the schedulable resource pool.

- extractAll

  `virtual int extractAll();`

  Retrieve all resource objects from their database tables

- extract

  `virtual PlRcResource* extract(PlRcComputer* newComputer);`

  Overloaded extraction methods for completing the construction of resource objects from the database.

- extract

  `virtual PlRcResource* extract(PlRcDataServer* dataServer);`

  Extract data server information

- handleConnections

  `virtual int handleConnections();`

  Resolve connections between associated resources.

- init

  `virtual int init();`

  Initialize by connecting to the database containing resource information.

- load

  `virtual int load();`

  Load the resource pool from the database tables.

- makeNewObject

  `virtual HObject* makeNewObject(const DpPrDbColValList& thisColValList);`

  Create an instance of a resource from a ColValList and associated detail tables.

- makeObjects

  ```
  virtual int makeObjects(const RWTValSlist& resultsList);
  ```

  Create new instances of resources from an array of ColValLists and add them to the resource pool.

- operator =

  ```
  PlDbResourceCfg& operator =(const PlDbResourceCfg& orig);
  ```

  Assignment operator

- save

  ```
  virtual int save();
  ```

  Save the current resource pool off to the database tables.

6.3.5 Planning_Workbench_Di Class Category

6.3.5.1 Overview

PlDiAppl
myWindow : DWindow* myFallbacks : HObjList myFallbackArray : char**
DECLARE_ABS_CLASS( ) makeWindow( ) create( ) setFallbackResources( ) setNotifier( ) run( ) window( ) closeAppl( ) quit( ) connectionMade( ) $winManagerClose( ) displayWinAtRun( ) catchCloseEvent( ) cleanupForWinDestroy( )

6.3.5.2 Planning_Workbench_Di

6.3.5.3 PlDiAppl Class

Overview:

class PlDiAppl (ECS Motif Application Class)

An instance of this class is a type of resource model client display application. PlDiAppl includes behavior common to all resource model client display applications.

NOTE: PlDiAppl copies the functionality of DAppl in the md and mdisp libraries but differs in the create by allowing the shell to resize, and in the run by not calling displayWinAtRun() until after a connection to the resource model has been made

Export Control: Public

Inheritance Relationships:

Inherits from PlClAppl

Attributes:

`myFallbackArray: char**`

My Xm resource fallback array

`myFallbacks: HObjList`

Collection of HStrings with my fallback resources.

`myWindow: DWindow*`

My application window

Constructors and Destructor:

`PlDiAppl();`

Default constructor

`PlDiAppl(const PlDiAppl& );`

Copy constructor (Disallows copying)

`virtual  ~PlDiAppl();`

Destructor

Operations:

- catchCloseEvent

  `virtual void catchCloseEvent();`

  Called by PlDiAppl::run to catch the window Close event.

- cleanupForWinDestroy

  `virtual void cleanupForWinDestroy();`

  Called by closeAppl just before top-level window is destroyed. Override if you want to do something before this happens.

- closeAppl

```
virtual void closeAppl();
```

Call from the window if you would like to quit the application.

- connectionMade

```
virtual int connectionMade();
```

This member is called when the srm noid has made a connection to the resource model. Overrides PlClAppl::connectionMade() to call displayWinAtRun() to realize the window

- create

```
virtual int create(const char* appClass, int& argc, char** argv,
XrmOptionDescList options, Cardinal numOptions);
```

This create calls makeWindow, creates a shell, and calls myWindow->createDisplay. If you want to override this functionality, you will still need to pass myWindow::create(..) a shell widget as shown in this function. NOTE: this is the only function (so far) that diverges from the member functions in DAppl in that it allows the shell to resize.

- DECLARE_ABS_CLASS

```
int DECLARE_ABS_CLASS(PlDiAppl );
```

class PlDiAppl (ECS Motif Application Class)

An instance of this class is a type of resource model client display application. PlDiAppl includes behavior common to all resource model client display applications.

NOTE: PlDiAppl copies the functionality of DAppl in the md and mdisp libraries but differs in the create by allowing the shell to resize, and in the run by not calling displayWinAtRun() until after a connection to the resource model has been made

- displayWinAtRun

```
virtual void displayWinAtRun();
```

Display the window at run. default behavior is to realize it

- makeWindow

```
virtual DWindow* makeWindow() = 0;
```

Override this class in your derived class to make the window (i.e., return new DerivedWindow();).

- operator =

```
PlDiAppl& operator =(const PlDiAppl& );
```

Assignment operator (Disallows assignments)

- quit

```
virtual void quit();
```

Quit the application. Overrides PlClAppl::quit to close the display application

- run

  `virtual void run();`

  Run the application. Create(..) should have already been called.

- setFallbackResources

  `virtual int setFallbackResources(char** fallbackList);`

  list of char** arguments with fallback resources. This can be called anytime prior to PlDiAppl::create().

  + an example of use: + + static char *fallbacks[] = { + "hmi.mainWin.background: papaya whip", + "hmi.mainWin.foreground: pale violet red", + "hmi.mainWin.btn.labelString: Push Me, Dude", + "*fontList: -*-palatino-*-r-*--14-*", + NULL}; /* don't forget this! */ + + myAppl.setFallbackResources(fallbacks);

- setNotifier

  `virtual int setNotifier(HNotifier* nf);`

  Set the notifier for this class

- window

  `virtual DWindow* window();`

  Get the window data.

- winManagerClose

  `static void winManagerClose(Widget protocolWidget, XtPointer clientData, XtPointer callData);`

  The function called on window manager menu "close." Cleans up the application and exits.

6.3.6 Planning_Workbench_Ip Class Category

6.3.6.1 Overview

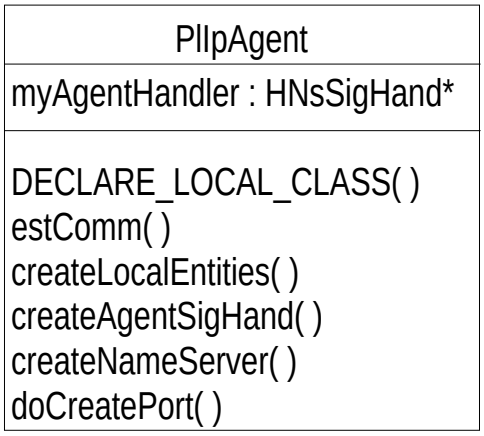


Figure 6.3.6.1-1

6.3.6.2 Planning_Workbench_Ip

6.3.6.3 PlIpAgent Class

Overview:

PlIpAgent

Instances of PlIpAgent are HNoidAgents that have choosen a particular NameServAbs implementation, register their HAddress with said NameServAbs, and interact with the global HSigMgr to install a HNsSigHand to take over in the event of disaster

Export Control: Public

Inheritance Relationships:

Attributes:

`myAgentHandler`:

Holds a signal handler

Constructors and Destructor:

```
PlIpAgent(void );
```

Default constructor

```
PlIpAgent(const PlIpAgent& orig);
```

Copy Constructor

```
virtual  ~PlIpAgent(void );
```

Destructor

Operations:

- createAgentSigHand

```
virtual int createAgentSigHand(void );
```

Create my agent signal handler

- createLocalEntities

```
virtual int createLocalEntities(void );
```

Create any local entities that are needed later. This is overridden from a base class to create my HNsSigHand and my HNameServAbs.

- createNameServer

```
virtual int createNameServer(void );
```

Create my HNameServAbs

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlIpAgent );
```

PlIpAgent

Instances of PlIpAgent are HNoidAgents that have choosen a particular NameServAbs implementation, register their HAddress with said NameServAbs, and interact with the global HSigMgr to install a HNsSigHand to take over in the event of disaster.

- doCreatePort

```
virtual int doCreatePort(void );
```

Returns TRUE so a service will be created. The port number will be chosen at create time. This is overridden from a base class.

- estComm

```
virtual int estComm(void );
```

Overridden from a base class to add initialization and registration of my agent signal handler.

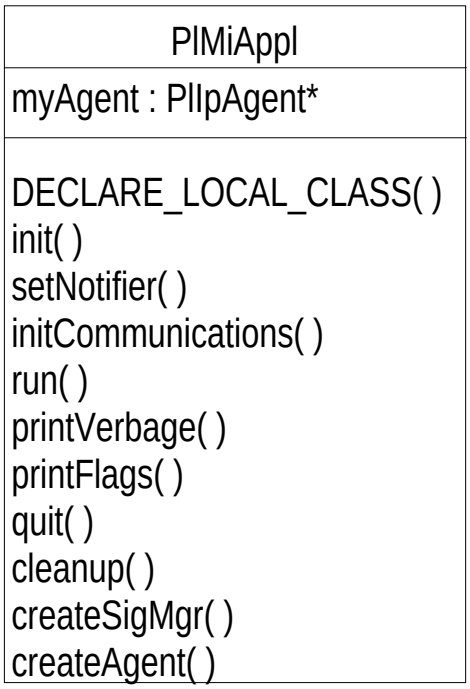
- operator =

```
PlIpAgent& operator =(const PlIpAgent& orig);
```

Assignment Operator

6.3.7 Planning_Workbench_Mi Class Category

6.3.7.1 Overview



6.3.7.2 Planning_Workbench_Mi

6.3.7.3 PlMiAppl Class

Overview:

class PlMiAppl (-)

Instances of PlMiAppl are base application classes for all applications. Functionality is included for creating the global signal manager for the process, printing some default flags and verbage, initializing, and running.

Export Control: Public

Inheritance Relationships:

Attributes:

```
myAgent: PlIpAgent*
```

My message agent

Constructors and Destructor:

```
PlMiAppl();
```

Default constructor

```
PlMiAppl(const PlMiAppl& orig);
```

Copy constructor, restricted

```
virtual  ~PlMiAppl();
```

Destructor

Operations:

- cleanup

```
void cleanup();
```

Cleanup my internals

- createAgent

```
virtual int createAgent();
```

Create and initialize my message agent. Default behavior is to create an PlIpAgent

- createSigMgr

```
virtual int createSigMgr();
```

Create and initialize the global signal catcher. The default behavior is to create a HSigMgr.

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlMiAppl );
```

class PlMiAppl (-)

Instances of PlMiAppl are base application classes for all applications. Functionality is included for creating the global signal manager for the process, printing some default flags and verbage, initializing, and running.

- initCommunications

```
virtual int initCommunications();
```

Initialize the communications part of the application. It will create and initialize a message agent. Derived classes override and call the base class initCommunications.

- init

  `virtual int init();`

  Initialize the application. Derived classes override and call the base class init.

- operator =

  `PlMiAppl& operator =(const PlMiAppl& orig);`

  Assignment operator, restricted

- printFlags

  `virtual void printFlags();`

  Dump a description of what flags are availible and required to standard output.

- printVerbage

  `virtual void printVerbage();`

  Dump a description of what this process does to standard output.

- quit

  `virtual void quit();`

  Quit the application.

- run

  `virtual void run();`

  Run the application. Init should have been already called. The default behavior is to run a HNotifier to catch messages. Derived class should override to create their own polling schemes.

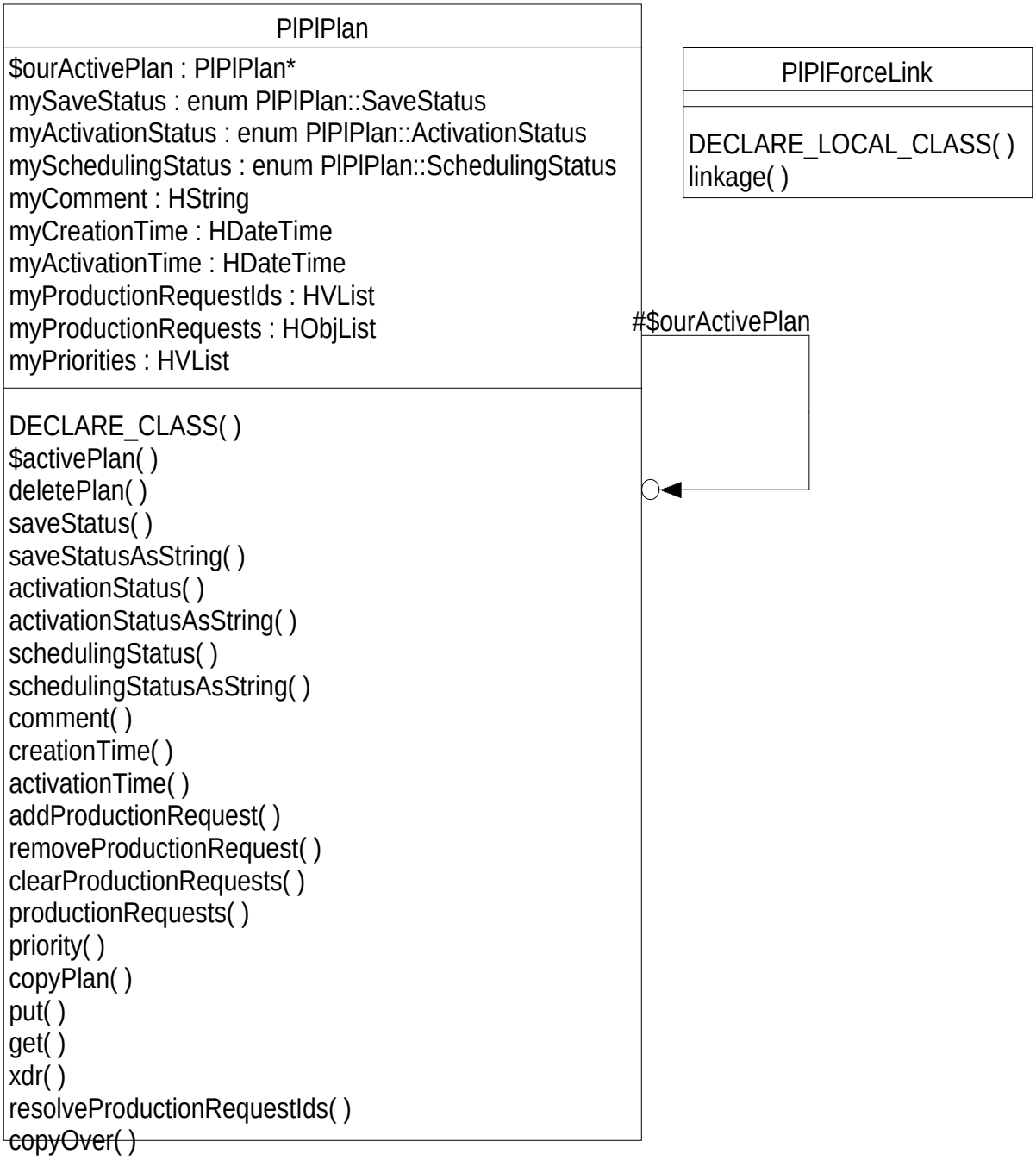
- setNotifier

  `virtual int setNotifier(HNotifier* nf);`

  Set the notifier.

6.3.8 Planning_Workbench_Pl Class Category

6.3.8.1 Overview



6.3.8.2 Planning_Workbench_Pl Classes

6.3.8.3 PlPlForceLink Class

Overview:

class PlPlForceLink

Instances of PlPlForceLink are used to force linkage of classes which may not be present at link time.

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
PlPlForceLink();
```

Default constructor

```
PlPlForceLink(const PlPlForceLink& orig);
```

Copy constructor

```
virtual  ~PlPlForceLink();
```

Destructor

Operations:

- DECLARE_LOCAL_CLASS

  ```
  int DECLARE_LOCAL_CLASS(PlPlForceLink );
  ```

  class PlPlForceLink

  Instances of PlPlForceLink are used to force linkage of classes which may not be present at link time.

- linkage

  ```
  void linkage();
  ```

  link stuff

- operator =

`PlPlForceLink& operator =(const PlPlForceLink& orig);`

Assignment operator

6.3.8.3.1 PlPlPlan Class

Overview:

class PlPlPlan (-)

Instances of PlPlPlan are production plans.

Export Control: Public

Inheritance Relationships:

Attributes:

`myActivationStatus: enum PlPlPlan::ActivationStatus`

My activation status

`myActivationTime: HDateTime`

My activation date/time

`myComment: HString`

My comment

`myCreationTime: HDateTime`

My creation date/time

`myPriorities: HVList`

My production request pritorities.

`myProductionRequestIds: HVList`

My production request activitity IDs. The list contains unique activity IDs.

`myProductionRequests: HObjList`

My production request activitities. The list should contains unique pointers.

`mySaveStatus: enum PlPlPlan::SaveStatus`

My save status

`mySchedulingStatus: enum PlPlPlan::SchedulingStatus`

My scheduling status

`ourActivePlan: PlPlPlan*`

Our active plan

Constructors and Destructor:

```
PlPlPlan();
```

Default constructor

```
PlPlPlan(const PlPlPlan& orig);
```

Copy constructor

```
virtual  ~PlPlPlan();
```

destructor

Operations:

- activationStatusAsString

```
virtual const char* activationStatusAsString() const;
```

Return my activation status as a character string.

- activationStatus

```
virtual enum PlPlPlan::ActivationStatus activationStatus()
const;
```

Get my activation status

- activationStatus

```
virtual int activationStatus(enum PlPlPlan::ActivationStatus );
```

Set my activation status

- activationStatus

```
virtual int activationStatus(const char* aStatus);
```

Set my activation status with a character string. Return TRUE/FALSE if the set worked or not.

- activationTime

```
virtual const HDateTime& activationTime() const;
```

Get the date/time this plan was last activated.

- activationTime

```
virtual int activationTime(const HDateTime& aDateTime);
```

Set the date/time this plan was last activated.

- activePlan

```
static PlPlPlan* activePlan();
```

Get our active plan.

- activePlan

```
static int activePlan(const char* planName);
```

Set our active plan. Takes the plan's name.

- activePlan

```
static int activePlan(PlPlPlan* aplan);
```

Set our active plan. Takes a pointer to a plan.

- addProductionRequest

```
virtual int addProductionRequest(const HString&
productionRequestName);
```

Add production request to list of PRs by name.

- addProductionRequest

```
virtual int addProductionRequest(PlAcPr* productionRequest);
```

Add production request to list of PRs by object.

- clearProductionRequests

```
void clearProductionRequests();
```

Clear the list of production requests.

- comment

```
virtual int comment(const char* aString);
```

Set my comment.

- comment

```
virtual const HString& comment() const;
```

Get my comment.

- copyOver

```
virtual int copyOver(const PlPlPlan& plan);
```

Clone and own the allocations in the source list.

- copyPlan

```
virtual int copyPlan(RPlan* sourcePlan, HEpochInterval* intvl);
```

Overide base class behavior. Copy over internals. Also, add a copy of the plan information from the activity and resource pool to my plan. If no interval is provided, then all of the plan is copied. I do not assume ownership of any passed in HEpochInterval.

- creationTime

  ```
  virtual const HDateTime& creationTime() const;
  ```

  Get my creation date/time.

- creationTime

  ```
  virtual int creationTime(const HDateTime& aDateTime);
  ```

  Set my creation date/time.

- DECLARE_CLASS

  ```
  int DECLARE_CLASS(PlPlPlan );
  ```

  class PlPlPlan (-)

  Instances of PlPlPlan are production plans.

- deletePlan

  ```
  virtual int deletePlan();
  ```

  Delete all traces of this plan.

- get

  ```
  virtual int get(istream& istr);
  ```

  Get myself onto a stream

- operator =

  ```
  PlPlPlan& operator =(const PlPlPlan& orig);
  ```

  Assignment operator

- priority

  ```
  virtual int priority(PlAcPr* productionRequest, int value);
  ```

  Set the priority of a production request for this plan. Returns success or failure. Operation will fail if the production request is not found in my list of production requests.

- priority

  ```
  virtual int priority(PlAcPr* productionRequest);
  ```

  Get the priority of a production request. Returns zero (currently an invalid priority) upon failure.

- priority

```
virtual int priority(const HString& productionRequestName, int
value);
```

- productionRequests

```
virtual HObjIter* productionRequests();
```

Get an iterator over my list of production requests.

- put

```
virtual int put(ostream& ostr) const;
```

Put myself onto a stream

- removeProductionRequest

```
virtual int removeProductionRequest(const HString&
productionRequestName);
```

Remove production request from the list of PRs by name.

- removeProductionRequest

```
virtual int removeProductionRequest(PlAcPr* productionRequest);
```

Remove production request from the list of PRs by object.

- resolveProductionRequestIds

```
virtual void resolveProductionRequestIds();
```

Try to resolve any production request ids to production requests from the activity pool.

- saveStatusAsString

```
virtual const char* saveStatusAsString() const;
```

Return my save status as a character string.

- saveStatus

```
virtual enum PlPlPlan::SaveStatus saveStatus() const;
```

Get my save status

- saveStatus

```
virtual int saveStatus(const char* aStatus);
```

Set my save status with a character string. Return TRUE/FALSE if the set worked or not.

- saveStatus

```
virtual int saveStatus(enum PlPlPlan::SaveStatus );
```

Set my save status

- schedulingStatusAsString

  `virtual const char* schedulingStatusAsString() const;`

  Return my scheduling status as a character string.

- schedulingStatus

  `virtual enum PlPlPlan::SchedulingStatus schedulingStatus() const;`

  Get my scheduling status

- schedulingStatus

  `virtual int schedulingStatus(const char* aStatus);`

  Set my scheduling status with a character string. Return TRUE/FALSE if the set worked or not.

- schedulingStatus

  `virtual int schedulingStatus(enum PlPlPlan::SchedulingStatus );`

  Set my scheduling status

- xdr

  `virtual int xdr(XDR* xdrs);`

  Get/put myself onto an XDR stream

6.3.9 Planning_Workbench_Rc Class Category

6.3.9.1 Overview

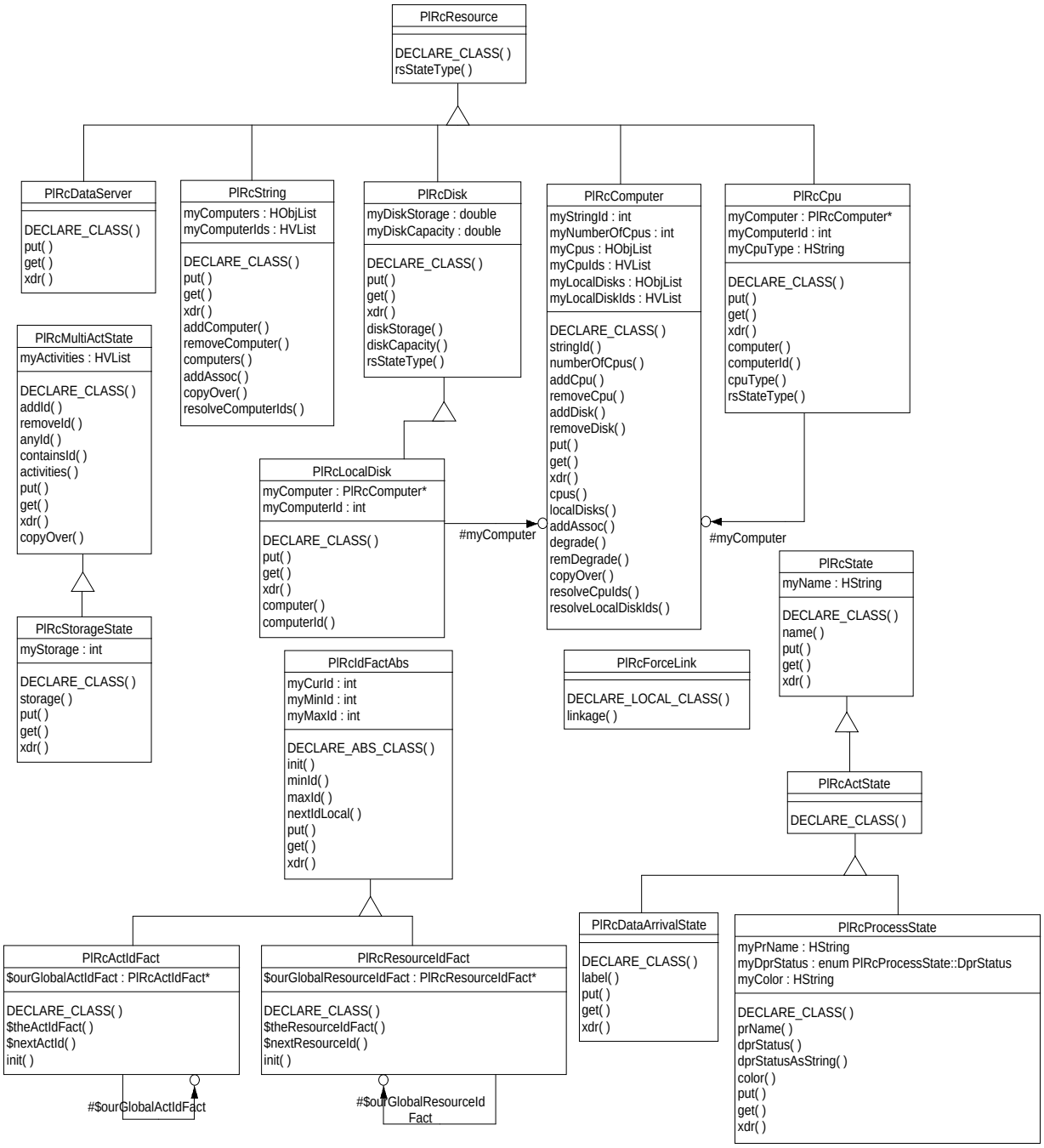


Figure 6.3.9.1-1 Planning_Workbench_Rc

6.3.9.2 Planning_Workbench_Rc Classes

6.3.9.3 PlRcActIdFact Class

Overview:

class PlRcActIdFact (Unique activity id factory)

Instances of PlRcActIdFact are activity ID generators. You can ask them for the next allowable activity ID that can be assigned. This class initializes its activity ID range from the current activity pool for now. Max ID is presumed to be MAXINT.

Export Control: Public

Inheritance Relationships:

Inherits from PlRcIdFactAbs

Attributes:

```
ourGlobalActIdFact: PlRcActIdFact*
```

Current global ID.

Constructors and Destructor:

```
PlRcActIdFact();
```

Default constructor

```
PlRcActIdFact(const PlRcActIdFact& orig);
```

Copy constructor

```
virtual  ~PlRcActIdFact();
```

Destructor

Operations:

- DECLARE_CLASS

```
int DECLARE_CLASS(PlRcActIdFact );
```

class PlRcActIdFact (Unique activity id factory)

Instances of PlRcActIdFact are activity ID generators. You can ask them for the next allowable activity ID that can be assigned. This class initializes its activity ID range from the current activity pool for now. Max ID is presumed to be MAXINT.

- init

```
virtual int init();
```

Initialize. Set myCurId, myMinId, and myMaxId to some initial values, based upon the contents of the current activity pool. Max ID value is presumed to be MAXINT.

- nextActId

  ```
  static int nextActId();
  ```

  Return the next available ID in the global PlRcActIdFact.

- operator =

  ```
  PlRcActIdFact& operator =(const PlRcActIdFact& orig);
  ```

  Assignment operator

- theActIdFact

  ```
  static PlRcActIdFact* theActIdFact();
  ```

  Get the global PlRcActIdFact.

- theActIdFact

  ```
  static int theActIdFact(PlRcActIdFact* idFactory);
  ```

  Set the global PlRcActIdFact. This class does not assume ownership.

6.3.9.3.1 PlRcActState Class

Overview:

class PlRcActState (Activity Resource State)

Instances of PlRcActState model the state of a resource during which that resource is in the process of executing an activity.

Export Control: Public

Inheritance Relationships:

Inherits from PlRcState

Attributes:

Constructors and Destructor:

```
PlRcActState(const HEpochInterval& intvl, int actId, const char*
name);
```

Constructor with interval, acitivity id and name specified

```
PlRcActState(const PlRcActState& orig);
```

Copy constructor

```
PlRcActState(const HEpochInterval& intvl, int actId);
```

Constructor with interval and acitivity id

```
PlRcActState();
```

Default constructor

```
virtual  ~PlRcActState();
```

Destructor

Operations:

- DECLARE_CLASS

```
int DECLARE_CLASS(PlRcActState );
```

class PlRcActState (Activity Resource State)

Instances of PlRcActState model the state of a resource during which that resource is in the process of executing an activity.

- operator =

```
PlRcActState& operator =(const PlRcActState& orig);
```

Assignment operator

6.3.9.3.2 PlRcComputer Class

Overview:

class PlRcComputer (-)

Instances of PlRcComputer are multi-processors with a known configuration of associated disks.

Export Control: Public

Inheritance Relationships:

Inherits from PlRcResource

Attributes:

```
myCpuIds: HVList
```

My list of CPU ids corresponding to the above list of CPUs. I may have ids without having CPUs if I have just been loaded from a stream (see resolveIds() below).

```
myCpus: HObjList
```

List of associated CPUs with this computer; I own these CPUs exclusively

myLocalDiskIds: HVList

My list of Local disk ids corresponding to the above list of LocalDisks. I may have ids without having disks if I have just been loaded from a stream (see resolveIds() below).

myLocalDisks: HObjList

List of LocalDisks associated with this computer; I own these disks exclusively and am in charge of deleting them

myNumberOfCpus: int

Number of CPUs

myStringId: int

My string ID

Constructors and Destructor:

PlRcComputer();

Default constructor

PlRcComputer(const PlRcComputer& orig);

Copy constructor

virtual ~PlRcComputer();

Destructor

Operations:

- addAssoc

virtual int addAssoc(HObject* rs);

Tell me that the given Resource is owned by me.

- addCpu

virtual int addCpu(RResource* cpu);

Add a CPU to me

- addDisk

virtual int addDisk(RResource* disk);

Add a disk to me.

- copyOver

virtual int copyOver(const PlRcComputer& orig);

Copy internals from supplied PlRcComputer

- copyOver

  ```
  virtual int copyOver(const RResource& orig);
  ```

  Copy internals from supplied RResource

- cpus

  ```
  virtual HObjIter* cpus();
  ```

  Return an iterator over my CPU list. The consumer must delete the iterator when done.

- DECLARE_CLASS

  ```
  int DECLARE_CLASS(PlRcComputer );
  ```

  class PlRcComputer (-)

  Instances of PlRcComputer are multi-processors with a known configuration of associated disks.

- degrade

  ```
  virtual int degrade(const HEpochInterval& intvl, const char*
  listName);
  ```

  Create an instance of RDegState and add to the specified list or the degradation list if no list name is present. If the state was successfully created and added, TRUE will be returned. Otherwise FALSE will be returned. Overrides base class to propagate degradation to this computers cpus

- degrade

  ```
  virtual int degrade(RDegState* state, const char* listName);
  ```

  Use the given state to degrade me on the given list. If the state was successfully added, TRUE will be returned. Otherwise FALSE will be returned. Overrides base class to propagate degradation to this computers cpus

- get

  ```
  int get(istream& );
  ```

  Get myself from a stream

- localDisks

  ```
  virtual HObjIter* localDisks();
  ```

  Return an iterator over my LocalDisk list. The consumer must delete the iterator when done.

- numberOfCpus

  ```
  virtual int numberOfCpus(int nCpus);
  ```

  Set number of CPUs for this computer

- numberOfCpus

```
virtual int numberOfCpus() const;
```

Get number of CPUs for this computer

- operator =

```
PlRcComputer& operator =(const PlRcComputer& orig);
```

Assignment operator

- put

```
virtual int put(ostream& ) const;
```

Put myself onto a stream

- remDegrade

```
virtual int remDegrade(const HEpochInterval& intvl, const char*
listName);
```

Remove all degradation states that overlap the given time interval in the given list or the degradation list if no list name is present. Overrides base class to propagate degradation to this computers cpus

- removeCpu

```
virtual int removeCpu(RResource* cpu);
```

Remove a CPU from me.

- removeDisk

```
virtual int removeDisk(RResource* disk);
```

Remove a disk from me.

- resolveCpuIds

```
virtual void resolveCpuIds();
```

Try to resolve any CPU ids to CPUs from the resource pool. CPUs are passed around (ipc, to/from streams) using their reference resource id number. I may know a bunch of ids without yet having pointers to resources. This behavior will resolve any unknown ids by attempting to look them up in the RResourcePool.

- resolveLocalDiskIds

```
virtual void resolveLocalDiskIds();
```

Try to resolve Local Disk ids to disks in the same manner as for CPUs above.

- stringId

```
virtual int stringId() const;
```

Get ID of the string which contains this computer

- stringId

  `virtual int stringId(int id);`

  Set ID for the string which contains this computer

- xdr

  `int xdr(XDR* );`

  Get/put myself onto an XDR stream

6.3.9.3.3 PlRcCpu Class

Overview:

class PlRcCpu (-)

Instances of PlRcCpu are processors associated with a particular computer and are of a certain type (Sun, SGI, etc.)

Export Control: Public

Inheritance Relationships:

Inherits from PlRcResource

Attributes:

`myComputerId: int`

My computer id

`myComputer: PlRcComputer*`

My computer

`myCpuType: HString`

My CPU type

Constructors and Destructor:

`PlRcCpu();`

Default constructor

`PlRcCpu(const PlRcCpu& orig);`

Copy constructor

`virtual  ~PlRcCpu();`

Destructor

Operations:

- computerId

  ```
  virtual void computerId(const int id);
  ```

  Set my associated computer id

- computerId

  ```
  virtual int computerId();
  ```

  Get my associated computer id

- computer

  ```
  virtual int computer(PlRcComputer* computer);
  ```

  Set my associated computer name

- computer

  ```
  virtual PlRcComputer* computer();
  ```

  Get my associated computer name

- cpuType

  ```
  virtual const HString& cpuType() const;
  ```

  Get my associated cpu type

- cpuType

  ```
  virtual void cpuType(const HString& computerType);
  ```

  Set my associated cpu type

- DECLARE_CLASS

  ```
  int DECLARE_CLASS(PlRcCpu );
  ```

  class PlRcCpu (-)

  Instances of PlRcCpu are processors associated with a particular computer and are of a certain type (Sun, SGI, etc.)

- get

  ```
  int get(istream& );
  ```

  Get myself from a stream

- operator =

  ```
  PlRcCpu& operator =(const PlRcCpu& orig);
  ```

  Assignment operator

- put

  `int put(ostream& ) const;`

  Put myself onto a stream

- rsStateType

  `virtual HClassDesc& rsStateType();`

  Get the class description of my type of resource state.

- xdr

  `int xdr(XDR* );`

  Get/put myself onto an XDR stream

6.3.9.3.4 PlRcDataArrivalState Class

Overview:

class PlRcDataArrivalState (-)

Instances of PlRcDataArrivalState are states which represent data server subsystem activity.

Export Control: Public

Inheritance Relationships:

Inherits from PlRcActState

Attributes:

`myLabel: HString`

My label

Constructors and Destructor:

`PlRcDataArrivalState();`

Default constructor

`PlRcDataArrivalState(const PlRcDataArrivalState& orig);`

Copy constructor

`virtual ~PlRcDataArrivalState();`

Destructor

Operations:

- DECLARE_CLASS

```
int DECLARE_CLASS(PlRcDataArrivalState );
```

class PlRcDataArrivalState (-)

Instances of PlRcDataArrivalState are states which represent data server subsystem activity.

- get

```
virtual int get(istream& istr);
```

Get myself from a stream

- label

```
virtual void label(const char* );
```

Set my label.

- label

```
virtual const HString& label() const;
```

Get my label.

- operator =

```
PlRcDataArrivalState& operator =(const PlRcDataArrivalState&
orig);
```

Assignment operator

- put

```
virtual int put(ostream& ostr) const;
```

Put myself onto a stream

- xdr

```
virtual int xdr(XDR* xdrs);
```

Get/put myself onto an XDR stream

6.3.9.3.5 PlRcDataServer Class

Overview:

class PlRcDataServer (-)

Instances of PlRcDataServer represent the state of the data server subsystem.

Export Control: Public

Inheritance Relationships:

Inherits from PlRcResource

Attributes:

Constructors and Destructor:

```
PlRcDataServer();
```

Default constructor

```
PlRcDataServer(const PlRcDataServer& orig);
```

Copy constructor

```
virtual  ~PlRcDataServer();
```

Destructor

Operations:

- DECLARE_CLASS

```
int DECLARE_CLASS(PlRcDataServer );
```

class PlRcDataServer (-)

Instances of PlRcDataServer represent the state of the data server subsystem.

- get

```
virtual int get(istream& istr);
```

Get myself from a stream

- operator =

```
PlRcDataServer& operator =(const PlRcDataServer& orig);
```

Assignment operator

- put

```
virtual int put(ostream& ostr) const;
```

Put myself onto a stream

- xdr

```
virtual int xdr(XDR* xdrs);
```

Get/put myself onto an XDR stream

6.3.9.3.6 PlRcDisk Class

Overview:

class PlRcDisk (-)

Instances of PlRcDisk are computer resources capable of providing disk storage and disk capacity information.

Export Control: Public

Inheritance Relationships:

Inherits from PlRcResource

Attributes:

`myDiskCapacity: double`

Amount of total storage available on this disk

`myDiskStorage: double`

Amount of current storage used on this disk

Constructors and Destructor:

`PlRcDisk();`

Default constructor

`PlRcDisk(const PlRcDisk& orig);`

Copy constructor

`virtual  ~PlRcDisk();`

Destructor

Operations:

- DECLARE_CLASS

  `int DECLARE_CLASS(PlRcDisk );`

  class PlRcDisk (-)

  Instances of PlRcDisk are computer resources capable of providing disk storage and disk capacity information.

- diskCapacity

  `virtual int diskCapacity() const;`

  Get the disk capacity

- diskCapacity

  `virtual void diskCapacity(const double diskCapacity);`

  Set the disk capacity

- diskStorage

  ```
  virtual int diskStorage() const;
  ```

  Get the current disk storage

- diskStorage

  ```
  virtual void diskStorage(const double diskStorage);
  ```

  Set the current disk storage

- get

  ```
  int get(istream& );
  ```

  Get myself from a stream

- operator =

  ```
  PlRcDisk& operator =(const PlRcDisk& orig);
  ```

  Assignment operator

- put

  ```
  int put(ostream& ) const;
  ```

  Put myself onto a stream

- rsStateType

  ```
  virtual HClassDesc& rsStateType();
  ```

  Get the HClassDesc of the resource states belonging to a PlRcDisk

- xdr

  ```
  int xdr(XDR* );
  ```

  Get/put myself onto an XDR stream

6.3.9.3.7 PlRcForceLink Class

Overview:

class PlRcForceLink (force link for resource model classes)

Instances of PlRcForceLink are used to force linkage of resource model classes which may not be present at link time.

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
PlRcForceLink();
```

Default constructor

```
PlRcForceLink(const PlRcForceLink& orig);
```

Copy constructor

```
~PlRcForceLink();
```

Destructor

Operations:

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlRcForceLink );
```

class PlRcForceLink (force link for resource model classes)

Instances of PlRcForceLink are used to force linkage of resource model classes which may not be present at link time.

- linkage

```
void linkage();
```

Link stuff

- operator =

```
PlRcForceLink& operator =(const PlRcForceLink& orig);
```

Assignment operator

6.3.9.3.8 PlRcIdFactAbs Class

Overview:

class PlRcIdFactAbs (Unique abstract id factory)

Instances of PlRcIdFactAbs are abstract base class id generators. They provide the next allowable id that can be assigned to an object. This class has the notion of a high and low limits for ids that an id factory can generate.

Export Control: Public

Inheritance Relationships:

Attributes:

`myCurId: int`

My id to be assigned next.

`myMaxId: int`

The maximum id that can be assigned.

`myMinId: int`

The minimum id that can be assigned.

Constructors and Destructor:

`PlRcIdFactAbs();`

Default constructor

`PlRcIdFactAbs(const PlRcIdFactAbs& orig);`

Copy constructor, restricted

`virtual  ~PlRcIdFactAbs();`

Destructor

Operations:

- DECLARE_ABS_CLASS

  `int DECLARE_ABS_CLASS(PlRcIdFactAbs );`

  class PlRcIdFactAbs (Unique abstract id factory)

  Instances of PlRcIdFactAbs are abstract base class id generators. They provide the next allowable id that can be assigned to an object. This class has the notion of a high and low limits for ids that an id factory can generate.

- get

  `int get(istream& istr);`

  Get myself from a stream

- init

  `virtual int init() = 0;`

  Pure virtual initialize. Set myCurId, myMinId, and myMaxId to some initial values, Derived classes must override to init themselves according to other hereustics.

- maxId

  ```
  virtual int maxId() const;
  ```

  Get for maximum ID.

- maxId

  ```
  virtual int maxId(int id);
  ```

  Set for maximum ID.

- minId

  ```
  virtual int minId() const;
  ```

  Get for minimum ID.

- minId

  ```
  virtual int minId(int id);
  ```

  Set for minimum ID.

- nextIdLocal

  ```
  virtual int nextIdLocal();
  ```

  Return the next available ID. If we run off the end of our legal range, then return -1.

- operator =

  ```
  PlRcIdFactAbs& operator =(const PlRcIdFactAbs& orig);
  ```

  Assignment operator, restricted

- put

  ```
  int put(ostream& ostr) const;
  ```

  Put myself onto a stream

- xdr

  ```
  int xdr(XDR* xdrs);
  ```

  Get/put myself onto an xdr stream

6.3.9.3.9 PlRcLocalDisk Class

Overview:

class PlRcLocalDisk (-)

Instances of PlRcLocalDisk are Disk storage units which are "Locally" or directly attached to a Computer and are therefore owned by that computer.

Export Control: Public

Inheritance Relationships:

Inherits from PlRcDisk

Attributes:

`myComputerId: int`

My computer id

`myComputer: PlRcComputer*`

My computer

Constructors and Destructor:

`PlRcLocalDisk();`

Default constructor

`PlRcLocalDisk(const PlRcLocalDisk& orig);`

Copy constructor

`virtual  ~PlRcLocalDisk();`

Destructor

Operations:

- computerId

  `virtual void computerId(const int id);`

  Set my associated computer id

- computerId

  `virtual int computerId();`

  Get my associated computer id

- computer

  `virtual PlRcComputer* computer();`

  Get my associated computer name

- computer

  `virtual int computer(PlRcComputer* computer);`

  Set my associated computer name

- DECLARE_CLASS

  ```
  int DECLARE_CLASS(PlRcLocalDisk );
  ```

  class PlRcLocalDisk (-)

  Instances of PlRcLocalDisk are Disk storage units which are "Locally" or directly attached to a Computer and are therefore owned by that computer.

- get

  ```
  int get(istream& );
  ```

  Get myself from a stream

- operator =

  ```
  PlRcLocalDisk& operator =(const PlRcLocalDisk& orig);
  ```

  Assignment operator

- put

  ```
  int put(ostream& ) const;
  ```

  Put myself onto a stream

- xdr

  ```
  int xdr(XDR* );
  ```

  Get/put myself onto an XDR stream

6.3.9.3.10 PlRcMultiActState Class

Overview:

class PlRcMultiActState (resource state for multiple activities)

This class is a RRsState that can keep track of associations between it and multiple RActivity 'tasking' requests.

Export Control: Public

Inheritance Relationships:

Attributes:

```
myActivities: HVList
```

My list of activity ids

Constructors and Destructor:

```
PlRcMultiActState(const HEpochInterval& orig);
```

Constructor with specific time interval

```
PlRcMultiActState(const PlRcMultiActState& orig);
```

Copy Constructor

```
PlRcMultiActState();
```

Default constructor

```
virtual  ~PlRcMultiActState();
```

Destructor

Operations:

- activities

```
virtual HVIter* activities();
```

Access to my allocation activities Consumer responsible for deleting iter.

- addId

```
virtual int addId(int id);
```

Add an activity allocation, by activity id

- anyId

```
virtual int anyId() const;
```

Returns TRUE/FALSE if there are any activity ids within me

- containsId

```
int containsId(int id);
```

Returns TRUE/FALSE if this activity id is contained within me.

- copyOver

```
virtual int copyOver(const PlRcMultiActState& );
```

Create new copies of allocation ids in the source list

- DECLARE_CLASS

```
int DECLARE_CLASS(PlRcMultiActState );
```

class PlRcMultiActState (resource state for multiple activities)

This class is a RRsState that can keep track of associations between it and multiple RActivity 'tasking' requests.

- get

  `virtual int get(istream& istr);`

  Get myself from a stream

- operator =

  `PlRcMultiActState& operator =(const PlRcMultiActState& orig);`

  Assignment operator

- put

  `virtual int put(ostream& ostr) const;`

  Put myself onto a stream

- removeId

  `virtual int removeId(int id);`

  Remove an activity allocation, by activity id

- xdr

  `virtual int xdr(XDR* xdrs);`

  Get/put myself onto an XDR stream

6.3.9.3.11 PlRcProcessState Class

Overview:

class PlRcProcessState (-)

Instances of PlRcProcessState are classes that represent the current processing state of the CPU.

Export Control: Public

Inheritance Relationships:

Inherits from PlRcActState

Attributes:

`myColor: HString`

My color

`myDprStatus: enum PlRcProcessState::DprStatus`

My DPR processing status

`myPrName: HString`

My production request name

Constructors and Destructor:

```
PlRcProcessState(const PlRcProcessState& orig);
```

Copy constructor

```
PlRcProcessState(const HEpochInterval& , int actId);
```

Constructor with interval specified

```
PlRcProcessState(const HEpochInterval& intvl, int actId, const
char* name);
```

Constructor with interval, acitivity id and name specified

```
PlRcProcessState();
```

Default constructor

```
virtual  ~PlRcProcessState();
```

Destructor

Operations:

- color

```
virtual void color(const char* );
```

Set my color

- color

```
virtual const HString& color() const;
```

Get my color

- DECLARE_CLASS

```
int DECLARE_CLASS(PlRcProcessState );
```

class PlRcProcessState (-)

Instances of PlRcProcessState are classes that represent the current processing state of the CPU.

- dprStatusAsString

```
virtual const char* dprStatusAsString() const;
```

Get the DPR processing status as a string.

- dprStatus

```
virtual enum PlRcProcessState::DprStatus dprStatus() const;
```

Get the DPR processing status

- dprStatus

```
virtual int dprStatus(int processingStatus);
```

Set the DPR processing status

- dprStatus

```
virtual void dprStatus(enum PlRcProcessState::DprStatus
processingStatus);
```

Set the DPR processing status

- get

```
int get(istream& );
```

Get myself from a stream

- operator =

```
PlRcProcessState& operator =(const PlRcProcessState& orig);
```

Assignment operator

- prName

```
virtual void prName(const char* );
```

Set the name of production request

- prName

```
virtual const HString& prName() const;
```

Get the name of production request

- put

```
int put(ostream& ) const;
```

Put myself onto a stream

- xdr

```
int xdr(XDR* );
```

Get/put myself onto an XDR stream

6.3.9.3.12 PlRcResourceIdFact Class

Overview:

class PlRcResourceIdFact (Unique activity id factory)

Instances of PlRcResourceIdFact are activity ID generators. You can ask them for the next allowable activity ID that can be assigned. This class initializes its activity ID range from the current activity pool for now. Max ID is presumed to be MAXINT.

Export Control: Public

Inheritance Relationships:

Inherits from PlRcIdFactAbs

Attributes:

`ourGlobalResourceIdFact: PlRcResourceIdFact*`

Current global ID.

Constructors and Destructor:

`PlRcResourceIdFact();`

Default constructor

`PlRcResourceIdFact(const PlRcResourceIdFact& orig);`

Copy constructor, restricted copy

`virtual ~PlRcResourceIdFact();`

Destructor

Operations:

- DECLARE_CLASS

  `int DECLARE_CLASS(PlRcResourceIdFact );`

  class PlRcResourceIdFact (Unique activity id factory)

  Instances of PlRcResourceIdFact are activity ID generators. You can ask them for the next allowable activity ID that can be assigned. This class initializes its activity ID range from the current activity pool for now. Max ID is presumed to be MAXINT.

- init

  `virtual int init();`

  Initialize. Set myCurId, myMinId, and myMaxId to some initial values, based upon the contents of the current activity pool. Max ID value is presumed to be MAXINT.

- nextResourceId

  `static int nextResourceId();`

  Return the next available ID in the global PlRcResourceIdFact.

- operator =

  `PlRcResourceIdFact& operator =(const PlRcResourceIdFact& orig);`

  Assignment operator, restricted assignment

- theResourceIdFact

  ```
  static PlRcResourceIdFact* theResourceIdFact();
  ```

  Get the global PlRcResourceIdFact.

- theResourceIdFact

  ```
  static int theResourceIdFact(PlRcResourceIdFact* idFactory);
  ```

  Set the global PlRcResourceIdFact. This class does not assume ownership.

6.3.9.3.13 PlRcResource Class

Overview:

class PlRcResource (-)

PlRcResource is a base class for all planning resources

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
PlRcResource();
```

Default onstructor

```
PlRcResource(const PlRcResource& orig);
```

Copy constructor

```
virtual  ~PlRcResource();
```

Destructor

Operations:

- DECLARE_CLASS

  ```
  int DECLARE_CLASS(PlRcResource );
  ```

  class PlRcResource (-)

  PlRcResource is a base class for all planning resources

- operator =

```
PlRcResource& operator =(const PlRcResource& orig);
```

assignment operator

- rsStateType

```
virtual HClassDesc& rsStateType();
```

Get the class description of my type of resource state. Default is to just return generic RRsState type. derived classes should override to return a specialized type

6.3.9.3.14 PlRcState Class

Overview:

class PlRcState (generic resource state)

Instances of PlRcState are generic activity states which simply contain the name of the generic state

Export Control: Public

Inheritance Relationships:

Attributes:

```
myName: HString
```

My name

Constructors and Destructor:

```
PlRcState(const HEpochInterval& intvl, int actId, const char*
name);
```

Constructor with interval, acitivity id and name specified

```
PlRcState(const PlRcState& orig);
```

Copy constructor

```
PlRcState(const HEpochInterval& intvl, int actId);
```

Constructor with interval and activity id

```
PlRcState();
```

Default constructor

```
virtual  ~PlRcState();
```

Destructor

Operations:

- DECLARE_CLASS

  ```
  int DECLARE_CLASS(PlRcState );
  ```

  class PlRcState (generic resource state)

  Instances of PlRcState are generic activity states which simply contain the name of the generic state

- get

  ```
  virtual int get(istream& istr);
  ```

  Get myself from a stream

- name

  ```
  virtual int name(const char* name);
  ```

  Set my name

- name

  ```
  virtual const HString& name() const;
  ```

  Get my name

- operator =

  ```
  PlRcState& operator =(const PlRcState& orig);
  ```

  Assignment operator

- put

  ```
  virtual int put(ostream& ostr) const;
  ```

  Put myself onto a stream

- xdr

  ```
  virtual int xdr(XDR* xdrs);
  ```

  Get/put myself onto an XDR stream

6.3.9.3.15 PlRcStorageState Class

Overview:

class PlRcStorageState

Instances of PlRcStorageState are states for storage utilization for a disk.

Export Control: Public

Inheritance Relationships:

Inherits from PlRcMultiActState

Attributes:

```
myStorage: int
```

The storage consumed over my time interval

Constructors and Destructor:

```
PlRcStorageState(const PlRcStorageState& orig);
```

Copy constructor

```
PlRcStorageState(const HEpochInterval& );
```

Constructor with interval specified

```
PlRcStorageState();
```

Default constructor

```
virtual  ~PlRcStorageState();
```

Destructor

Operations:

- DECLARE_CLASS

```
int DECLARE_CLASS(PlRcStorageState );
```

class PlRcStorageState

Instances of PlRcStorageState are states for storage utilization for a disk.

- get

```
int get(istream& );
```

Get myself from a stream

- operator =

```
PlRcStorageState& operator =(const PlRcStorageState& orig);
```

Assignment operator

- put

```
int put(ostream& ) const;
```

Put myself onto a stream

- storage

`virtual int storage(int newStorage);`

Set the total storage consumed

- storage

`virtual int storage() const;`

Get the total storage consumed

- xdr

`int xdr(XDR* );`

Get/put myself onto an XDR stream

6.3.9.3.16 PlRcString Class

Overview:

Export Control: Public

Inheritance Relationships:

Inherits from PlRcResource

Attributes:

`myComputerIds: HVList`

My list of Computer ids corresponding to the above list of Computers. I may have ids without having Computers if I have just been loaded from a stream (see resolveIds() below).

`myComputers: HObjList`

My list of associated Computers with this PlRcString

Constructors and Destructor:

`PlRcString();`

Default constructor

`PlRcString(const PlRcString& orig);`

Copy constructor

`virtual ~PlRcString();`

Destructor

Operations:

- addAssoc

  ```
  virtual int addAssoc(HObject* rs);
  ```

  Add an association to a resource.

- addComputer

  ```
  virtual int addComputer(RResource* computer);
  ```

  Add a Computer to me.

- computers

  ```
  virtual HObjIter* computers();
  ```

  Return an iterator over my Computer list. The consumer must delete the iterator when done.

- copyOver

  ```
  virtual int copyOver(const PlRcString& orig);
  ```

  Copy over the internals from a PlRcString

- copyOver

  ```
  virtual int copyOver(const RResource& orig);
  ```

  Copy over the internals from a PlRcString

- DECLARE_CLASS

  ```
  int DECLARE_CLASS(PlRcString );
  ```

- get

  ```
  int get(istream& );
  ```

  Get myself from a stream

- operator =

  ```
  PlRcString& operator =(const PlRcString& orig);
  ```

  Assignment operator

- put

  ```
  int put(ostream& ) const;
  ```

  Put myself onto a stream

- removeComputer

```
virtual int removeComputer(RResource* computer);
```

Remove a Computer from me.

- resolveComputerIds

```
virtual void resolveComputerIds();
```

Try to resolve any Computer ids to Computers from the resource pool. Computers are passed around (ipc, to/from streams) using their reference resource id number. I may know a bunch of ids without yet having pointers to resources. This behavior will resolve any unknown ids by attempting to look them up in the RResourcePool.

- xdr

```
int xdr(XDR* );
```

Get/put myself onto an XDR stream

6.3.10 Planning_Workbench_Rm Class Category

6.3.10.1 Overview

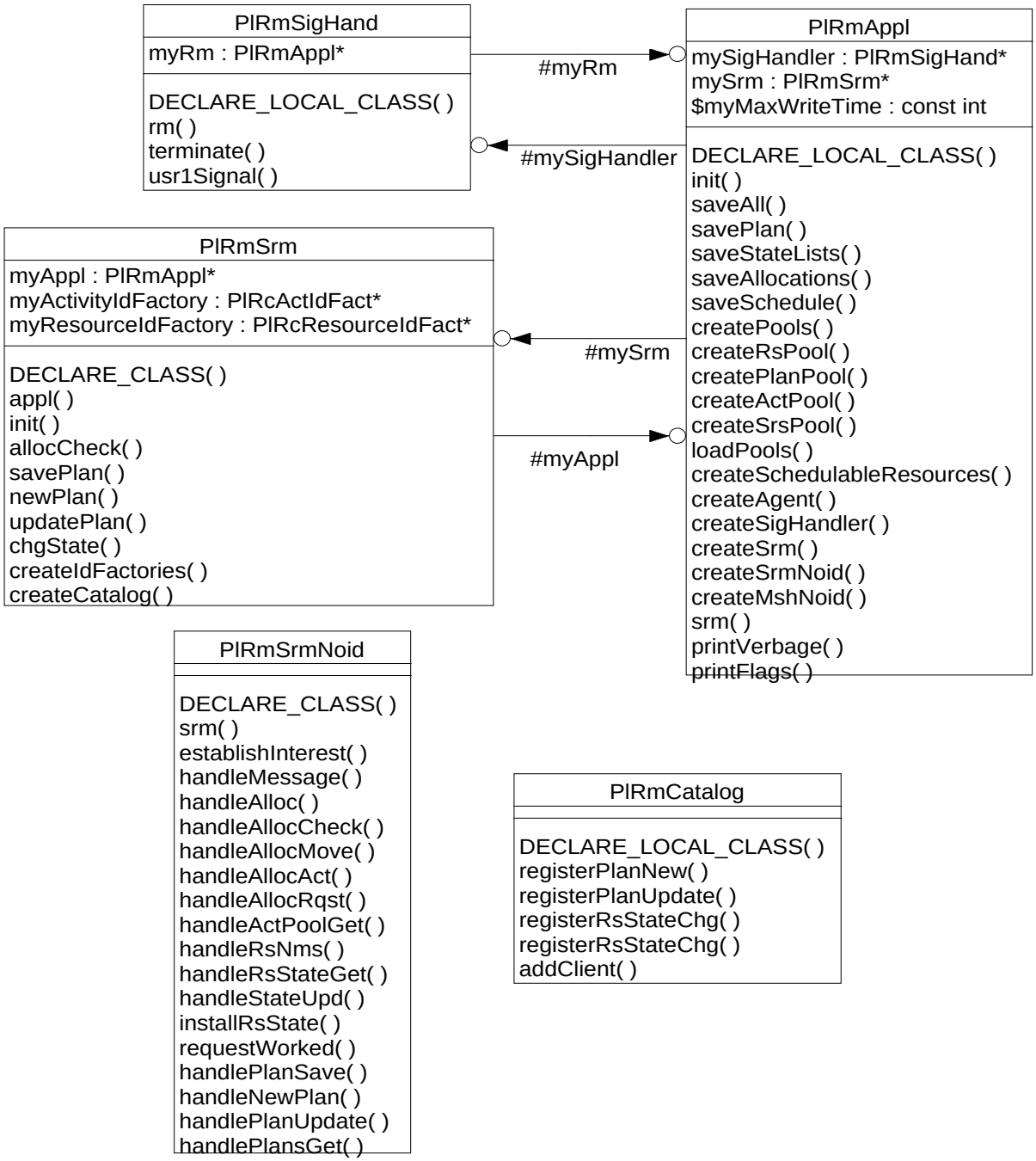


Figure 6.3.10.1-1 Planning_Workbench_Rm

6.3.10.2 Planning_Workbench_Rm Classes

6.3.10.3 PlRmAppl Class

Overview:

class PlRmAppl (resource model application)

Instances of PlRmAppl are resource model applications. PlRmAppl is responsible for creating and/or connecting to the name server, message agent, resource and activity pools, and scheduling noids. PlRmAppl is also responsible for settting up the notifier for running the resource model application.

Export Control: Public

Inheritance Relationships:

Inherits from PlMiAppl

Attributes:

`myMaxWriteTime: const int`

My IPC write timeout time

`mySigHandler: PlRmSigHand*`

My signal handler

`mySrm: PlRmSrm*`

My resource model workhorse to handle client requests

Constructors and Destructor:

`PlRmAppl();`

Default constructor

`PlRmAppl(const PlRmAppl& orig);`

Copy constructor

`virtual ~PlRmAppl();`

Destructor

Operations:

- createActPool

`virtual int createActPool();`

Create the activity pool

- createAgent

  `virtual int createAgent();`

  Create an instance of a HMsgAgent. Overrides PlMiAppl::createAgent to set the connection's maximum write time

- createMshNoid

  `virtual int createMshNoid();`

  Create a connection to the message handler.

- createPlanPool

  `virtual int createPlanPool();`

  Create the plan pool

- createPools

  `virtual int createPools();`

  Create the pools that I need

- createRsPool

  `virtual int createRsPool();`

  Create the resource pool

- createSchedulableResources

  `virtual int createSchedulableResources() const;`

  Create new schedulable resources (PlRcResource) and add them to the schedulable resource pool.

- createSigHandler

  `virtual int createSigHandler();`

  Create a signal handler who will call me when certain signals are received

- createSrmNoid

  `virtual int createSrmNoid();`

  Create my message noid.

- createSrm

  `virtual int createSrm();`

  Create my resource model main class. This is an instance of an PlRmSrm object which has scheduling behavior in response to external input.

- createSrsPool

```
virtual int createSrsPool();
```

Create the SResource pool

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlRmAppl );
```

class PlRmAppl (resource model application)

Instances of PlRmAppl are resource model applications. PlRmAppl is responsible for creating and/or connecting to the name server, message agent, resource and activity pools, and scheduling noids. PlRmAppl is also responsible for settting up the notifier for running the resource model application.

- init

```
virtual int init();
```

Initialize this application. Overrides PlMiAppl::init to initialize those things I need.

- loadPools

```
virtual int loadPools() const;
```

Fill the various pools with their static and dynamic states from a database.

- operator =

```
PlRmAppl& operator =(const PlRmAppl& orig);
```

Assignment operator

- printFlags

```
virtual void printFlags();
```

Dump out a description of the run-time flags required to run this application. Overrides PlMiAppl::printFlags to print out more flags needed by me

- printVerbage

```
virtual void printVerbage();
```

Dump out a usage message as to why this appl exists. Overrides PlMiAppl::printVerbage to print out what I am/do

- saveAllocations

```
virtual int saveAllocations(const char* plan, const
HEpochInterval& interval, const HVList* rsIds);
```

Save the activity allocations given the resources, plan and time interval. If no resources are specified, all resources are saved. If no plan is specified, all plans are saved, if no interval is set, the entire time span is saved.

- saveAll

```
virtual int saveAll();
```

Save all my state to files

- savePlan

```
virtual int savePlan(PlPlPlan* plan);
```

Save a plan to a file. Note: If the plan is NULL, all plans in the resource model will be saved

- savePlan

```
virtual int savePlan(const char* plan, const HEpochInterval&
interval, const HVList* rsIds);
```

Save my state to a file for the given plan over the given interval for the given set of resources. If any parameters are NULL I will save all that I contain for that parameter.

- saveSchedule

```
virtual int saveSchedule(const char* plan, const HEpochInterval&
interval, const HVList* rsIds);
```

Save the schedule in a human readable form to a file. If no resources are specified, all resources are saved. If no plan is specified, all plans are saved, if no interval is set, the entire time span is saved.

- saveStateLists

```
virtual int saveStateLists(const char* plan, const
HEpochInterval& interval, const HVList* rsIds);
```

Save the resource states given the resources, plan and time interval to a file. If no resources are specified, all resources are saved. If no plan is specified, all plans are saved, if no interval is set, the entire time span is saved.

- srm

```
virtual PlRmSrm* srm() const;
```

Get my resource model

- srm

```
virtual int srm(PlRmSrm* anSrm);
```

Set my resource model

6.3.10.3.1 PlRmCatalog Class

Overview:

class PlRmCatalog (PlRmSrm client catalog)

An instance of PlRmCatalog is owned by a PlRmSrm client. PlRmCatalog adds additional behaviour to the SrmCatalog to handle registration and notification specific to PlRmSrm clients.

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
PlRmCatalog();
```

Default constructor

```
PlRmCatalog(const PlRmCatalog& orig);
```

Copy constructor

```
virtual  ~PlRmCatalog();
```

Destructor

Operations:

- addClient

```
virtual int addClient(HAddress* cliAdr);
```

Add a client to me. I will clone the passed-in HAddress. overridden from base class to make sure I add an PlRmCliEntry.

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlRmCatalog );
```

class PlRmCatalog (PlRmSrm client catalog)

An instance of PlRmCatalog is owned by a PlRmSrm client. PlRmCatalog adds additional behaviour to the SrmCatalog to handle registration and notification specific to PlRmSrm clients.

- operator =

```
PlRmCatalog& operator =(const PlRmCatalog& orig);
```

Assignment operator

- registerPlanNew

```
virtual int registerPlanNew(const char* planName);
```

Register new plan. Defer to base class. The following two functions had to be overridden to avoid hiding them, since the function has been overloaded in this class.

- registerPlanNew

```
virtual int registerPlanNew(PlPlPlan* plan);
```

Register new plan. Send out a notice to any clients interested in plan names.

- registerPlanNew

```
virtual int registerPlanNew(PlPlPlan* plan, HAddress* fromWhom);
```

Register new plan.

- registerPlanNew

```
virtual int registerPlanNew(const char* planName, HAddress*
fromWhom);
```

Register new plan. Defer to base class. The following function had to be overridden to

- registerPlanUpdate

```
virtual int registerPlanUpdate(PlPlPlan* plan);
```

Register updated plan. Send out a notice to any clients interested in plan names.

- registerPlanUpdate

```
virtual int registerPlanUpdate(PlPlPlan* plan, HAddress*
fromWhom);
```

Register updated plan.

- registerRsStateChg

```
virtual int registerRsStateChg(int rsId, const char* planName,
const HEpochInterval& when, HAddress* fromWhom);
```

- registerRsStateChg

```
virtual int registerRsStateChg(RUpdState* upd);
```

Register change in a resource state. Overridden from base class to register update notice for srm clients.

- registerRsStateChg

```
virtual int registerRsStateChg(int rsId, const char* planName,
const HEpochInterval& when);
```

6.3.10.3.2 PlRmSigHand Class

Overview:

class PlRmSigHand (resource model signal handler)

Instances of PlRmSigHand are signal catchers for a resource model to detect termination signals and save our resource model states.

Export Control: Public

Inheritance Relationships:

Attributes:

`myRm: PlRmAppl*`

My resource model application class who I contact when I receive a signal

Constructors and Destructor:

`PlRmSigHand();`

Default constructor

`PlRmSigHand(const PlRmSigHand& orig);`

Copy constructor

`virtual ~PlRmSigHand();`

Destructor

Operations:

- DECLARE_LOCAL_CLASS

  `int DECLARE_LOCAL_CLASS(PlRmSigHand );`

  class PlRmSigHand (resource model signal handler)

  Instances of PlRmSigHand are signal catchers for a resource model to detect termination signals and save our resource model states.

- operator =

  `PlRmSigHand& operator =(const PlRmSigHand& orig);`

  Assignment operator

- rm

  `virtual PlRmAppl* rm() const;`

Get my resource model

- rm

```
virtual int rm(PlRmAppl* anRm);
```

Set my resource model

- terminate

```
void terminate(int sig);
```

This member can be used to do final cleanup before a program terminates.

- usr1Signal

```
virtual void usr1Signal(int sig);
```

Usr signal 1, override base class behavior to ask my resource model to save its state

6.3.10.3.3 PlRmSrmNoid Class

Overview:

Class PlRmSrmNoid (access to Srm services via IPC messages)

The PlRmSrmNoid is an IPC msgAgent that services resource model requests. These requests include allocation of activities, adding and removing resources, and requesting information about the resources.

Additional message handling behavior for new messages is added including handling of oversubscription, plans, etc.

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
PlRmSrmNoid();
```

Default constructor.

```
PlRmSrmNoid(const PlRmSrmNoid& noid);
```

Copy constructor

```
virtual  ~PlRmSrmNoid();
```

Destructor.

Operations:

- DECLARE_CLASS

  ```
  int DECLARE_CLASS(PlRmSrmNoid );
  ```

  Class PlRmSrmNoid (access to Srm services via IPC messages)

  The PlRmSrmNoid is an IPC msgAgent that services resource model requests. These requests include allocation of activities, adding and removing resources, and requesting information about the resources.

  Additional message handling behavior for new messages is added including handling of oversubscription, plans, etc.

- establishInterest

  ```
  virtual int establishInterest();
  ```

  Establish interest with my agent in all resource model messages

- handleActPoolGet

  ```
  virtual int handleActPoolGet(HMsgConn* curCon, HAddress* curAdr,
  PlClMsgGetActPool* curMsg);
  ```

  Handle a message which requests a copy of the current activity pool.

- handleAllocAct

  ```
  virtual int handleAllocAct(HMsgConn* curCon, HAddress* curAdr,
  SMsgAllocAct* curMsg);
  ```

  Handle a message to do an allocation with constraint checking and a supplied RActivity. Overridden to see if we have the derived message type that knows about oversubscription of messages. If so, we set the Srm's derived ecs SaAllocator accordingly.

- handleAllocCheck

  ```
  virtual int handleAllocCheck(HMsgConn* conn, HAddress* adr,
  SMsgAllocCheck* msg);
  ```

  Handle an inbound message instructing me to check an allocation. Overridden to see if we have the derived message type that knows about oversubscription of messages. If so, we pass that information along to the PlRmSrm so that it can do the right thing.

- handleAllocMove

  ```
  virtual int handleAllocMove(HMsgConn* conn, HAddress* adr,
  SMsgAllocMove* msg);
  ```

Handle an inbound message instructing me to move an allocation. Overridden to see if we have the derived message type that knows about oversubscription of messages. If so, we set the Srm's derived ecs SaAllocator accordingly.

- handleAllocRqst

```
virtual int handleAllocRqst(HMsgConn* curCon, HAddress* curAdr,
SMsgAllocRqst* curMsg);
```

Handle a message to do an allocation with constraint checking, and a collection of RSchRqsts. Overridden to see if we have the derived message type that knows about oversubscription of messages. If so, we set the Srm's derived ecs SaAllocator accordingly.

- handleAlloc

```
virtual int handleAlloc(HMsgConn* conn, HAddress* adr,
SMsgAlloc* msg);
```

Handle an inbound message instructing me to do an allocation. Overridden to see if we have the derived message type that knows about oversubscription of messages. If so, we set the Srm's derived ecs SaAllocator accordingly.

- handleMessage

```
virtual int handleMessage(HMsgConn* curCon, HAddress* curAdr,
HMessage* curMsg);
```

Handle incoming message.

- handleNewPlan

```
virtual int handleNewPlan(HMsgConn* curCon, HAddress* curAdr,
PlClMsgPlanNew* curMsg);
```

Handle a message which requests plans. This is similar to my base SrmNoid::handlePlanNew

- handlePlanSave

```
virtual int handlePlanSave(HMsgConn* curCon, HAddress* curAdr,
PlClMsgPlanSave* curMsg);
```

Handle a message which requests a plan to be saved to a file.

- handlePlansGet

```
virtual int handlePlansGet(HMsgConn* curCon, HAddress* curAdr,
PlClMsgGetPlans* curMsg);
```

Handle a message which requests plans.

- handlePlanUpdate

```
virtual int handlePlanUpdate(HMsgConn* curCon, HAddress* curAdr,
PlClMsgPlanUpdate* curMsg);
```

Handle a message which requests an update on a plan.

- handleRsNms

```
virtual int handleRsNms(HMsgConn* curCon, HAddress* curAdr,
PlClMsgGetRsNms* curMsg);
```

Handle a message which requests a list of all the resource names that the srm knows about.

- handleRsStateGet

```
virtual int handleRsStateGet(HMsgConn* curCon, HAddress* curAdr,
SMsgRsStateGet* curMsg);
```

Handle a message to send all resource state to the current client.

- handleStateUpd

```
virtual int handleStateUpd(HMsgConn* curCon, HAddress* curAdr,
SMsgStateUpd* curMsg);
```

Handle a message containing information about a resource state update. The default behavior is to extract out from the message the contained RUpdState and pass it along to installRsState().

- installRsState

```
virtual int installRsState(RUpdState* anUpdate);
```

Take the RRsStateList contained within the provided RUpdState, and edit the state list for the appropriate Resource. Ownership of the passed in RUpdState and its components is assumed by this behavior.

- operator =

```
PlRmSrmNoid& operator =(const PlRmSrmNoid& noid);
```

Restrict access to assignment.

- requestWorked

```
virtual int requestWorked(SMsgStat* msg);
```

Examine a returned SMsgStat and return if it's status indicates success or failure, by returning TRUE/FALSE, respectively.

- srm

```
virtual int srm(Srm* aSrm);
```

Set my visibility to an Srm. I do not assume ownership. Overridden from base class to ensure that I am actually associated with an PlRmSrm.

- srm

```
virtual Srm* srm() const;
```

Get my visibility to an Srm.

6.3.10.3.4 PlRmSrm Class

Overview:

class PlRmSrm (system resource model)

Instances of PlRmSrm are resource model editors that know about ecs notions, plan accesses, etc.

Export Control: Public

Inheritance Relationships:

Attributes:

`myActivityIdFactory: PlRcActIdFact*`

My activity id factory

`myAppl: PlRmAppl*`

My parent application

`myResourceIdFactory: PlRcResourceIdFact*`

My resource id factory

Constructors and Destructor:

`PlRmSrm();`

Default constructor

`PlRmSrm(const PlRmSrm& orig);`

Copy constructor, restricted copy

`virtual ~PlRmSrm();`

Destructor

Operations:

- allocCheck

`virtual int allocCheck(const char* planName, const HEpochInterval& intvl, int rsId, int actId);`

Check allocation (do constraint-checking, but do not allocate). This determines if an allocation COULD be performed. The default is to not permit oversubscription.

- allocCheck

  ```
  virtual int allocCheck(const char* planName, const
  HEpochInterval& intvl, int rsId, int actId, int ovrsAllowed);
  ```

  Check allocation (do constraint-checking, but do not allocate). This determines if an allocation COULD be performed, allowing the caller to specify if oversubscription is permitted or not. The specified resource is checked to see if it is a PlScResource type, which knows about oversubscription issues. If so, then it is passed the oversubscription flag. If not, then the SResource checkAll is used.

- appl

  ```
  virtual const PlRmAppl* appl() const;
  ```

  Get my parent application

- appl

  ```
  virtual int appl(PlRmAppl* anAppl);
  ```

  Set my parent application

- chgState

  ```
  virtual int chgState(RUpdState* anUpdate);
  ```

  A client has sent a request to update a resource state.

- createCatalog

  ```
  virtual int createCatalog();
  ```

  Create and initialize my client catalog

- createIdFactories

  ```
  virtual int createIdFactories();
  ```

  Create the id generator factories

- DECLARE_CLASS

  ```
  int DECLARE_CLASS(PlRmSrm );
  ```

  class PlRmSrm (system resource model)

  Instances of PlRmSrm are resource model editors that know about ecs notions, plan accesses, etc.

- init

  ```
  virtual int init();
  ```

  Initialize

- newPlan

```
virtual int newPlan(HAddress* fromAdr, const char* planName,
HVList* rsIds);
```

A client has sent a request to create a new plan given the name of the plan only. Overrides Srm::newPlan to not hide newPlan(). I do not assume ownership to the passed in HAddress, which may be NULL.

- newPlan

```
virtual int newPlan(HAddress* fromAdr, PlPlPlan* newPlan,
HVList* rsIds);
```

A client has sent a request to create a new derived plan I do not assume ownership to the passed in HAddress, which may be NULL. I will only create a new state list on those resources whose id is in the passed in list.

- operator =

```
PlRmSrm& operator =(const PlRmSrm& orig);
```

Assignment operator

- savePlan

```
virtual int savePlan(const char* plan, const HEpochInterval&
interval, HVList* rsIds);
```

Save a plan's states and allocations. Portions of the plan may be saved by specifying the resources to save on a specif plan over a specific interval of time. If the collection is empty all resource states and allocations associated with those states are saved. If the plan is not specified, all plans are saved. If the interval is not set than the entire time span is saved

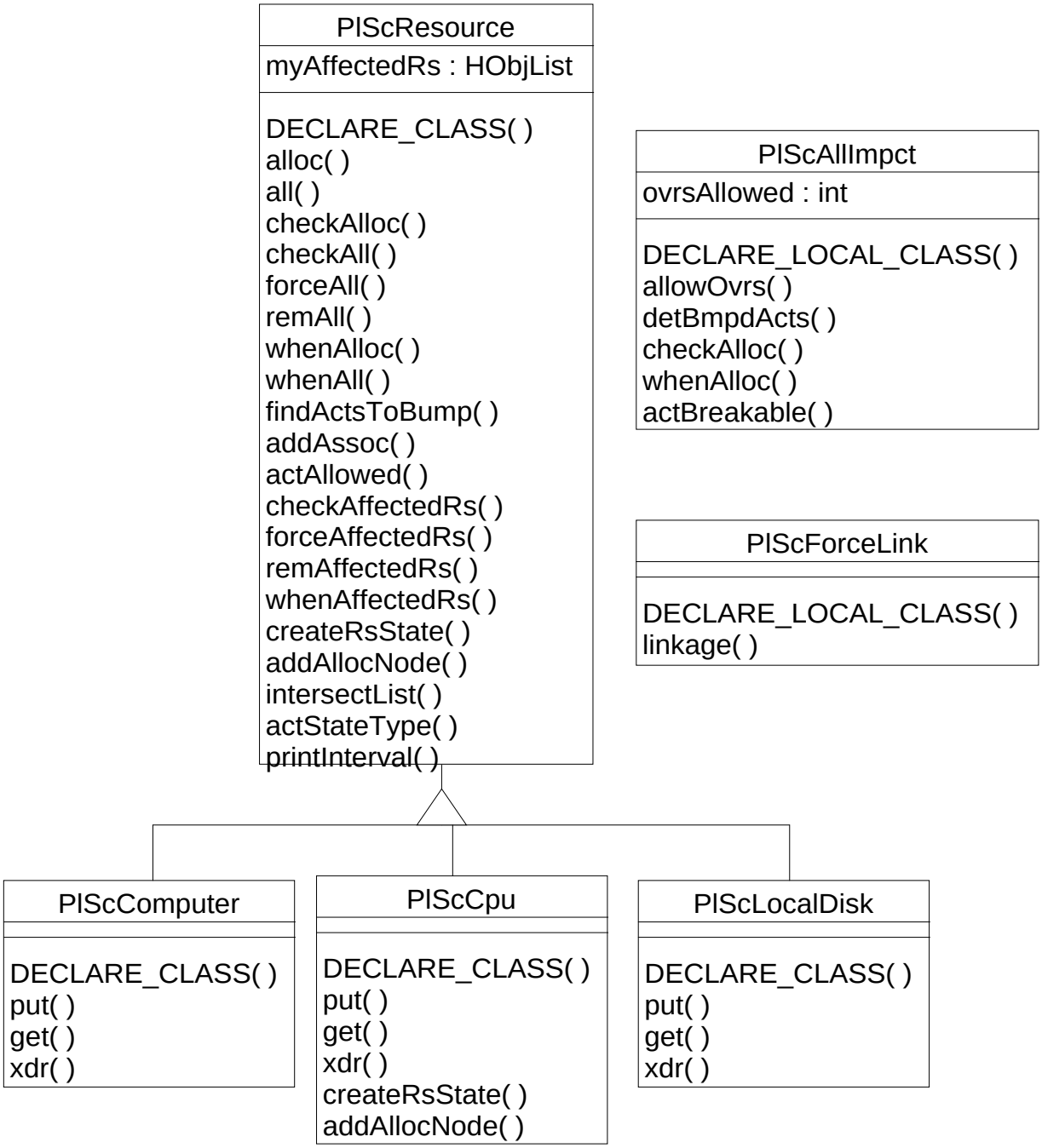
- updatePlan

```
virtual int updatePlan(HAddress* fromAdr, PlPlPlan*
updatedPlan);
```

A client has sent a request to update an old derived plan.

6.3.11 Planning_Workbench_Sc Class Category

6.3.11.1 Overview



6.3.11.2 Planning_Workbench_Sc Classes

6.3.11.3 PlScAllImpct Class

Overview:

class PlScAllImpct (Impact Scheduler)

Instances of PlScAllImpct add the knowledge of oversubscription to the impact scheduler. They also prevent any activity from being broken.

Export Control: Public

Inheritance Relationships:

Attributes:

`ovrsAllowed: int`

The oversubscription allowed flag

Constructors and Destructor:

`PlScAllImpct();`

Default constructor

`PlScAllImpct(const PlScAllImpct& orig);`

Copy constructor

`virtual ~PlScAllImpct();`

Destructor

Operations:

- actBreakable

  `virtual int actBreakable(RSimpleAct* act);`

  Return TRUE if the act is breakable, FALSE if not. If the activity is a GenericAct, check its breakable flag. Overrides SaAllImpct::actBreakable

- allowOvrs

  `virtual int allowOvrs() const;`

  Get oversubsription allowed flag

- allowOvrs

```
virtual void allowOvrs(int );
```

Set oversubsription allowed flag

- checkAlloc

```
virtual int checkAlloc(RSimpleAct* act, RSchOpp* opp);
```

Return TRUE if the given activity can be allocated as specified in the given opportunity. If the resource involved is an PlScResource, the checkAlloc behavior with the oversubscription flag is called. overrides SaAllImpct::checkAlloc

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlScAllImpct );
```

class PlScAllImpct (Impact Scheduler)

Instances of PlScAllImpct add the knowledge of oversubscription to the impact scheduler. They also prevent any activity from being broken.

- detBmpdActs

```
virtual int detBmpdActs(SResource* sRs, const HEpochInterval&
reqIntvl, HObjCollection& bmpdActs);
```

Determine the activities to be bumped. Overrides SaAllImpct::detBmpdActs()

- operator =

```
PlScAllImpct& operator =(const PlScAllImpct& orig);
```

Assignment operator

- whenAlloc

```
virtual int whenAlloc(RSimpleAct* act, RSchOpp* opp,
HObjCollection& col);
```

Determine when the given activity can be allocated on the plan within the given interval, according to the specifications in the given opportunity. Return allowable intervals in col. If the resource involved is an PlScResource, the whenAlloc behavior with the oversubscription flag is called. overrides SaAllImpct::whenAlloc

6.3.11.3.1 PlScComputer Class

Overview:

class PlScComputer (-)

Instances of PlScComputer are schedulable computer resources.

Export Control: Public

Inheritance Relationships:

Inherits from PlScResource

Attributes:

Constructors and Destructor:

```
PlScComputer();
```

Default constructor

```
PlScComputer(const PlScComputer& orig);
```

Copy constructor

```
virtual  ~PlScComputer();
```

Destructor

Operations:

- DECLARE_CLASS

```
int DECLARE_CLASS(PlScComputer );
```

class PlScComputer (-)

Instances of PlScComputer are schedulable computer resources.

- get

```
virtual int get(istream& istr);
```

Get myself from a stream

- operator =

```
PlScComputer& operator =(const PlScComputer& orig);
```

Assignment operator

- put

```
virtual int put(ostream& ostr) const;
```

Put myself onto a stream

- xdr

```
virtual int xdr(XDR* xdrs);
```

Get/put myself onto an XDR stream

6.3.11.3.2 PlScCpu Class

Overview:

class PlScCpu (-)

Instances of PlScCpu are schedulable CPU resources.

Export Control: Public

Inheritance Relationships:

Inherits from PlScResource

Attributes:

Constructors and Destructor:

```
PlScCpu();
```

Default constructor

```
PlScCpu(const PlScCpu& orig);
```

Copy constructor

```
virtual  ~PlScCpu();
```

Destructor

Operations:

- addAllocNode

```
virtual int addAllocNode(RSimpleAct* act, const HEpochInterval&
interval, const char* planName);
```

Add the appropriate alloc node for me to the given activity for the plan and interval.

- createRsState

```
RRsState* createRsState(RSimpleAct* act, const HEpochInterval&
intvl);
```

Create the appropriate resource state for this activity and interval.

- DECLARE_CLASS

```
int DECLARE_CLASS(PlScCpu );
```

class PlScCpu (-)

Instances of PlScCpu are schedulable CPU resources.

- get

  ```
  virtual int get(istream& istr);
  ```

  Get myself from a stream

- operator =

  ```
  PlScCpu& operator =(const PlScCpu& orig);
  ```

  Assignment operator

- put

  ```
  virtual int put(ostream& ostr) const;
  ```

  Put myself onto a stream

- xdr

  ```
  virtual int xdr(XDR* xdrs);
  ```

  Get/put myself onto an XDR stream

6.3.11.3.3 PlScForceLink Class

Overview:

class PlScForceLink (force link for schedule resource classes)

Instances of PlScForceLink are used to force linkage of schedule resource model classes which may not be present at link time.

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
PlScForceLink();
```

Default constructor

```
PlScForceLink(const PlScForceLink& orig);
```

Copy constructor

```
virtual  ~PlScForceLink();
```

Destructor

Operations:

- DECLARE_LOCAL_CLASS

  ```
  int DECLARE_LOCAL_CLASS(PlScForceLink );
  ```

  class PlScForceLink (force link for schedule resource classes)

  Instances of PlScForceLink are used to force linkage of schedule resource model classes which may not be present at link time.

- linkage

  ```
  void linkage();
  ```

  Link stuff

- operator =

  ```
  PlScForceLink& operator =(const PlScForceLink& orig);
  ```

  Assignment operator

6.3.11.3.4 PlScLocalDisk Class

Overview:

class PlScLocalDisk (-)

Instances of PlScLocalDisk are schedulable local disk resources.

Export Control: Public

Inheritance Relationships:

Inherits from PlScResource

Attributes:

Constructors and Destructor:

```
PlScLocalDisk();
```

Default constructor

```
PlScLocalDisk(const PlScLocalDisk& orig);
```

Copy constructor

```
virtual  ~PlScLocalDisk();
```

Destructor

Operations:

- DECLARE_CLASS

  ```
  int DECLARE_CLASS(PlScLocalDisk );
  ```

  class PlScLocalDisk (-)

  Instances of PlScLocalDisk are schedulable local disk resources.

- get

  ```
  virtual int get(istream& istr);
  ```

  Get myself from a stream

- operator =

  ```
  PlScLocalDisk& operator =(const PlScLocalDisk& orig);
  ```

  Assignment operator

- put

  ```
  virtual int put(ostream& ostr) const;
  ```

  Put myself onto a stream

- xdr

  ```
  virtual int xdr(XDR* xdrs);
  ```

  Get/put myself onto an XDR stream

6.3.11.3.5 PlScResource Class

Overview:

class PlScResource (base class schedulable resource)

This class sets up protocol for all schedulable resources. It also adds a list of associated scheduling resources which will be notified when the state of this resource changes due to a scheduling request.

Export Control: Public

Inheritance Relationships:

Attributes:

```
myAffectedRs: HObjList
```

my list of RResources

Constructors and Destructor:

```
PlScResource();
```

Constructor

```
PlScResource(const PlScResource& rsrc);
```

Copy constructor

```
virtual  ~PlScResource();
```

Destructor

Operations:

- actAllowed

```
virtual int actAllowed(const RSimpleAct* act);
```

Return TRUE if given activity is allowed to be scheduled on this resource. Default behavior is to return TRUE if activity is a PlAcActivity. Derived classes should override for specialized activities

- actStateType

```
virtual HClassDesc& actStateType();
```

Returns a class description for the type of state which I will add to my resource when I need to allocate an activity.

- addAllocNode

```
virtual int addAllocNode(RSimpleAct* act, const HEpochInterval&
intvl, const char* plan);
```

Add the appropriate alloc node for me to the given activity for the plan and interval.

- addAssoc

```
virtual int addAssoc(HObject* obj);
```

Add a scheduling resource association Default behavior is to add to my list of affected resources. I retain visibility to the passed in resource, but do not assume ownership.

- alloc

```
virtual int alloc(RSimpleAct* act, const HEpochInterval& intvl,
const char* plan, int ovrsAllowed);
```

Try to allocate the activity to my resource for the given time interval. TRUE is returned if the allocation is successful. Tries the allocation on all of its associated schedulable resources, then calls checkAlloc and forceAll on my resource. This interface adds a flag indicating whether or

not to allow oversubsription (ie. allow overlapping activities) of my resource. The name has been changed from all() to alloc() to prevent the need to override more than one behavior from this class.

- all

  ```
  virtual int all(RSimpleAct* act, const HEpochInterval& intvl, const char* plan);
  ```

  Original interface to do allocation in SResource. Overridden here to call alloc() above with ovrsAllowed set to FALSE.

- checkAffectedRs

  ```
  virtual int checkAffectedRs(RSimpleAct* act, const HEpochInterval& intvl, const char* plan, int ovrsAllowed);
  ```

  Check to see if the activity can be allocated on my affected resources for the given time interval. TRUE if allocation would be successful, FALSE otherwise.

- checkAlloc

  ```
  virtual int checkAlloc(RSimpleAct* act, const HEpochInterval& intvl, const char* plan, int ovrsAllowed);
  ```

  Check to see if the activity can be allocated to my resource for the given time interval. TRUE if allocation would be successful, FALSE otherwise. Checks to make sure that the activity can be allocated on my affected schedulable resources then checks for mutual exclusion if oversubsciption is disallowed. This interface adds a flag indicating whether or not to allow oversubsription (ie. allow overlapping activities) of my resource. The name has been changed from checkAll() to checkAlloc() to prevent the need to override more than one behavior from this class in classes derived from this class.

- checkAll

  ```
  virtual int checkAll(RSimpleAct* act, const HEpochInterval& intvl, const char* plan);
  ```

  Original interface to check allocation in SResource. overridden here to call checkAlloc above with ovrsAllowed set to FALSE.

- createRsState

  ```
  virtual RRsState* createRsState(RSimpleAct* act, const HEpochInterval& intvl);
  ```

  Create the appropriate resource state for this activity and interval.

- DECLARE_CLASS

  ```
  int DECLARE_CLASS(PlScResource );
  ```

  class PlScResource (base class schedulable resource)

This class sets up protocol for all schedulable resources. It also adds a list of associated scheduling resources which will be notified when the state of this resource changes due to a scheduling request.

- findActsToBump

```
virtual int findActsToBump(const char* plan, const
HEpochInterval& intvl, HObjCollection& col, int ovrsAllowed);
```

Find the activities allocated to me that would need to be bumped in order to make room on me over the given interval on the given plan. activities found are added to the collection, if they are not already in the collection. the number added is returned. base class behavior is to add nothing to collection and return 0.

- forceAffectedRs

```
virtual int forceAffectedRs(RSimpleAct* act, const
HEpochInterval& intvl, const char* plan);
```

Force an allocation for the given activity onto my affected resources over the given time interval. TRUE if allocation is successful, FALSE otherwise.

- forceAll

```
virtual int forceAll(RSimpleAct* act, const HEpochInterval&
intvl, const char* planName);
```

Create resource state and put it on my associated resource. This member creates an instance of RActState for the resource and tells the activity to remember the allocation. The exact state type added to the resource is determined by actStateType(), which may be overridden in derived classes. TRUE is returned if everything worked, FALSE otherwise.

- intersectList

```
virtual int intersectList(const HEpochIntervalList& list1, const
HEpochIntervalList& list2, HObjCollection& retColl, int
emptyRetColl);
```

Determine intersection of two lists and fill collection with results. If emptyRetColl is TRUE, retColl will be emptied before being filled with the intersection intervals. this enables you to specify the same list as both an input list and the output list if you wish, since the algorithm will completely process the input lists before clearing the output list. this technique can be useful if you are tying to compute the overall intersection of list1, list2, list3, and list4, for example, and you don't need to keep the original list1 around. then you could compute the desired intersection by making the calls: intersectList(list1, list2, list1, TRUE); intersectList(list1, list3, list1, TRUE); intersectList(list1, list4, list1, TRUE); if emptyRetColl is FALSE (the default), retColl will NOT be emptied before being filled with the intersection intervals.

- operator =

```
PlScResource& operator =(const PlScResource& rsrc);
```

Assignment

- printInterval

```
void printInterval(const HEpochInterval& intvl);
```

Print the given interval to stdout in a nice form. Used for debugging.

- remAffectedRs

```
virtual int remAffectedRs(RSimpleAct* act, const HEpochInterval&
intvl, const char* plan);
```

Remove the allocation for the given activity from my affected resources over the given time interval. TRUE if removal is successful, FALSE otherwise.

- remAll

```
virtual int remAll(RSimpleAct* act, const HEpochInterval& intvl,
const char* planName);
```

Remove the resource states generated by the activity over the given interval.

- whenAffectedRs

```
virtual int whenAffectedRs(RSimpleAct* act, const
HEpochInterval* intvl, const char* plan, HEpochIntervalList&
availList, int ovrsAllowed);
```

Modify periods in availList to acccomodate available periods in my affected resources.

- whenAlloc

```
virtual int whenAlloc(RSimpleAct* act, const HEpochInterval*
intvl, const char* plan, HObjCollection& col, int ovrsAllowed);
```

Check to see when the activity could be allocated. The number of intervals added to the collection is returned. An interval is added to the collection if allocation is possible over that interval. This interface adds a flag indicating whether or not to allow oversubscription (ie. allow overlapping activities) of my resource. The name has been changed from whenAll() to whenAlloc() to prevent the need to override more than one behavior from this class.

- whenAll

```
virtual int whenAll(RSimpleAct* act, HEpochInterval* intvl,
const char* plan, HObjCollection& col);
```

Original interface to when allocation in SResource. overridden here to call whenAlloc above with ovrsAllowed set to FALSE.

6.3.12 Planning_Workbench_Si Class Category

6.3.12.1 Overview

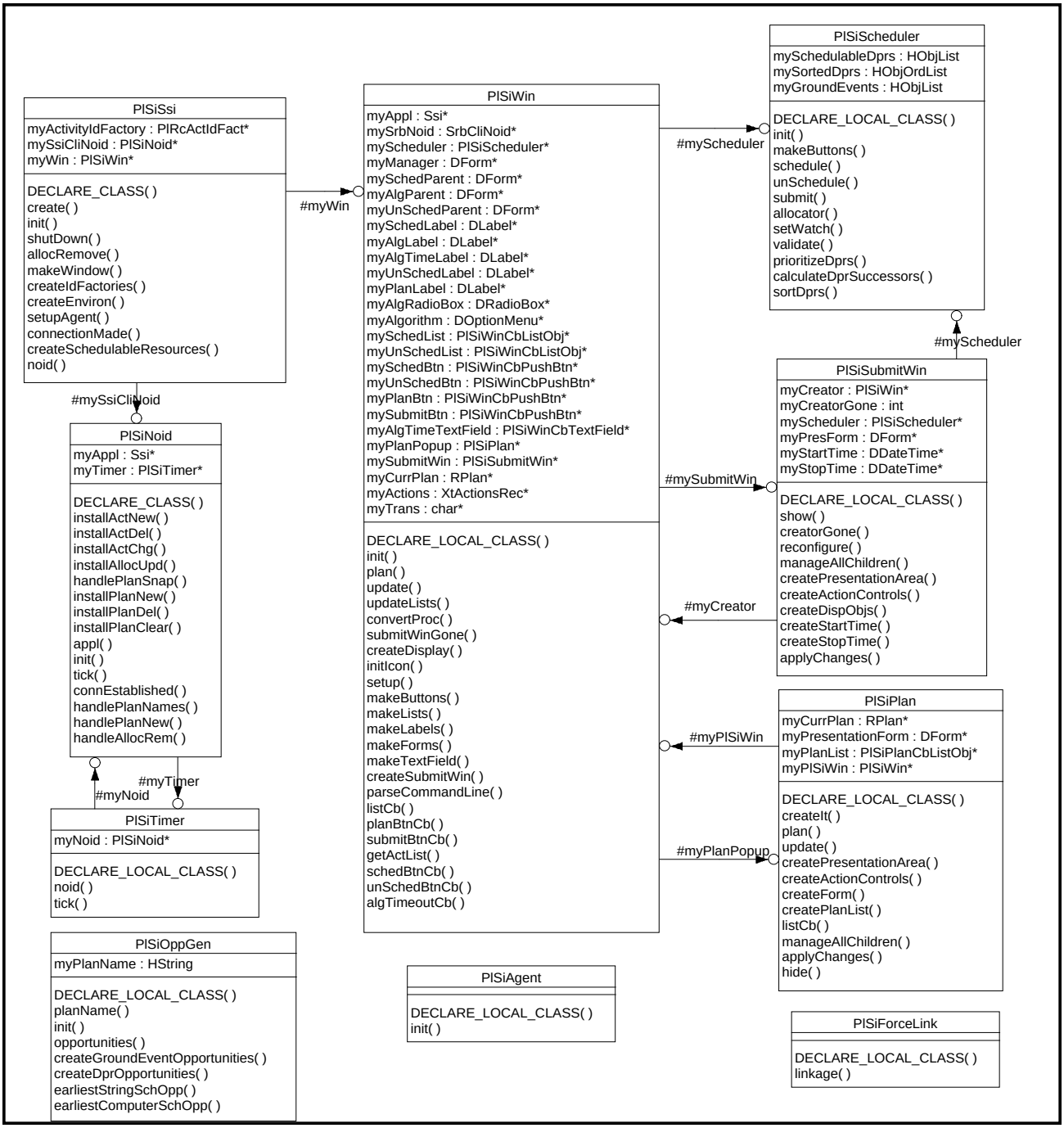


Figure 6.3.12.1-1 Planning_Workbench_Si

6.3.12.2 Planning_Workbench_Si Classes

6.3.12.3 PlSiAgent Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
PlSiAgent(const PlSiAgent& orig);
```

copy constructor

```
PlSiAgent(PlSiSsi* appl);
```

default constructor

```
virtual  ~PlSiAgent();
```

default destructor

Operations:

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlSiAgent );
```

- init

```
virtual int init();
```

Override from base class

- operator =

```
PlSiAgent& operator =(const PlSiAgent& orig);
```

assignment operator

6.3.12.3.1 PlSiForceLink Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
PlSiForceLink();
```

default constructor

```
PlSiForceLink(const PlSiForceLink& orig);
```

copy constructor

```
~PlSiForceLink();
```

destructor

Operations:

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlSiForceLink );
```

- linkage

```
void linkage();
```

link stuff

- operator =

```
PlSiForceLink& operator =(const PlSiForceLink& orig);
```

assignment operator

6.3.12.3.2 PlSiNoid Class

Overview:

Export Control: Public

Inheritance Relationships:

Inherits from PlClSrmNoid

Attributes:

```
myAppl: Ssi*
```

```
myTimer: PlSiTimer*
```

Constructors and Destructor:

```
PlSiNoid(const PlSiNoid& orig);
```

Copy constructor

```
PlSiNoid();
```

Constructor

```
virtual  ~PlSiNoid();
```

Destructor

Operations:

- appl

```
virtual void appl(Ssi* appl);
```

Give me visibility to my application.

- connEstablished

```
virtual int connEstablished();
```

When connection is established, initialize.

- DECLARE_CLASS

```
int DECLARE_CLASS(PlSiNoid );
```

- handleAllocRem

  ```
  virtual int handleAllocRem(HMsgConn* , HAddress* , SMsgAllocRem* );
  ```

  Method to handle a message indicating whether an allocation remove that I requested was successful.

- handlePlanNames

  ```
  virtual int handlePlanNames(HMsgConn* , HAddress* , SMsgPlanNmsRsp* msg);
  ```

  handle plan names by giving then to the parent

- handlePlanNew

  ```
  virtual int handlePlanNew(HMsgConn* , HAddress* , SMsgPlanNew* );
  ```

  Method to handle new plan name

- handlePlanSnap

  ```
  virtual int handlePlanSnap(HMsgConn* , HAddress* , SMsgPlanSnap* msg);
  ```

  Override install behavior for plans.

- init

  ```
  virtual int init();
  ```

  Initialize the application.

- installActChg

  ```
  virtual int installActChg(RActivity* act);
  ```

- installActDel

  ```
  virtual int installActDel(int actId);
  ```

- installActNew

  ```
  virtual int installActNew(RActivity* act);
  ```

  Override install behavior for activites.

- installAllocUpd

  ```
  virtual int installAllocUpd(RUpdAllocAbs* anUpdate);
  ```

  Override install behavior for allocations.

- installPlanClear

```
virtual int installPlanClear(const char* planName,
HEpochInterval* intvl, HVList* rsIds);
```

- installPlanDel

```
virtual int installPlanDel(const char* planName);
```

- installPlanNew

```
virtual int installPlanNew(const char* planName, HVList* rsIds);
```

- operator =

```
PlSiNoid& operator =(const PlSiNoid& orig);
```

Assignment operator

- tick

```
virtual void tick();
```

Fire my timer.

6.3.12.3.3 PlSiOppGen Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

```
myPlanName: HString
```

The plan for which I am generating opportunities.

Constructors and Destructor:

```
PlSiOppGen();
```

default constructor

```
virtual  ~PlSiOppGen();
```

default destructor

Operations:

- createDprOpportunities

```
virtual int createDprOpportunities(RActivity* act, RSchRqst*
req);
```

Attempt to create a scheduling opportunity for this request from the DPR activity information

- createGroundEventOpportunities

```
virtual int createGroundEventOpportunities(RActivity* act,
RSchRqst* req);
```

Attempt to create a scheduling opportunity for this request from the DPR activity information

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlSiOppGen );
```

- earliestComputerSchOpp

```
const RSchOpp earliestComputerSchOpp(PlRcComputer* computer,
const HEpochTime& earliestStartTime, double dprDuration);
```

Find a scheduling opportunity for the supplied computer starting later than the given start time for the given duration.

- earliestStringSchOpp

```
const RSchOpp earliestStringSchOpp(PlRcString* string, const
HEpochTime& earliestStartTime, double dprDuration);
```

Find a scheduling opportunity for one of the supplied computers starting later than the given start time for the given duration.

- init

```
virtual int init();
```

initialization

- opportunities

```
virtual int opportunities(RActivity* act, RSchRqst* req);
```

Attempt to create a scheduling opportunity for this request from the activity information

- planName

```
virtual HString& planName();
```

- planName

```
virtual void planName(const char* plan);
```

set/get the plan name

6.3.12.3.4 PlSiPlan Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

```
myCurrPlan: RPlan*
```

```
myPlanList: PlSiPlanCbListObj*
```

```
myPlSiWin: PlSiWin*
```

```
myPresentationForm: DForm*
```

Constructors and Destructor:

```
PlSiPlan(const PlSiPlan& orig);
```

Copy constructor

```
PlSiPlan();
```

Constructor

```
virtual  ~PlSiPlan();
```

Destructor

Operations:

- applyChanges

```
virtual void applyChanges();
```

- createActionControls

```
virtual int createActionControls();
```

Modify my action area controls. Overriden from base class.

- createForm

```
virtual int createForm();
```

Create my form that will hold my two lists.

- createIt

```
virtual int createIt(PlSiWin* parent);
```

- createPlanList

```
virtual int createPlanList();
```

Creates the accept list of clients.

- createPresentationArea

```
virtual int createPresentationArea();
```

Establishes protocol for creating the display. Overridden from base class.

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlSiPlan );
```

- hide

```
virtual void hide();
```

- listCb

```
virtual void listCb(PlSiPlanCbListObj* , XtPointer );
```

- manageAllChildren

```
virtual void manageAllChildren();
```

Ask all of my pieces to manageChild().

- operator =

```
PlSiPlan& operator =(const PlSiPlan& orig);
```

Assignment operator

- plan

```
virtual RPlan* plan();
```

- plan

```
virtual void plan(RPlan* aplan);
```

- update

```
virtual void update();
```

6.3.12.3.5 PlSiScheduler Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

`myGroundEvents: HObjList`

List of Ground Events created from the selected activities.

`mySchedulableDprs: HObjList`

List of valid, schedulable DPR activities from the selected activities.

`mySortedDprs: HObjOrdList`

Sorted list of DPRs. This relies on the DPR compare operation.

Constructors and Destructor:

```
PlSiScheduler(const PlSiScheduler& );
```

Copy constructor (Disallows copying)

```
PlSiScheduler(Ssi* appl);
```

Default constructor

```
virtual  ~PlSiScheduler();
```

Default destructor

Operations:

- allocator

```
virtual SaAllocator* allocator();
```

- allocator

```
virtual int allocator(SaAllocator* allocator);
```

Get/Set my allocator explicitly

- calculateDprSuccessors

```
virtual void calculateDprSuccessors();
```

Calculate the number of DPR successors for each of the DPRs

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlSiScheduler );
```

- init

```
virtual int init();
```

- makeButtons

```
virtual int makeButtons(DRadioBox* parent);
```

- operator =

```
PlSiScheduler& operator =(const PlSiScheduler& );
```

Assignment operator (Disallows assignments)

- prioritizeDprs

  ```
  virtual void prioritizeDprs();
  ```

  Calculate the "task level" and the earliest scheduled start time for each of the schedulable DPRs.

- schedule

  ```
  virtual int schedule(const char* planName, HObjCollection& col);
  ```

  Schedule activities

- setWatch

  ```
  virtual void setWatch(int flag);
  ```

  Set my cursor to watch (if TRUE) or back to normal (if FALSE). overridden from base class to avoid core dump when there is no window.

- sortDprs

  ```
  virtual void sortDprs();
  ```

  Sort the DPRs and insert them into the sorted DPR list.

- submit

  ```
  int submit(const RPlan* plan, const HEpochInterval& intvl);
  ```

  Submit jobs to processing for the given interval of the currently selected plan.

- unSchedule

  ```
  virtual int unSchedule(const char* planName, HObjCollection&
  col);
  ```

  Unschedule activities

- validate

  ```
  virtual void validate(HObjCollection& selectedActs);
  ```

  Check selected activites to make sure their dependencies can be met and store schedulable activities in the DPR and Ground Event lists

6.3.12.3.6 PlSiSsi Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

`myActivityIdFactory: PlRcActIdFact*`

my activity id factory

`mySsiCliNoid: PlSiNoid*`

my noid

`myWin: PlSiWin*`

set my window

Constructors and Destructor:

`PlSiSsi(const PlSiSsi& orig);`

copy constructor

`PlSiSsi();`

default constructor

`virtual  ~PlSiSsi();`

destructor

Operations:

- allocRemove

  `virtual int allocRemove(const char* planName, const HEpochInterval& when, int rsId, int actId);`

  Remove an allocation against the specified parameters.

- connectionMade

  `virtual int connectionMade();`

  do the rest of the initializing when a connection is made

- createEnviron

  `virtual int createEnviron();`

  set up an Environment

- createIdFactories

  `virtual int createIdFactories();`

  create the id generator factories

- createSchedulableResources

```
virtual int createSchedulableResources();
```

Create new schedulable resources (PlRcResource) and add them to the schedulable resource pool.

- create

```
virtual int create(const char* appClass, int& argc, char** argv,
XrmOptionDescList options, Cardinal numOptions);
```

- DECLARE_CLASS

```
int DECLARE_CLASS(PlSiSsi );
```

- init

```
virtual int init();
```

initialize the Scheduling interface

- makeWindow

```
virtual DWindow* makeWindow();
```

make a PDPS Ssi Window

- noid

```
virtual PlSiNoid* noid() const;
```

gets SsiCliNoid

- operator =

```
PlSiSsi& operator =(const PlSiSsi& orig);
```

assignment operator

- setupAgent

```
virtual int setupAgent();
```

create a new Agent and initialize it.

- shutDown

```
virtual void shutDown();
```

clean up pointers

6.3.12.3.7 PlSiSubmitWin Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

`myCreatorGone: int`
tells me if my creator is still around

`myCreator: PlSiWin*`
my creator window

`myPresForm: DForm*`
my presentation area form

`myScheduler: PlSiScheduler*`
my scheduler

`myStartTime: DDateTime*`
my start and stop time items

`myStopTime: DDateTime*`

Constructors and Destructor:

`PlSiSubmitWin(PlSiWin* creator, PlSiScheduler* ssiScheduler);`
constructor.

`PlSiSubmitWin(const PlSiSubmitWin& orig);`
copy, assignment - disabled

`~PlSiSubmitWin();`
destructor

Operations:

- applyChanges

```
virtual void applyChanges();
```

apply changes. tell PlSiScheduler to switch to the new total interval overrides DApplyDialog::applyChanges()

- createActionControls

```
virtual int createActionControls();
```

Modify my action area controls. Overriden from base class.

- createDispObjs

```
virtual int createDispObjs();
```

create the display objects that make up my presentation area.

- createPresentationArea

```
virtual int createPresentationArea();
```

make my presentation area. overrides DApplyDialog::createPresentationArea()

- createStartTime

```
virtual int createStartTime();
```

create the start time item

- createStopTime

```
virtual int createStopTime();
```

create the stop time item

- creatorGone

```
virtual void creatorGone(int tf);
```

set whether or not my creator is gone. if my creator gets destroyed first, it will tell me

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlSiSubmitWin );
```

- manageAllChildren

```
virtual void manageAllChildren();
```

ask all of pieces to manageChild. does the pieces I created plus my base class. overrides DApplyDialog::manageAllChildren()

- operator =

```
PlSiSubmitWin& operator =(const PlSiSubmitWin& orig);
```

- reconfigure

  ```
  virtual int reconfigure();
  ```

  reconfigure. reinitialize the available and viewed lists

- setStartStopTimes

  ```
  virtual int setStartStopTimes();
  ```

  initialize the start and stop times with the current total interval

- setStartTimePos

  ```
  virtual void setStartTimePos();
  ```

  set the position of the start time item

- setStopTimePos

  ```
  virtual void setStopTimePos();
  ```

  set the position of the stop time item

- show

  ```
  virtual void show();
  ```

  show me on the desktop. override to set start and stop times to current total interval. then do base class behavior

6.3.12.3.8 PlSiTimer Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

```
myNoid: PlSiNoid*
```

Constructors and Destructor:

```
PlSiTimer(const PlSiTimer& orig);
```

Copy constructor

```
PlSiTimer(long microSeconds);
```

Default constructor

```
~PlSiTimer();
```

Destructor

Operations:

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlSiTimer );
```

- noid

```
virtual void noid(PlSiNoid* n);
```

Set my noid.

- operator =

```
PlSiTimer& operator =(const PlSiTimer& orig);
```

Assignment operator

- tick

```
virtual void tick();
```

Override base tick() behavior.

6.3.12.3.9 PlSiWin Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

```
myActions: XtActionsRec*
```

```
myAlgLabel: DLabel*
```

```
myAlgorithm: DOptionMenu*

myAlgParent: DForm*

myAlgRadioBox: DRadioBox*

myAlgTimeLabel: DLabel*

myAlgTimeTextField: PlSiWinCbTextField*

myAppl: Ssi*

myCurrPlan: RPlan*

myManager: DForm*

myPlanBtn: PlSiWinCbPushBtn*

myPlanLabel: DLabel*

myPlanPopup: PlSiPlan*

mySchedBtn: PlSiWinCbPushBtn*

mySchedLabel: DLabel*

mySchedList: PlSiWinCbListObj*

mySchedParent: DForm*
```

```
myScheduler: PlSiScheduler*

mySrbNoid: SrbCliNoid*

mySubmitBtn: PlSiWinCbPushBtn*

mySubmitWin: PlSiSubmitWin*

myTrans: char*

myUnSchedBtn: PlSiWinCbPushBtn*

myUnSchedLabel: DLabel*

myUnSchedList: PlSiWinCbListObj*

myUnSchedParent: DForm*
```

Constructors and Destructor:

```
PlSiWin(const PlSiWin& orig);
```

Copy constructor

```
PlSiWin(Ssi* appl);
```

Default Constructor

```
virtual  ~PlSiWin();
```

Default Destructor

Operations:

- algTimeoutCb

```
virtual void algTimeoutCb(PlSiWinCbTextField* , XtPointer cbs);
```

- convertProc

```
virtual int convertProc(Widget widget, Atom* selection, Atom*
target, Atom* type, XtPointer* value, unsigned long*
value_length, int* format);
```

- createDisplay

```
virtual int createDisplay();
```

Overriden from base class to provide window display creation.

- createSubmitWin

```
virtual PlSiSubmitWin* createSubmitWin();
```

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlSiWin );
```

- getActList

```
virtual int getActList(PlSiWinCbListObj* list, HObjCollection&
objCol);
```

- initIcon

```
virtual void initIcon();
```

Overriden from base class to set my personal icon.

- init

```
virtual int init();
```

- listCb

```
virtual void listCb(PlSiWinCbListObj* , XtPointer cbs);
```

- makeButtons

```
virtual int makeButtons();
```

Build toggle buttons.

- makeForms

```
virtual int makeForms();
```

- makeLabels

```
virtual int makeLabels();
```

- makeLists

```
virtual int makeLists();
```

Build interface items.

- makeTextField

```
virtual int makeTextField();
```

- operator =

```
PlSiWin& operator =(const PlSiWin& orig);
```

Assignment operator

- parseCommandLine

```
virtual int parseCommandLine();
```

Callback functions.

- planBtnCb

```
virtual void planBtnCb(PlSiWinCbPushBtn* , XtPointer cbs);
```

- plan

```
virtual RPlan* plan();
```

- plan

```
virtual int plan(const char* planName);
```

- schedBtnCb

```
virtual void schedBtnCb(PlSiWinCbPushBtn* , XtPointer cbs);
```

- setup

  ```
  virtual int setup();
  ```

  Setup display.

- submitBtnCb

  ```
  virtual void submitBtnCb(PlSiWinCbPushBtn* , XtPointer cbs);
  ```

- submitWinGone

  ```
  virtual void submitWinGone();
  ```

  Tell me that the submit window is gone.

- unSchedBtnCb

  ```
  virtual void unSchedBtnCb(PlSiWinCbPushBtn* , XtPointer cbs);
  ```

- updateLists

  ```
  virtual int updateLists();
  ```

- update

  ```
  virtual int update();
  ```

6.3.13 Planning_Workbench_TI Class Category

6.3.13.1 Overview

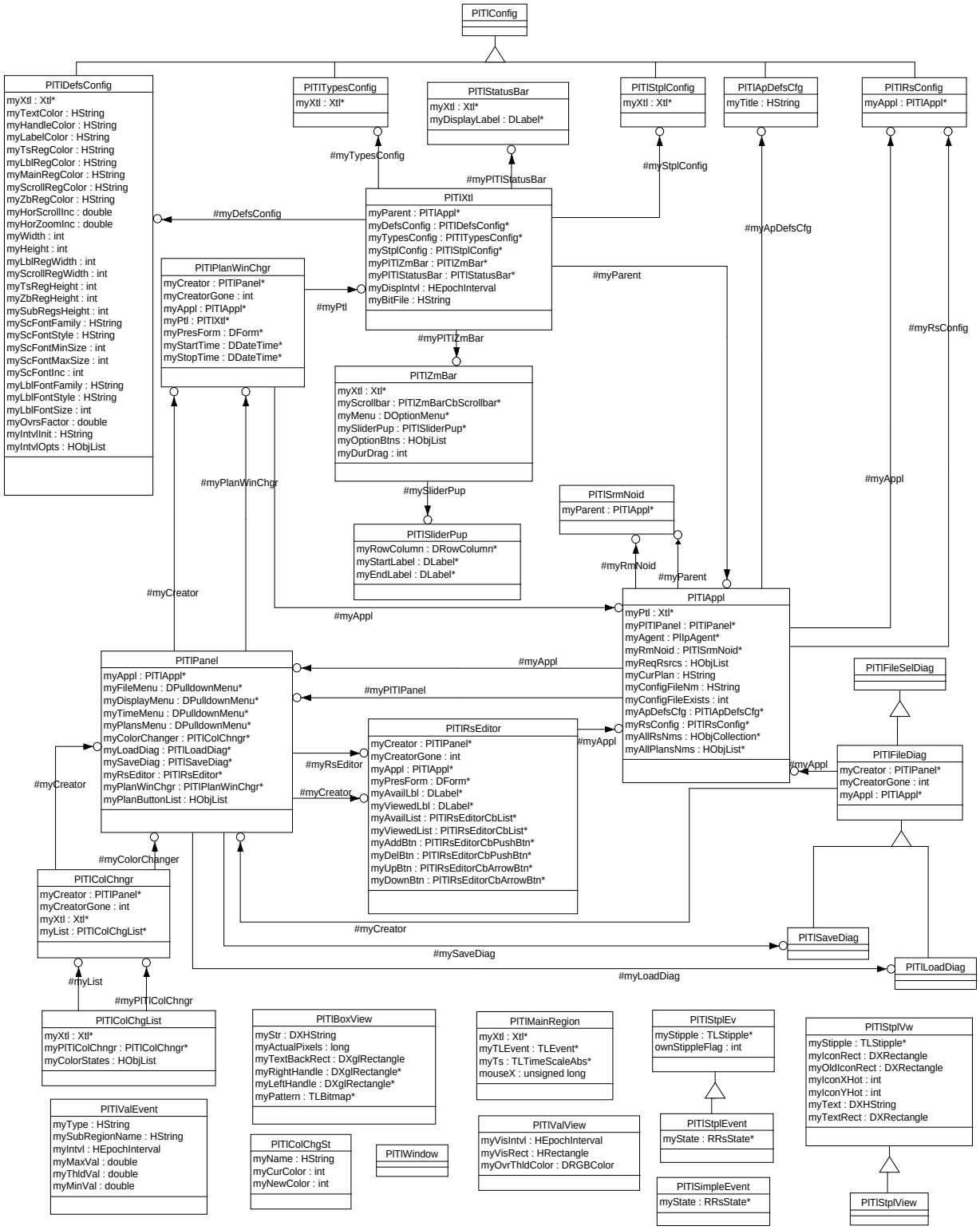


Figure 6.3.13.1-1 Planning_Workbench_TI

6.3.13.2 Planning_Workbench_Tl Classes

6.3.13.3 PlTlApDefsCfg Class

Overview:

class PlTlApDefsCfg (-)

Instances of PlTlApDefsCfg handle configuration files for default display resource settings for the timeline.

Export Control: Public

Inheritance Relationships:

Inherits from PlTlConfig

Attributes:

`myTitle: HString`

My frame title

Constructors and Destructor:

`PlTlApDefsCfg();`

Default constructor

`PlTlApDefsCfg(const PlTlApDefsCfg& orig);`

Copy constructor

`virtual ~PlTlApDefsCfg();`

destructor

Operations:

- DECLARE_LOCAL_CLASS

  `int DECLARE_LOCAL_CLASS(PlTlApDefsCfg );`

  class PlTlApDefsCfg (-)

  Instances of PlTlApDefsCfg handle configuration files for default display resource settings for the timeline.

- handleKeyword

  `virtual int handleKeyword(const HString& kywd);`

  Handle keyword. Overrides ConfigurationFile::handleKeyword()

- operator =

```
PlTlApDefsCfg& operator =(const PlTlApDefsCfg& orig);
```

Assignment operator

- saveConfig

```
virtual int saveConfig(ostream& str);
```

Save the current set of resources to the given stream. Overrides PlTlConfig::saveConfig()

- title

```
virtual const HString& title() const;
```

Get my frame title

6.3.13.3.1 PlTlAppl Class

Overview:

class PlTlAppl (timeline application)

A PlTlAppl is the main application class for the timeline. It is responsible for setting up communication to the resource model as well as initialization of resources and their states. PlTlAppl also handles the cleanup and subsequent destruction of the appl.

Export Control: Public

Inheritance Relationships:

Attributes:

```
myAgent: PlIpAgent*
```

My noid agent

```
myAllPlansNms: HObjList*
```

My list of the names of all available plans

```
myAllRsNms: HObjCollection*
```

My list of the names of all available resources

```
myApDefsCfg: PlTlApDefsCfg*
```

My PlTlXtl appl defaults configuration file

```
myConfigFileExists: int
```

Flag indicating whether config file exists

```
myConfigFileNm: HString
```

My configuration file name

`myCurPlan: HString`

The current plan I am interested in

`myPlTlPanel: PlTlPanel*`

My PlTlPanel display

`myPtl: Xtl*`

My planning timeline

`myReqRsrcs: HObjList`

My requested resources. I own the list and am responsible for its deletion

`myRmNoid: PlTlSrmNoid*`

My resource model noid

`myRsConfig: PlTlRsConfig*`

My PlTlXtl resource configuration file

Constructors and Destructor:

`PlTlAppl();`

Default constructor

`PlTlAppl(const PlTlAppl& orig);`

Copy constructor

`virtual  ~PlTlAppl();`

Destructor

Operations:

- addPlanToMenu

  `virtual int addPlanToMenu(HString );`

  Add a new button to the PlTlXtl's Plans menu and also add plan to my list of plan names

- allPlansNms

  `virtual void allPlansNms(HObjList* );`

  Set the list containing the names of all availabe plans. I assume ownership of the contents of the list.

- allPlansNms

```
virtual HObjList* allPlansNms();
```

Get the list containing the names of all availabe plans.

- allRsNms

```
virtual void allRsNms(HObjCollection* );
```

set/get the list containing the names of all available resources. I assume ownership of the contents of the list.

- allRsNms

```
virtual HObjCollection* allRsNms();
```

Get the list containing the names of all available resources.

- changeViewedRs

```
virtual int changeViewedRs(const HObjCollection& newRsNms);
```

Change the resources I am displaying to those whose names are given in the collection.

- cleanupForWinDestroy

```
virtual void cleanupForWinDestroy();
```

Delete my PlTlXtl before the top level window gets destroyed so that all the regions will still be there when stuff cleans up

- cleanup

```
virtual void cleanup();
```

Delete all dynamically created entities

- configFileExists

```
virtual int configFileExists();
```

Get whether or not the config file exists right now

- configFileNm

```
virtual const HString& configFileNm() const;
```

Get my config file name

- connectionMade

```
virtual int connectionMade();
```

Upon notification that a connection to the resource model has been made, proceed with rest of initialization, specifically display creation, resource model loading, and obtaining state

- createApDefsCfg

```
virtual PlTlApDefsCfg* createApDefsCfg();
```

Create my appl defaults configuratin file

- createPtl

```
virtual Xtl* createPtl();
```

create my PlTlXtl

- createRsConfig

```
virtual PlTlRsConfig* createRsConfig();
```

Create my resource configuration file

- createSigMgr

```
virtual int createSigMgr();
```

Create my global signal manager

- currentPlan

```
virtual HString& currentPlan();
```

Get the current plan

- currentPlan

```
virtual void currentPlan(const char* );
```

Set the current plan

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlTlAppl );
```

class PlTlAppl (timeline application)

A PlTlAppl is the main application class for the timeline. It is responsible for setting up communication to the resource model as well as initialization of resources and their states. PlTlAppl also handles the cleanup and subsequent destruction of the appl.

- deletePlan

```
virtual int deletePlan();
```

Delete plan events

- delRsStates

```
virtual int delRsStates(const HObjCollection& rsToDelete);
```

Delete the state lists of resources I no longer want to display

- deregisterUpdates

```
virtual int deregisterUpdates();
```

Deregister for all updates

- detConfigFile

  ```
  virtual int detConfigFile();
  ```

  Determine the configuration file name. If specified on command line, the file must exist. If not specified on command line, use a default name which may or may not exist. return FALSE if file must exist, but does not

- detRsToAdd

  ```
  virtual void detRsToAdd(const HObjCollection& newRsNms,
  HObjCollection& rsToAdd);
  ```

  Determine new resources not in the current set

- detRsToDel

  ```
  virtual void detRsToDel(const HObjCollection& newRsNms,
  HObjCollection& rsToDelete);
  ```

  Determine current resources not in the new set

- displayWinAtRun

  ```
  virtual void displayWinAtRun();
  ```

  Override to NOT map the window at run we don't want to map it until later, after we have gotten all our connections and our data

- getInitialStates

  ```
  virtual int getInitialStates();
  ```

  Get initial resource states

- getNewStates

  ```
  virtual int getNewStates(const HObjCollection& rsToAdd);
  ```

  Request new state lists for new resources I now want to display

- initCommunications

  ```
  virtual int initCommunications();
  ```

  Initialize message agent and resource model noid and try to connect to resource model.

- init

  ```
  virtual int init();
  ```

  Initialize the application

- loadAllPlans

```
virtual int loadAllPlans();
```

Load the plans the SRM knows about

- loadAllRsrcNms

```
virtual int loadAllRsrcNms();
```

Load the names of all the resources the SRM knows about

- loadApplDefs

```
virtual int loadApplDefs();
```

Load my application defaults

- loadConfig

```
virtual int loadConfig(const char* filenm);
```

Load a new configuration from the given file and snap to that configurtation

- loadNewRs

```
virtual int loadNewRs(const HObjCollection& rsToAdd);
```

Load any new resources not already in the pool from the SRM

- loadResourcePool

```
virtual int loadResourcePool();
```

Create and load the resource pool with the resources associated with the sub regions

- makeWindow

```
virtual DWindow* makeWindow();
```

Make the derived DWindow which holds the top-level shell

- newPlansMenu

```
virtual int newPlansMenu(HObjList* );
```

Tell my PlTlXtl to re-make its Plans menu with a new list of plan names

- newRs

```
virtual void newRs(RResource* aRs);
```

Called when my noid gets notification of a new resource added to my pool.

- operator =

```
PlTlAppl& operator =(const PlTlAppl& orig);
```

Assignment operator

- ptlChangeViewedRs

```
virtual int ptlChangeViewedRs(const HObjCollection& rsToDelete);
```

Notify my PlTlXtl to change the resources it is displaying

- ptlSaveConfig

```
virtual int ptlSaveConfig(const char* filenm);
```

Tell my PlTlXtl to save its configuration to the given file

- reconfigPtl

```
virtual int reconfigPtl();
```

Tell my PlTlXtl to reconfigure

- reconfigRs

```
virtual int reconfigRs();
```

Reconfigure my resources. This includes deregistering for all current updates, clearing my resource pool, and requesting the new set of resources

- reconfigure

```
virtual int reconfigure();
```

Reconfigure according to new config file

- registerForPlans

```
virtual int registerForPlans();
```

Register for act, resource and plan instance updates

- registerForUpdates

```
virtual int registerForUpdates();
```

Register for resource state updates

- remPlanFromMenu

```
virtual int remPlanFromMenu(HString );
```

Remove deleted plan from the PlTlXtl's Plans menu and also remove plan from my list of plan names

- reqRsrcs

```
virtual HObjList& reqRsrcs();
```

Get the resources

- reqRsrcs

```
virtual void reqRsrcs(const HObjList& );
```

Set the resources

- saveConfig

  `virtual int saveConfig(const char* filenm);`

  Save the current configuration to the given file

- setDefConfigFile

  `virtual void setDefConfigFile();`

  Set the default config file name.

- snapToNewPlan

  `virtual int snapToNewPlan(const char* planName);`

  Request snap to new plan

- srmCliNoid

  `virtual PlTlSrmNoid* srmCliNoid();`

  Get my SRM client noid

- stateUpdate

  `virtual void stateUpdate(RUpdState* aState);`

  Called when my noid gets notification of a state update. I get the list of updates.

6.3.13.3.2 PlTlBoxView Class

Overview:

class PlTlBoxView (rectangular view on events)

Instances of this class are rectangular views on events. Each rectangle can have a name, color, handles (if selected), and a tile pattern.

Export Control: Public

Inheritance Relationships:

Attributes:

`myActualPixels: long`

The actual number of pixels corresponding to my interval width (whether or not myRect is "chopped" at the TLMainRegion border or not).

`myLeftHandle: DXglRectangle*`

myPattern: TLBitmap*

Tile pattern (null if no pattern).

myRightHandle: DXglRectangle*

Right and left handles (null if not selected).

myStr: DXHString

String to display.

myTextBackRect: DXglRectangle

Used to render rectangle behind string.

Constructors and Destructor:

PlTlBoxView(TLEvent*);

Constructor

~PlTlBoxView();

Destructor

Operations:

- actualPixels

  virtual long actualPixels();

  Return actual number of pixels corresponding to interval width.

- adjPos

  virtual void adjPos(long deltaX, long deltaY);

  Move display rectangle by given deltas.

- canDoHandles

  virtual int canDoHandles();

  If the box is too small then don't display handles.

- clearContext

  virtual int clearContext();

- createLeftHandle

```
virtual void createLeftHandle(unsigned long margX, unsigned long margY, unsigned long handleW, unsigned long handleH);
```

Create and initialize the left handle.

- createRightHandle

```
virtual void createRightHandle(unsigned long margX, unsigned long margY, unsigned long handleW, unsigned long handleH);
```

Create and initialize the right handle.

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlTlBoxView );
```

class PlTlBoxView (rectangular view on events)

Instances of this class are rectangular views on events. Each rectangle can have a name, color, handles (if selected), and a tile pattern.

- eraseOld

```
virtual void eraseOld();
```

Erase old rectangle.

- erase

```
virtual void erase();
```

Erase rectangle.

- handleHeight

```
virtual unsigned long handleHeight() const;
```

- handleSelectOnSelectedEvent

```
virtual void handleSelectOnSelectedEvent();
```

Handle another select on an already selected event.

- handleSelectOnUnselectedEvent

```
virtual void handleSelectOnUnselectedEvent();
```

Handle select on a currently unselected event.

- handleWidth

```
virtual unsigned long handleWidth() const;
```

- handleXMargin

```
virtual unsigned long handleXMargin() const;
```

Members to return information about placement and size of handles.

- handleYMargin

```
virtual unsigned long handleYMargin() const;
```

- pattern

```
virtual void pattern(TLBitmap* );
```

Set/get my pattern.

- pattern

```
virtual TLBitmap* pattern();
```

- recalcBox

```
virtual void recalcBox(TLTimeScaleAbs* );
```

- recalcHandles

```
virtual void recalcHandles();
```

- recalcName

```
virtual void recalcName();
```

- recalc

```
virtual void recalc(TLTimeScaleAbs* );
```

Members to calculate certain features of the box.

- repaintBox

```
virtual int repaintBox();
```

Repaint colored box.

- repaintName

```
virtual int repaintName();
```

Repaint picture bitmap onto view if one exists.

- repaintOutline

```
virtual int repaintOutline();
```

Repaint outline of box (usually for dragging).

- repaintSelected

```
virtual int repaintSelected();
```

Repaint handles (if selected).

- repaint

```
virtual int repaint();
```

Specifies protocol for repainting.

- selectView

```
virtual void selectView();
```

Select/unselect - handles will be displayed or not as appropriate.

- select

```
virtual void select();
```

Select - call appropriate routine to handle select depending on whether or not event is already selected. Overrides TLEventView::select.

- setRegionDragType

```
virtual void setRegionDragType();
```

Set the region drag type depending on event selection and mouse position.

- setupContext

```
virtual int setupContext();
```

- unselectView

```
virtual void unselectView();
```

- visInterval

```
virtual const HEpochInterval& visInterval();
```

Converts pixels to time and returns interval corresponding to current display rectangle.

6.3.13.3.3 PlTlColChgList Class

Overview:

class PlTlColChgList (-)

Instances of PlTlColChgList are lists which are part of the color mapper dialog.

Export Control: Public

Inheritance Relationships:

Attributes:

`myColorStates: HObjList`

My list of color states

`myPlTlColChngr: PlTlColChngr*`

My parent color changer

`myXtl: Xtl*`

My timeline

Constructors and Destructor:

`PlTlColChgList(Xtl* xtl, PlTlColChngr* chngr);`

Constructor.

`PlTlColChgList(const PlTlColChgList& orig);`

Copy construtor

`virtual  ~PlTlColChgList();`

Destructor

Operations:

- apply

  `virtual void apply();`

  Apply all the changes that have been made to the color mapper

- changeColor

  `virtual void changeColor(int ix, int cix);`

  Change the color at the given index to the given color index (pixel)

- colorAt

  `virtual int colorAt(int ix);`

  Return the color index at the given index in the list

- DECLARE_LOCAL_CLASS

  ```
  int DECLARE_LOCAL_CLASS(PlTlColChgList );
  ```

  class PlTlColChgList (-)

  Instances of PlTlColChgList are lists which are part of the color mapper dialog.

- deleteAllItems

  ```
  virtual int deleteAllItems();
  ```

  Remove all items in the scrolling list. Overridden to get rid of the objects that are associated with the list

- init

  ```
  virtual int init();
  ```

  Init the scrolling list with all the types in TLTypes.

- operator =

  ```
  PlTlColChgList& operator =(const PlTlColChgList& orig);
  ```

  Assignment operator

- reset

  ```
  virtual void reset();
  ```

  Reset any states that might have changed to their original colors

- singleSelectionRef

  ```
  virtual void singleSelectionRef(XtPointer callData, int row,
  void* ref);
  ```

  Called when a list item is selected

- update

  ```
  virtual int update();
  ```

  Update the scrolling list with new types in TLTypes.

6.3.13.3.4 PlTlColChgSt Class

Overview:

class PlTlColChgSt (-)

Instances of PlTlColChgSt map colors to types.

Export Control: Public

Inheritance Relationships:

Attributes:

`myCurColor: int`

My current color index

`myName: HString`

My type name

`myNewColor: int`

My new color index

Constructors and Destructor:

`PlTlColChgSt(const char* typNm, int colorIndex);`

Constructor. Arguments are my type name and my current color index.

`PlTlColChgSt(const PlTlColChgSt& orig);`

Copy constructor

`virtual  ~PlTlColChgSt();`

Destructor

Operations:

- colorChanged

  `virtual int colorChanged();`

  Returns TRUE if curColor and newColor are different, FALSE otherwise.

- compare

  `virtual int compare(const HObject& obj) const;`

  Compare members.

- compare

  `virtual int compare(const PlTlColChgSt& colorChgSt) const;`

- curColor

  `virtual void curColor(int index);`

  Set my current color index

- curColor

  ```
  virtual int curColor() const;
  ```

  Get my current color index

- DECLARE_LOCAL_CLASS

  ```
  int DECLARE_LOCAL_CLASS(PlTlColChgSt );
  ```

  class PlTlColChgSt (-)

  Instances of PlTlColChgSt map colors to types.

- hash

  ```
  virtual unsigned int hash() const;
  ```

  My hash is the hash of my name.

- name

  ```
  virtual void name(const char* typNm);
  ```

  Set my type name

- name

  ```
  virtual const HString& name() const;
  ```

  Get my type name

- newColor

  ```
  virtual void newColor(int index);
  ```

  Set my new color index

- newColor

  ```
  virtual int newColor() const;
  ```

  Get my new color index

- operator =

  ```
  PlTlColChgSt& operator =(const PlTlColChgSt& orig);
  ```

  Assignment operator

- reset

  ```
  virtual void reset();
  ```

  set the new color to be the current color.

- update

  ```
  virtual void update();
  ```

Set the current color to be the new color.

6.3.13.3.5 PlTlColChngr Class

Overview:

class PlTlColChngr (PTL color changer)

Instances of PlTlColChngr can be used to specify colors for types.

Export Control: Public

Inheritance Relationships:

Attributes:

`myCreatorGone: int`

Flag to indicate if my creator is still around.

`myCreator: PlTlPanel*`

My parent panel. If my paenl is still around when I get destroyed, I will tell it I am gone.

`myList: PlTlColChgList*`

My state list of types and colors

`myXtl: Xtl*`

My timeline

Constructors and Destructor:

`PlTlColChngr(PlTlPanel* creator, Xtl* xtl);`

Constructor. Argument is my creating PlTlPanel and the Xtl

`PlTlColChngr(const PlTlColChngr& orig);`

Copy constructor

`virtual ~PlTlColChngr();`

Destructor

Operations:

- applyChanges

`virtual void applyChanges();`

Apply all the color changes overrides DApplyDialog::applyChanges()

- colorChangerGone

  ```
  virtual void colorChangerGone();
  ```

  Tell my parent that I have been destroyed. Default behavior is to tell my ptl parent.

- colorChg

  ```
  virtual void colorChg(DXColChgCbPushBtn* );
  ```

  Called when the user selects a color in the palette. Update the state and preview color. overrides DXColChg::colorChg()

- createActionControls

  ```
  int createActionControls();
  ```

  Create my action controls. Overridden to add a DListObj to the action area. Overrides DApplyDialog::createActionControls()

- creatorGone

  ```
  virtual void creatorGone(int tf);
  ```

  Set when creator is destroyed.

- DECLARE_LOCAL_CLASS

  ```
  int DECLARE_LOCAL_CLASS(PlTlColChngr );
  ```

  class PlTlColChngr (PTL color changer)

  Instances of PlTlColChngr can be used to specify colors for types.

- hide

  ```
  virtual void hide();
  ```

  Reset all colors, then do base class behavior overrides DPopup::hide()

- operator =

  ```
  PlTlColChngr& operator =(const PlTlColChngr& orig);
  ```

  Assignment operator

- positionApply

  ```
  void positionApply();
  ```

  Position my apply button. Override from base class to take account of my list, added to the action area. overrides DApplyDialog::positionApply()

- positionCancel

  ```
  void positionCancel();
  ```

Position my cancel button. Override from base class to take account of my list, added to the action area. overrides DApplyDialog::positionCancel()

- positionOk

  ```
  void positionOk();
  ```

  Position my ok button. Override from base class to take account of my list, added to the action area. overrides DApplyDialog::positionOk()

- reconfigure

  ```
  virtual int reconfigure();
  ```

  Reconfigure by clearing the current scrolling list, then rebuilding it to pick up a new set of types/colors.

- sizeActionForm

  ```
  virtual int sizeActionForm();
  ```

  Overridden to take into account my addition of my scroll list of region names. Overrides DApplyDialog::sizeActionForm()

- update

  ```
  virtual int update();
  ```

  Update my type list

6.3.13.3.6 PlTlConfig Class

Overview:

class PlTlConfig (-)

PlTlConfig is the base class for all configuration files. It provides protocol for loading and saving configuration from the given file.

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
PlTlConfig();
```

Default constructor

```
PlTlConfig(const PlTlConfig& orig);
```

Copy constructor

```
virtual  ~PlTlConfig();
```

Destructor

Operations:

- DECLARE_LOCAL_CLASS

  ```
  int DECLARE_LOCAL_CLASS(PlTlConfig );
  ```

  class PlTlConfig (-)

  PlTlConfig is the base class for all configuration files. It provides protocol for loading and saving configuration from the given file.

- getConfig

  ```
  virtual int getConfig();
  ```

  Get any config info that I am storing which might have changed.

- operator =

  ```
  PlTlConfig& operator =(const PlTlConfig& orig);
  ```

  Assigment operator

- saveConfig

  ```
  virtual int saveConfig(ostream& str);
  ```

  Save my current config info to the given stream.

- save

  ```
  virtual int save(const char* fn);
  ```

  Save the configuration file. First get any config info which might have changed, then save the current values of my config info, along with the appropriate keywords to the given file. Overrides ConfigurationFile::save()

6.3.13.3.7 PlTlDefsConfig Class

Overview:

class PlTlDefsConfig (-)

Instances of PlTlDefsConfig load and save basic timeline configuration files.

Export Control: Public

Inheritance Relationships:

Inherits from PlTlConfig

Attributes:

`myHandleColor: HString`

My handle colorname

`myHeight: int`

My height

`myHorScrollInc: double`

My horizontal scroll increment

`myHorZoomInc: double`

My horizontal zoom increment

`myIntvlInit: HString`

My initial display interval string

`myIntvlOpts: HObjList`

My list of interval options for display

`myLabelColor: HString`

My label colorname

`myLblFontFamily: HString`

My label font family

`myLblFontSize: int`

My label font size

`myLblFontStyle: HString`

My label font style

`myLblRegColor: HString`

My label region colorname

`myLblRegWidth: int`

My label region width

`myMainRegColor: HString`

My main region colorname

`myOvrsFactor: double`

My oversubscription display factor

myScFontFamily: HString

My scalable font family

myScFontInc: int

My scalable font increment

myScFontMaxSize: int

My scalable font maximum size

myScFontMinSize: int

My scalable font minimum size

myScFontStyle: HString

My scalable font style

myScrollRegColor: HString

My scroll region colorname

myScrollRegWidth: int

My scrool region width

mySubRegsHeight: int

My subregions height

myTextColor: HString

My text colorname

myTsRegColor: HString

My timescale region colorname

myTsRegHeight: int

My timescale region height

myWidth: int

My width

myXtl: Xtl*

My owning xtl

myZbRegColor: HString

My zoombar region colorname

myZbRegHeight: int

My zoombar height

Constructors and Destructor:

```
PlTlDefsConfig(Xtl* );
```

Constructor. Argument is parent Xtl

```
PlTlDefsConfig(const PlTlDefsConfig& orig);
```

Copy constructor

```
virtual  ~PlTlDefsConfig();
```

Destructor

Operations:

- configure

```
virtual int configure(int argc, char** argv);
```

Configure

- configure

```
virtual int configure();
```

Configure

- configure

```
virtual int configure(const char* configInfo);
```

Configure. Override to empty any existing collection of interval options in preparation to read in new ones.

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlTlDefsConfig );
```

class PlTlDefsConfig (-)

Instances of PlTlDefsConfig load and save basic timeline configuration files.

- getConfig

```
virtual int getConfig();
```

Get any config info that I am storing which might have changed. Overrides PlTlConfig::getConfig()

- handleColor

```
virtual const HString& handleColor() const;
```

Get my handle colorname

- handleKeyword

  ```
  virtual int handleKeyword(const HString& kywd);
  ```

  Handle keyword. Look in the file for the specific keywords associated with my variables and set my variables to the following values. Overrides ConfigurationFile::handleKeyword()

- height

  ```
  virtual int height() const;
  ```

  Get my height

- horScrollInc

  ```
  virtual double horScrollInc() const;
  ```

  Get my horizontal scroll increments

- horZoomInc

  ```
  virtual double horZoomInc() const;
  ```

  Get my horizontal zoom increments

- intvlInit

  ```
  virtual const HString& intvlInit() const;
  ```

  Get my display interval initial string

- intvlOpts

  ```
  virtual HObjCollection& intvlOpts();
  ```

  Get my collection of display interval options

- labelColor

  ```
  virtual const HString& labelColor() const;
  ```

  Get my label colorname

- lblFontFamily

  ```
  virtual const HString& lblFontFamily() const;
  ```

  Get my label font family

- lblFontSize

  ```
  virtual int lblFontSize() const;
  ```

  Get my label font size

- lblFontStyle

  ```
  virtual const HString& lblFontStyle() const;
  ```

Get my label font style

- lblRegColor

```
virtual const HString& lblRegColor() const;
```

Get my label region colorname

- lblRegWidth

```
virtual int lblRegWidth() const;
```

Get my label region width

- mainRegColor

```
virtual const HString& mainRegColor() const;
```

Get my main region colorname

- operator =

```
PlTlDefsConfig& operator =(const PlTlDefsConfig& orig);
```

Assignment operator

- ovrsFactor

```
virtual double ovrsFactor() const;
```

Get my oversubscription display factor for graphs.

- saveConfig

```
virtual int saveConfig(ostream& str);
```

Save my current config info to the given stream.

- scFontFamily

```
virtual const HString& scFontFamily() const;
```

Get my scalable font family

- scFontInc

```
virtual int scFontInc() const;
```

Get my scalable font increment

- scFontMaxSize

```
virtual int scFontMaxSize() const;
```

Get my scalable font maximum size

- scFontMinSize

```
virtual int scFontMinSize() const;
```

Get my scalable font minimum size

- scFontStyle

```
virtual const HString& scFontStyle() const;
```

Get my scalable font style

- scrollRegColor

```
virtual const HString& scrollRegColor() const;
```

Get my scroll region colorname

- scrollRegWidth

```
virtual int scrollRegWidth() const;
```

Get my scroll region width

- subRegsHeight

```
virtual int subRegsHeight() const;
```

Get my subregions height

- textColor

```
virtual const HString& textColor() const;
```

Get my text colorname

- tsRegColor

```
virtual const HString& tsRegColor() const;
```

Get my timescale region colorname

- tsRegHeight

```
virtual int tsRegHeight() const;
```

Get my timescale region height

- width

```
virtual int width() const;
```

Get my width

- zbRegColor

```
virtual const HString& zbRegColor() const;
```

Get my timeline region colorname

- zbRegHeight

```
virtual int zbRegHeight() const;
```

Get my zoombar region height

6.3.13.3.8 PlTlFileDiag Class

Overview:

class PlTlFileDiag (-)

Instances of PlTlFileDiag are base classe dialogs for handling the loading and saving of configuration files.

Export Control: Public

Inheritance Relationships:

Inherits from PlTlFileSelDiag

Attributes:

```
myAppl: PlTlAppl*
```

My timeline application

```
myCreatorGone: int
```

Flag to indicate that the creator has been destroyed.

```
myCreator: PlTlPanel*
```

My creator

Constructors and Destructor:

```
PlTlFileDiag(PlTlPanel* creator, PlTlAppl* appl);
```

Constructor. Arguments are my creator and the application.

```
PlTlFileDiag(const PlTlFileDiag& orig);
```

Copy constructor

```
virtual  ~PlTlFileDiag();
```

Destructor

Operations:

- cancel

```
virtual void cancel(XtPointer callData);
```

Cancel button callback

- creatorGone

  ```
  virtual void creatorGone(int tf);
  ```

  Set if my creator is destroyed.

- DECLARE_LOCAL_CLASS

  ```
  int DECLARE_LOCAL_CLASS(PlTlFileDiag );
  ```

  class PlTlFileDiag (-)

  Instances of PlTlFileDiag are base classe dialogs for handling the loading and saving of configuration files.

- operator =

  ```
  PlTlFileDiag& operator =(const PlTlFileDiag& orig);
  ```

  Assignment operator

- show

  ```
  virtual void show();
  ```

  Show. Overridden to set the pattern and the initial selection

6.3.13.3.9 PlTlFileSelDiag Class

Overview:

class PlTlFileSelDiag (FileSelectionBox widget with parent dialogue shell)

This is a derived DFileSelectBox that automatically creates a FileSelectionDialog (i.e. both the FileSelectionBox widget and its DialogShell parent are created no matter what create function is called.) In addition, this class adds behavior to make sure the DialogShell parent behaves just like a DPopup.

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
PlTlFileSelDiag();
```

Default constructor.

```
PlTlFileSelDiag(const PlTlFileSelDiag& orig);
```

Copy constructor

```
virtual  ~PlTlFileSelDiag();
```

Destructor

Operations:

- catchCloseEvent

```
virtual void catchCloseEvent();
```

Callback to intercept the window manager close event on the DialogShell.

- createFileSelectionBox

```
virtual int createFileSelectionBox(const char* name, DWidget*
parent, Arg* arglist, Cardinal argcount, int manage);
```

Create file selection box.

- create

```
virtual int create(const char* name, DWidget* parent);
```

Create routine. Overridden to make sure a FileSelectionDialog is actually created in addition to the FileSelectionBox widget.

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlTlFileSelDiag );
```

class PlTlFileSelDiag (FileSelectionBox widget with parent dialogue shell)

This is a derived DFileSelectBox that automatically creates a FileSelectionDialog (i.e. both the FileSelectionBox widget and its DialogShell parent are created no matter what create function is called.) In addition, this class adds behavior to make sure the DialogShell parent behaves just like a DPopup.

- hide

```
virtual void hide();
```

Hide me

- operator =

```
PlTlFileSelDiag& operator =(const PlTlFileSelDiag& orig);
```

Assignment operator

- show

```
virtual void show();
```

Show me

- winManagerClose

```
static void winManagerClose(Widget protocolWidget, XtPointer
clientData, XtPointer callData);
```

Static callback for window manager close events.

6.3.13.3.10 PlTlLoadDiag Class

Overview:

class PlTlLoadDiag (-)

Instances of PlTlLoadDiag are dialogs for loading saved timeline configuration files.

Export Control: Public

Inheritance Relationships:

Inherits from PlTlFileDiag

Attributes:

Constructors and Destructor:

```
PlTlLoadDiag(PlTlPanel* creator, PlTlAppl* appl);
```

Constructor. Arguments are my creator and the application.

```
PlTlLoadDiag(const PlTlLoadDiag& orig);
```

Copy constructor

```
virtual  ~PlTlLoadDiag();
```

Destructor

Operations:

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlTlLoadDiag );
```

class PlTlLoadDiag (-)

Instances of PlTlLoadDiag are dialogs for loading saved timeline configuration files.

- ok

```
virtual void ok(XtPointer callData);
```

Ok button callback. Tell the ptl appl to load and switch to a new configuration.

- operator =

  `PlTlLoadDiag& operator =(const PlTlLoadDiag& orig);`

  Assignment operator

6.3.13.3.11 PlTlMainRegion Class

Overview:

class PlTlMainRegion (-)

A PlTlMainRegion is a _TLMainRegion_ that implements "timeline" mouse tracking behavior. As part of this, it adds the functionality of displaying the current activity region that the mouse cursor is on.

Export Control: Public

Inheritance Relationships:

Attributes:

`mouseX: unsigned long`

My mouse X position

`myTLEvent: TLEvent*`

My TLEvent

`myTs: TLTimeScaleAbs*`

My TLTimeScaleAbs timescale

`myXtl: Xtl*`

My owning xtl

Constructors and Destructor:

`PlTlMainRegion(const PlTlMainRegion& orig);`

Copy constructor

`PlTlMainRegion(Xtl* , int x, int y, int w, int h);`

Constructor. Arguments are parent Xtl and dimensions.

`PlTlMainRegion(Xtl* );`

Constructor. Argument is parent Xtl.

```
virtual  ~PlTlMainRegion();
```

Destructor

Operations:

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlTlMainRegion );
```

class PlTlMainRegion (-)

A PlTlMainRegion is a _TLMainRegion_ that implements "timeline" mouse tracking behavior. As part of this, it adds the functionality of displaying the current activity region that the mouse cursor is on.

- handleMouseTrack

```
virtual void handleMouseTrack(Event* );
```

Mouse track handling

- handleSimpleEvent

```
virtual const HString handleSimpleEvent() const;
```

Get TLSimpleEvent info to be displayed

- operator =

```
PlTlMainRegion& operator =(const PlTlMainRegion& orig);
```

Assignment operator

- setHandlerEvents

```
virtual void setHandlerEvents();
```

Set up events which I will catch.

6.3.13.3.12 PlTlPanel Class

Overview:

class PlTlPanel (specialized TLPanel to add additional GUI)

This class, which is derived from TLPanel, adds capabilities for interactively changing colors and resources, and for saving and loading from different configuration files.

Export Control: Public

Inheritance Relationships:

Attributes:

`myAppl: PlTlAppl*`

My timeline application

`myColorChanger: PlTlColChngr*`

My color changer

`myDisplayMenu: DPulldownMenu*`

My display menu

`myFileMenu: DPulldownMenu*`

My file menu

`myLoadDiag: PlTlLoadDiag*`

My load config dialog

`myPlanButtonList: HObjList`

My list of plan menu buttons

`myPlansMenu: DPulldownMenu*`

My plans menu

`myPlanWinChgr: PlTlPlanWinChgr*`

My plan window changer

`myRsEditor: PlTlRsEditor*`

My resource editor

`mySaveDiag: PlTlSaveDiag*`

My save config dialog

`myTimeMenu: DPulldownMenu*`

My time menu

Constructors and Destructor:

`PlTlPanel(PlTlAppl* appl, Xtl* xtl);`

Constructor. Arguments are the application and the Xtl.

`PlTlPanel(const PlTlPanel& orig);`

Copy constructor

`virtual  ~PlTlPanel();`

Destructor.

Operations:

- addToPlansMenu

  ```
  virtual int addToPlansMenu();
  ```

  Add a button to my Plans menu for the given plan name

- clearPlansMenu

  ```
  virtual int clearPlansMenu();
  ```

  Clear all buttons out of my Plans menu

- colorChangerGone

  ```
  virtual void colorChangerGone();
  ```

  Set when the color changer is closed

- colorChanger

  ```
  virtual void colorChanger(PlTlPanelCbPushBtn* );
  ```

  Create color changer popup. Callback for display menu button item.

- createColorChanger

  ```
  virtual PlTlColChngr* createColorChanger();
  ```

  Create the color changer

- createDisplayMenuItems

  ```
  virtual int createDisplayMenuItems();
  ```

  Create the push buttons on the display menu.

- createDisplayMenu

  ```
  virtual int createDisplayMenu();
  ```

  Create my display button.

- createFileMenuItems

  ```
  virtual int createFileMenuItems();
  ```

  Create the push buttons on the configuration menu.

- createFileMenu

  ```
  virtual int createFileMenu();
  ```

  Create my configuration menu.

- createLoadDiag

```
virtual PlTlLoadDiag* createLoadDiag();
```

Create the load config dialog

- createMenus

```
virtual int createMenus();
```

Create panel menus.

- createPlansMenuItems

```
virtual int createPlansMenuItems();
```

Create the initial push buttons on the plans menu.

- createPlansMenu

```
virtual int createPlansMenu();
```

Create my plans menu

- createPlanWinChgr

```
virtual PlTlPlanWinChgr* createPlanWinChgr();
```

Create the plan window changer

- createRsEditor

```
virtual PlTlRsEditor* createRsEditor();
```

Create the resource editor

- createSaveDiag

```
virtual PlTlSaveDiag* createSaveDiag();
```

Create the save config dialog

- createTimeMenuItems

```
virtual int createTimeMenuItems();
```

Create the push buttons on the time menu

- createTimeMenu

```
virtual int createTimeMenu();
```

Create my time menu

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlTlPanel );
```

class PlTlPanel (specialized TLPanel to add additional GUI)

This class, which is derived from TLPanel, adds capabilities for interactively changing colors and resources, and for saving and loading from different configuration files.

- loadConfig

  ```
  virtual void loadConfig(PlTlPanelCbPushBtn* );
  ```

  Load a configuration. Callback for file menu button item.

- loadDiagGone

  ```
  virtual void loadDiagGone();
  ```

  Set when the load config dialog is closed

- operator =

  ```
  PlTlPanel& operator =(const PlTlPanel& orig);
  ```

  Assignment opertor

- planSnapper

  ```
  virtual void planSnapper(PlTlPanelCbPushBtn* );
  ```

  Handle requested plan snap. Callback for plans menu button items.

- planWinChanger

  ```
  virtual void planWinChanger(PlTlPanelCbPushBtn* );
  ```

  Create plan window changer popup. Callback for time menu button item.

- planWinChgrGone

  ```
  virtual void planWinChgrGone();
  ```

  Set when my plan window changer is gone

- printTimeline

  ```
  virtual void printTimeline(PlTlPanelCbPushBtn* );
  ```

  Print the timeline by spawning a postscript timeline. Callback for file menu button item.

- reconfigure

  ```
  virtual int reconfigure();
  ```

  Reconfigure. Notify any of my displays.

- remFromPlansMenu

  ```
  virtual int remFromPlansMenu();
  ```

  Remove the named button from the Plans menu. Returns TRUE if the button was found AND removed

- rsEditorGone

  `virtual void rsEditorGone();`

  Set when the resource editor is gone

- rsrcChanger

  `virtual void rsrcChanger(PlTlPanelCbPushBtn* );`

  Create resource changer popup. Callback for display menu button item.

- saveConfig

  `virtual void saveConfig(PlTlPanelCbPushBtn* );`

  Save the current configuration. Callback for file menu button item.

- saveDiagGone

  `virtual void saveDiagGone();`

  Set when the save config dialog is closed

- update

  `virtual int update();`

  Update. Notify my displays.

6.3.13.3.13 PlTlPlanWinChgr Class

Overview:

class PlTlPlanWinChgr (-)

Instances of PlTlPlanWinChgr are dialogs for selecting a plan.

Export Control: Public

Inheritance Relationships:

Attributes:

`myAppl: PlTlAppl*`

My PlTlXtl appl

`myCreatorGone: int`

Flag indicating the creator is closed

`myCreator: PlTlPanel*`

My creator panel

`myPresForm: DForm*`

My presentation area form

`myPtl: PlTlXtl*`

My PlTlXtl

`myStartTime: DDateTime*`

My start time

`myStopTime: DDateTime*`

My stop time

Constructors and Destructor:

`PlTlPlanWinChgr(PlTlPanel* creator, PlTlAppl* appl, PlTlXtl* PlTlXtl);`

Constructor. Arguments are the parent panel, the application , and Xtl.

`PlTlPlanWinChgr(const PlTlPlanWinChgr& orig);`

Copy constructor

`virtual  ~PlTlPlanWinChgr();`

Destructor

Operations:

- applyChanges

  `virtual void applyChanges();`

  Apply changes. Tell PlTlXtl to switch to the new total interval. Overrides DApplyDialog::applyChanges()

- createDispObjs

  `virtual int createDispObjs();`

  Create the display objects that make up my presentation area.

- createPresentationArea

  `virtual int createPresentationArea();`

  Make my presentation area. Overrides DApplyDialog::createPresentationArea()

- createStartTime

  `virtual int createStartTime();`

Create the start time

- createStopTime

```
virtual int createStopTime();
```

Create the stop time item

- creatorGone

```
virtual void creatorGone(int tf);
```

Set if creator is closed

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlTlPlanWinChgr );
```

class PlTlPlanWinChgr (-)

Instances of PlTlPlanWinChgr are dialogs for selecting a plan.

- manageAllChildren

```
virtual void manageAllChildren();
```

Ask all of pieces to manageChild.

- operator =

```
PlTlPlanWinChgr& operator =(const PlTlPlanWinChgr& orig);
```

Assignment operator

- reconfigure

```
virtual int reconfigure();
```

Reconfigure. Reinitialize the available and viewed lists

- setStartStopTimes

```
virtual int setStartStopTimes();
```

Initialize the start and stop times with the current total interval

- setStartTimePos

```
virtual void setStartTimePos();
```

Set the position of the start time item

- setStopTimePos

```
virtual void setStopTimePos();
```

Set the position of the stop time item

- show

```
virtual void show();
```

Show me. Override to set start and stop times to current total interval.

6.3.13.3.14 PlTlRsConfig Class

Overview:

class PlTlRsConfig (-)

Instances of PlTlRsConfig are responsible for loading a set of names of resources to be displayed on the timeline.

Export Control: Public

Inheritance Relationships:

Inherits from PlTlConfig

Attributes:

```
myAppl: PlTlAppl*
```

My owning PlTlAppl

Constructors and Destructor:

```
PlTlRsConfig(PlTlAppl* );
```

Constructor. Argument is owning PlTlAppl

```
PlTlRsConfig(const PlTlRsConfig& orig);
```

Copy constructor

```
virtual  ~PlTlRsConfig();
```

Destructor

Operations:

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlTlRsConfig );
```

class PlTlRsConfig (-)

Instances of PlTlRsConfig are responsible for loading a set of names of resources to be displayed on the timeline.

- handleKeyword

```
virtual int handleKeyword(const HString& kywd);
```

Handle keyword. For each resource or resource type in the config file, ask the SRM client noid to get the appropriate resource(s) from the resource allocator.

- handleRsType

  `virtual int handleRsType();`

  Handle a resource type in the config file

- handleRs

  `virtual int handleRs();`

  Handle an individual resource in the config file

- operator =

  `PlTlRsConfig& operator =(const PlTlRsConfig& orig);`

  Assignment operator

- saveConfig

  `virtual int saveConfig(ostream& str);`

  Save the current set of resources to the given stream. Overrides PlTlConfig::saveConfig()

6.3.13.3.15 PlTlRsEditor Class

Overview:

class PlTlRsEditor (-)

Instances of PlTlRsEditor are dialogs for choosing the set of resources to be displayed on the timeline.

Export Control: Public

Inheritance Relationships:

Attributes:

`myAddBtn: PlTlRsEditorCbPushBtn*`

My add button

`myAppl: PlTlAppl*`

My ptl appl

`myAvailLbl: DLabel*`

My available list label

myAvailList: PlTlRsEditorCbList*

My available list

myCreatorGone: int

Flag indicating if my creator has been closed

myCreator: PlTlPanel*

My creator panel

myDelBtn: PlTlRsEditorCbPushBtn*

My delete button

myDownBtn: PlTlRsEditorCbArrowBtn*

My down button

myPresForm: DForm*

My presentation area form

myUpBtn: PlTlRsEditorCbArrowBtn*

My up button

myViewedLbl: DLabel*

My viewed list label

myViewedList: PlTlRsEditorCbList*

My viewed list

Constructors and Destructor:

PlTlRsEditor(PlTlPanel* creator, PlTlAppl* appl);

Constructor. Arguments are my parent panel and the application

PlTlRsEditor(const PlTlRsEditor& orig);

Copy constructor

virtual ~PlTlRsEditor();

Destructor

Operations:

- addCb

virtual void addCb(PlTlRsEditorCbPushBtn*);

Callback for add button

- applyChanges

  `virtual void applyChanges();`

  Apply changes. Tell the application to switch resources.

- availListSelCb

  `virtual void availListSelCb(PlTlRsEditorCbList* );`

  Select callback for available resource list

- createAddBtn

  `virtual int createAddBtn();`

  Create the add-to-viewed resource button

- createAvailLbl

  `virtual int createAvailLbl();`

  Create the label for the available resource list

- createAvailList

  `virtual int createAvailList();`

  Create the available resource list

- createBtns

  `virtual int createBtns();`

  Create the control buttons

- createDelBtn

  `virtual int createDelBtn();`

  Create the delete-from-viewed resource button

- createDispObjs

  `virtual int createDispObjs();`

  Create the display objects that make up my presentation area.

- createDownBtn

  `virtual int createDownBtn();`

  Create the move-down in the viewed list button

- createPresentationArea

  `virtual int createPresentationArea();`

  create my presentation area.

- createUpBtn

  ```
  virtual int createUpBtn();
  ```

  Create the move-up in the viewed list button

- createViewedLbl

  ```
  virtual int createViewedLbl();
  ```

  create the label for the viewed resource list

- createViewedList

  ```
  virtual int createViewedList();
  ```

  Create the viewed resource list

- creatorGone

  ```
  virtual void creatorGone(int tf);
  ```

  Set if my creator is closed

- DECLARE_LOCAL_CLASS

  ```
  int DECLARE_LOCAL_CLASS(PlTlRsEditor );
  ```

  class PlTlRsEditor (-)

  Instances of PlTlRsEditor are dialogs for choosing the set of resources to be displayed on the timeline.

- delCb

  ```
  virtual void delCb(PlTlRsEditorCbPushBtn* );
  ```

  Callback for delete button

- downCb

  ```
  virtual void downCb(PlTlRsEditorCbArrowBtn* );
  ```

  Callback for down button

- initAvailList

  ```
  virtual int initAvailList();
  ```

  Initialize the list with all the resources available

- initViewedList

  ```
  virtual int initViewedList();
  ```

  Initialize the list with the current resources being viewed

- manageAllChildren

```
virtual void manageAllChildren();
```

Ask all of pieces to manageChild.

- operator =

```
PlTlRsEditor& operator =(const PlTlRsEditor& orig);
```

Assignment operator

- reconfigure

```
virtual int reconfigure();
```

Reconfigure. Reinitialize the available and viewed lists

- setAddBtnPos

```
virtual void setAddBtnPos();
```

Set the add button position

- setAvailLblPos

```
virtual void setAvailLblPos();
```

Set the position of available list label

- setAvailListPos

```
virtual void setAvailListPos();
```

Set the postion of the available list

- setDelBtnPos

```
virtual void setDelBtnPos();
```

Set the delete button position

- setDownBtnPos

```
virtual void setDownBtnPos();
```

Set the down button position

- setUpBtnPos

```
virtual void setUpBtnPos();
```

Set the up button position

- setViewedLblPos

```
virtual void setViewedLblPos();
```

Set the position of viewed list label

- setViewedListPos

```
virtual void setViewedListPos();
```

Set the position of the viewed list

- show

```
virtual void show();
```

Show me. Override to initialize viewed list.

- upCb

```
virtual void upCb(PlTlRsEditorCbArrowBtn* );
```

Callback for up button

- viewedListSelCb

```
virtual void viewedListSelCb(PlTlRsEditorCbList* );
```

Select callback for viewed resource list

6.3.13.3.16 PlTlSaveDiag Class

Overview:

class PlTlSaveDiag (-)

Instances of PlTlSaveDiag are dialogs for saving timeline configuration files.

Export Control: Public

Inheritance Relationships:

Inherits from PlTlFileDiag

Attributes:

Constructors and Destructor:

```
PlTlSaveDiag(PlTlPanel* creator, PlTlAppl* appl);
```

Constructor. Arguments are my creator and the appl.

```
PlTlSaveDiag(const PlTlSaveDiag& orig);
```

Copy constructor

```
virtual  ~PlTlSaveDiag();
```

Destructor

Operations:

- DECLARE_LOCAL_CLASS

  ```
  int DECLARE_LOCAL_CLASS(PlTlSaveDiag );
  ```

  class PlTlSaveDiag (-)

  Instances of PlTlSaveDiag are dialogs for saving timeline configuration files.

- ok

  ```
  virtual void ok(XtPointer callData);
  ```

  Ok button callback. Tell the ptl appl to save configuration

- operator =

  ```
  PlTlSaveDiag& operator =(const PlTlSaveDiag& orig);
  ```

  Assignment operator

6.3.13.3.17 PlTlSimpleEvent Class

Overview:

class PlTlSimpleEvent (-)

Instances of PlTlSimpleEvent provide base class behavior that all timeline simple events will provide. Derived PlTlSimpleEvent(s) have a resource state, know how to determine that state, determine if it is a valid state and can set the type of label to display on the timeline.

Export Control: Public

Inheritance Relationships:

Attributes:

```
myState: RRsState*
```

My associated state

Constructors and Destructor:

```
PlTlSimpleEvent(Xtl* xtl);
```

Constructor. Argument is the Xtl

```
PlTlSimpleEvent(const PlTlSimpleEvent& orig);
```

Copy constructor

```
virtual  ~PlTlSimpleEvent();
```

destructor

Operations:

- createView

  ```
  virtual TLEventView* createView(TLEventSubRegion* );
  ```

  Called when a new view is needed (for the given subregion).

- DECLARE_LOCAL_CLASS

  ```
  int DECLARE_LOCAL_CLASS(PlTlSimpleEvent );
  ```

  class PlTlSimpleEvent (-)

  Instances of PlTlSimpleEvent provide base class behavior that all timeline simple events will provide. Derived PlTlSimpleEvent(s) have a resource state, know how to determine that state, determine if it is a valid state and can set the type of label to display on the timeline.

- isValidState

  ```
  virtual int isValidState(RRsState* );
  ```

  Return TRUE if the given RRsState is one I will accept.

- operator =

  ```
  PlTlSimpleEvent& operator =(const PlTlSimpleEvent& orig);
  ```

  Assignment operator

- overlap

  ```
  virtual int overlap(const char* rsrcNm, const HEpochInterval&
  intvl);
  ```

  Return TRUE if I belong to the specified resource and overlap the given interval

- rsrcNm

  ```
  virtual const HString& rsrcNm() const;
  ```

  Get my associated resource name, which corresponds to the name of my subregion

- rsrcNm

  ```
  virtual void rsrcNm(const char* );
  ```

  Set my associated resource name, which corresponds to the name of my subregion

- setBackground

  ```
  virtual int setBackground(TLStipple* stipple);
  ```

  Set background pattern

- setLabelAndType

  `virtual void setLabelAndType();`

  Set my label and type according to myState

- state

  `virtual RRsState* state() const;`

  Get my associated state

- state

  `virtual int state(RRsState* );`

  Set my associated state and determine label and type for this event. If state is accepted by event, return TRUE; If state is not accepted by event, returns FALSE;

6.3.13.3.18 PlTlSliderPup Class

Overview:

class PlTlSliderPup (-)

Instances of PlTlSliderPup are popup windows which display a start time and stop time for the corresponding plan region.

Export Control: Public

Inheritance Relationships:

Attributes:

`myEndLabel: DLabel*`

My end time label

`myRowColumn: DRowColumn*`

My row-column manager widget

`myStartLabel: DLabel*`

My start time label

Constructors and Destructor:

`PlTlSliderPup();`

Default constructor

`PlTlSliderPup(const PlTlSliderPup& orig);`

Copy constructor

```
virtual  ~PlTlSliderPup();
```

Destructor

Operations:

- createDisplay

```
virtual int createDisplay();
```

Create my display.

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlTlSliderPup );
```

class PlTlSliderPup (-)

Instances of PlTlSliderPup are popup windows which display a start time and stop time for the corresponding plan region.

- manageAllChildren

```
virtual void manageAllChildren();
```

Manage all of my pieces

- operator =

```
PlTlSliderPup& operator =(const PlTlSliderPup& orig);
```

Assignment operator

- setEnd

```
virtual void setEnd(const HEpochTime& time);
```

Set end time to the given epoch time

- setStart

```
virtual void setStart(const HEpochTime& time);
```

Set start time to the given epoch time

6.3.13.3.19 PlTlSrmNoid Class

Overview:

class PlTlSrmNoid (-)

Instances of PlTlSrmNoid are timeline resource model client noids. A PlTlSrmNoid knows how to establish a connection with a resource model, retrieve initial information and is notified when new updates are received by the noids agent

Export Control: Public

Inheritance Relationships:

Inherits from PlClSrmNoid

Attributes:

```
myParent: PlTlAppl*
```

My parent appl

Constructors and Destructor:

```
PlTlSrmNoid(const PlTlSrmNoid& orig);
```

Copy constructor

```
PlTlSrmNoid(PlTlAppl* appl);
```

Constructor specifying parent

```
PlTlSrmNoid();
```

Default constructor

```
virtual  ~PlTlSrmNoid();
```

Destructor

Operations:

- connEstablished

```
virtual int connEstablished();
```

Notification connection established for first time. overridden from PlClSrmNoid::connEstablished()

- connReestablished

```
virtual int connReestablished();
```

Notification reestablished after being lost

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlTlSrmNoid );
```

class PlTlSrmNoid (-)

Instances of PlTlSrmNoid are timeline resource model client noids. A PlTlSrmNoid knows how to establish a connection with a resource model, retrieve initial information and is notified when new updates are received by the noids agent

- handlePlanDel

```
virtual int handlePlanDel(HMsgConn* curCon, HAddress* curAdr,
SMsgPlanDel* curMsg);
```

A plan has been deleted, inform my parent of this.

- handlePlanNew

```
virtual int handlePlanNew(HMsgConn* curCon, HAddress* curAdr,
SMsgPlanNew* curMsg);
```

A new plan has been created. Give the name to my parent.

- handlePlansGetRsp

```
virtual int handlePlansGetRsp(HMsgConn* curCon, HAddress*
curAdr, PlClMsgGetPlansRsp* curMsg);
```

Handle getting plans into our local plan pool and then getting a separate list of plan names

- handlePlanSnap

```
virtual int handlePlanSnap(HMsgConn* curCon, HAddress* curAdr,
SMsgPlanSnap* curMsg);
```

Ask my parent appl to handle a request to snap to a new plan.

- handlePlClMsgPlanNew

```
virtual int handlePlClMsgPlanNew(HMsgConn* con, HAddress* adr,
PlClMsgPlanNew* msg);
```

Handle derived new plan messages.

- handleRsGetNmsRsp

```
virtual int handleRsGetNmsRsp(HMsgConn* curCon, HAddress*
curAdr, PlClMsgGetRsNmsRsp* curMsg);
```

Turn over the list of resource names to my parent. Overrides PlClSrmNoid::handleRsGetNmsRsp()

- installRsGetRs

```
virtual int installRsGetRs(RResource* rs);
```

Tell my parent about a resource that has just been loaded into my resource pool.

- installRsState

```
virtual int installRsState(RUpdState* anUpdate);
```

Tell my parent about the state update I just received. Overrides PlClSrmNoid::installRsState()

- operator =

```
PlTlSrmNoid& operator =(const PlTlSrmNoid& orig);
```

Assignment operator

- parent

  `PlTlAppl* parent() const;`

  Get my parent appl

- parent

  `void parent(PlTlAppl* appl);`

  Set my parent appl

6.3.13.3.20 PlTlStatusBar Class

Overview:

class PlTlStatusBar (-)

Instances of PlTlStatusBar provide a line at the bottom of the timeline for displaying messages.

Export Control: Public

Inheritance Relationships:

Attributes:

`myDisplayLabel: DLabel*`

My display label

`myXtl: Xtl*`

My parent xtl

Constructors and Destructor:

`PlTlStatusBar(Xtl* );`

Constructor. Argument is a pointer to my parent Xtl

`PlTlStatusBar(const PlTlStatusBar& orig);`

Copy constructor

`virtual ~PlTlStatusBar();`

Destructor

Operations:

- clear

```
virtual void clear();
```

Clear.

- createMyLabel

```
virtual int createMyLabel();
```

Create status display label

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlTlStatusBar );
```

class PlTlStatusBar (-)

Instances of PlTlStatusBar provide a line at the bottom of the timeline for displaying messages.

- displayMesg

```
virtual void displayMesg(const HString& mesg);
```

Display received message

- init

```
virtual int init();
```

Initialize. Create the form and its child, DLabel

- operator =

```
PlTlStatusBar& operator =(const PlTlStatusBar& orig);
```

Assignment operator

- positionWidgets

```
virtual int positionWidgets();
```

Position widgets

6.3.13.3.21 PlTlStplConfig Class

Overview:

class PlTlStplConfig (-)

Instances of PlTlStplConfig handle stipple keywords in configuration files.

Export Control: Public

Inheritance Relationships:

Inherits from PlTlConfig

Attributes:

`myXtl: Xtl*`

My owning xtl

Constructors and Destructor:

`PlTlStplConfig(Xtl* );`

Constructor. Argument is parent Xtl

`PlTlStplConfig(const PlTlStplConfig& orig);`

Copy constructor

`virtual  ~PlTlStplConfig();`

Destructor

Operations:

- DECLARE_LOCAL_CLASS

  `int DECLARE_LOCAL_CLASS(PlTlStplConfig );`

  class PlTlStplConfig (-)

  Instances of PlTlStplConfig handle stipple keywords in configuration files.

- handleKeyword

  `virtual int handleKeyword(const HString& kywd);`

  Process "stipple" keyword

- handleType

  `virtual int handleType();`

  Load TLTypes with type name and TLStplMapper with the type name and associated bitmap file.

- operator =

  `PlTlStplConfig& operator =(const PlTlStplConfig& orig);`

  Assignment operator

- saveConfig

  `virtual int saveConfig(ostream& str);`

  Save my current config info to the given stream

6.3.13.3.22 PlTlStplEvent Class

Overview:

class PlTlStplEvent (-)

Instances of PlTlStplEvent are stipple events associated with resource states.

Export Control: Public

Inheritance Relationships:

Inherits from PlTlStplEv

Attributes:

`myState: RRsState*`

My associated state

Constructors and Destructor:

`PlTlStplEvent(Xtl* xtl);`

Constructor. Argument is parent Xtl.

`PlTlStplEvent(const PlTlStplEvent& orig);`

Copy constructor

`virtual ~PlTlStplEvent();`

Destructor

Operations:

- createView

  `virtual TLEventView* createView(TLEventSubRegion* );`

  Create an LtlStplView

- DECLARE_LOCAL_CLASS

  `int DECLARE_LOCAL_CLASS(PlTlStplEvent );`

  class PlTlStplEvent (-)

  Instances of PlTlStplEvent are stipple events associated with resource states.

- isValidState

  `virtual int isValidState(RRsState* );`

  Return TRUE if the given RRsState is one I will accept.

- operator =

  ```
  PlTlStplEvent& operator =(const PlTlStplEvent& orig);
  ```

  Assignment operator

- overlap

  ```
  virtual int overlap(const char* rsrcNm, const HEpochInterval&
  intvl);
  ```

  Return TRUE if I belong to the specified resource and overlap the given interval

- rsrcNm

  ```
  virtual void rsrcNm(const char* );
  ```

  Set my associated resource name, which corresponds to the name of my subregion

- rsrcNm

  ```
  virtual const HString& rsrcNm() const;
  ```

  Get my associated resource name, which corresponds to the name of my subregion

- setLabelAndType

  ```
  virtual void setLabelAndType();
  ```

  Set my label and type according to myState

- state

  ```
  virtual RRsState* state() const;
  ```

  Get my associated state

- state

  ```
  virtual int state(RRsState* );
  ```

  Set my associated state and determine label and type for this event. If state is accepted by event, return TRUE; If state is not accepted by event, returns FALSE;

6.3.13.3.23 PlTlStplEv Class

Overview:

class PlTlStplEv (-)

Instances of PlTlStplEv are stipple events.

Export Control: Public

Inheritance Relationships:

Attributes:

`myStipple: TLStipple*`

My TLStipple containing the bitmap I am to draw

`ownStippleFlag: int`

Flag indicating whether I own the stipple or not. Default is that I DO NOT own the stipple.

Constructors and Destructor:

`PlTlStplEv(Xtl* xtl);`

Constructor. Argument is owning Xtl

`PlTlStplEv(const PlTlStplEv& orig);`

Copy constructor

`virtual  ~PlTlStplEv();`

Destructor

Operations:

- createView

  `virtual TLEventView* createView(TLEventSubRegion* );`

  Create a PlTlStplVw. Overrides TLEvent::createView

- DECLARE_LOCAL_CLASS

  `int DECLARE_LOCAL_CLASS(PlTlStplEv );`

  class PlTlStplEv (-)

  Instances of PlTlStplEv are stipple events.

- interval

  `virtual const HEpochInterval& interval() const;`

  Set my interval. set overridden to ensure that myIntvl is degenerate, i.e. of 0 duration, with time of intvl.start().

- interval

  `virtual void interval(const HEpochInterval& intvl);`

  Get my interval

- operator =

  ```
  PlTlStplEv& operator =(const PlTlStplEv& orig);
  ```

  Assigmnet operator

- ownStipple

  ```
  virtual void ownStipple(int tf);
  ```

  set whether or not I own the stipple. Default is that I do NOT own the stipple.

- ownStipple

  ```
  virtual int ownStipple() const;
  ```

  Get wheter or not I own the stipple.

- stipple

  ```
  virtual TLStipple* stipple();
  ```

  Get my stipple.

- stipple

  ```
  virtual void stipple(TLStipple* tlSt);
  ```

  Set my stipple. If ownStippleFlag is TRUE, I own stipple and will delete it. If ownStippleFlag is FALSE, I do not own stipple and will not delete it.

- subRegionNm

  ```
  virtual void subRegionNm(const char* );
  ```

  Set my associated subregion name

- subRegionNm

  ```
  virtual const HString& subRegionNm() const;
  ```

  Get my associated subregion name

- times

  ```
  virtual int times(HEpochTime& st, HEpochTime& en);
  ```

  Get my start and end time (which are the same).

- time

  ```
  virtual void time(const HEpochTime& tm);
  ```

  Set time

6.3.13.3.24 PlTlStplView Class

Overview:

class PlTlStplView (-)

Instances of PlTlStplView are views associated with stipple events.

Export Control: Public

Inheritance Relationships:

Inherits from PlTlStplVw

Attributes:

Constructors and Destructor:

```
PlTlStplView(PlTlStplEv* ev);
```

Constructor. Argument is my owning PlTlStplEv

```
PlTlStplView(const PlTlStplView& orig);
```

Copy constructor

```
virtual  ~PlTlStplView();
```

Destructor

Operations:

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlTlStplView );
```

class PlTlStplView (-)

Instances of PlTlStplView are views associated with stipple events.

- move

```
virtual int move();
```

Move me. Compute position on new subregion, then tell my event to move me.

- operator =

```
PlTlStplView& operator =(const PlTlStplView& orig);
```

Assignment operator

6.3.13.3.25 PlTlStplVw Class

Overview:

class PlTlStplVw (-)

Instances of PlTlStplVw are views of stipple events.

Export Control: Public

Inheritance Relationships:

Attributes:

`myIconRect: DXRectangle`

My current icon rectangle (this rectangle moves during a drag)

`myIconXHot: int`

The x location of the hot spot within my icon rect, relative to the upper left corner of the icon rect. The icon rect is placed so that the "hot spot" is at the exact time of the event.

`myIconYHot: int`

The y location of the hot spot within my icon rect, relative to the upper left corner of the icon rect

`myOldIconRect: DXRectangle`

My old icon rectangle, describing my last displayed position. Used to replace by current icon rectangle with my last if a drag fails.

`myStipple: TLStipple*`

My stipple (visibility only)

`myTextRect: DXRectangle`

My text rectangle, for displaying the associated text. this rectangle does not move during drags.

`myText: DXHString`

My associated text

Constructors and Destructor:

`PlTlStplVw(PlTlStplEv* ev);`

Constructor. Argument is my owning PlTlStplEv

`PlTlStplVw(const PlTlStplVw& orig);`

Copy constructor

```
virtual  ~PlTlStplVw();
```

Destructor

Operations:

- adjPos

```
virtual void adjPos(long dx, long dy);
```

Adjust position of icon rect to take into account movement as specified.

- containedBy

```
virtual unsigned int containedBy(HRectangle* ) const;
```

Return TRUE if I am contained by the given rectangle.

- contains

```
virtual unsigned int contains(HRectangle* ) const;
```

Return TRUE if I contain the give rectangle

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlTlStplVw );
```

class PlTlStplVw (-)

Instances of PlTlStplVw are views of stipple events.

- eraseOld

```
virtual void eraseOld();
```

Erase old view which is just the old icon rect plus the text.

- erase

```
virtual void erase();
```

Erase whole current view which is current icon rectangle plus the text.

- findEventRegion

```
virtual TLEventSubRegion* findEventRegion();
```

Find subregion I am on by looking at position of current icon rect.

- getPos

```
virtual void getPos();
```

Restore current icon rectangle to old icon rectangle.

- handleSelectOnSelectedEvent

```
virtual void handleSelectOnSelectedEvent();
```

Handle another select on an already selected event

- handleSelectOnUnselectedEvent

```
virtual void handleSelectOnUnselectedEvent();
```

Handle select on a currently unselected event

- height

```
virtual void height(int );
```

Set my height.

- intersects

```
virtual unsigned int intersects(unsigned long x, unsigned long
y) const;
```

Return TRUE if I intersect the given point

- intersects

```
virtual unsigned int intersects(const HRectangle* ) const;
```

Return TRUE if I intersect the given rectangle

- interval

```
virtual void interval(const HEpochInterval& intvl);
```

Set my interval.

- interval

```
virtual const HEpochInterval& interval();
```

Get my interval.

- moveInterval

```
virtual HEpochInterval moveInterval(TLEventSubRegion* tsr);
```

Returns the time interval (which is degenerate in this case) that corresponds to where my icon rect hot spot currently is

- operator =

```
PlTlStplVw& operator =(const PlTlStplVw& orig);
```

Assignment operator

- recalcIcon

```
virtual void recalcIcon(TLTimeScaleAbs* );
```

Recalculate the position of the icon rect and the location of the hot spot relative to the icon rect.

- recalcName

  ```
  virtual void recalcName();
  ```

  Recalculate the position of the text rect.

- recalc

  ```
  virtual void recalc(TLTimeScaleAbs* );
  ```

  Recalculate all of my display elements.

- repaintIcon

  ```
  virtual int repaintIcon();
  ```

  Repaint my icon

- repaintName

  ```
  virtual int repaintName();
  ```

  Repaint my text string

- repaintOutline

  ```
  virtual int repaintOutline();
  ```

  Repaint outline of current icon rectangle, (usually while dragging)

- repaintSelected

  ```
  virtual int repaintSelected();
  ```

  Repaint selected rectangle around icon

- repaint

  ```
  virtual int repaint();
  ```

  Repaint myself.

- savePos

  ```
  virtual void savePos();
  ```

  Save current icon rectangle in old icon rectangle.

- selectView

  ```
  virtual void selectView();
  ```

  Tell me to select. I set my selected flag and repaint, so I pick up selected rectangle around icon.

- select

```
virtual void select();
```

Handle select according to whether or not event is already selected.

- stipple

```
virtual void stipple(TLStipple* );
```

Set my stipple. I do NOT own it.

- stipple

```
virtual TLStipple* stipple() const;
```

Get my stipple. I do NOT own it.

- unselectView

```
virtual void unselectView();
```

Tell me to unselect. I reset my selected flag and repaint, so I get rid of my selected rectangle around icon.

- visInterval

```
virtual const HEpochInterval& visInterval();
```

Get my visable interval.

6.3.13.3.26 PlTlTypesConfig Class

Overview:

class PlTlTypesConfig (-)

Instances of PlTlTypesConfig handle configuration files which can contain class names which are added to the list of the timeline's known types.

Export Control: Public

Inheritance Relationships:

Inherits from PlTlConfig

Attributes:

```
myXtl: Xtl*
```

My owning xtl

Constructors and Destructor:

```
PlTlTypesConfig(Xtl* );
```

Constructor. Argument is owning Xtl.

```
PlTlTypesConfig(const PlTlTypesConfig& orig);
```

Copy constructor

```
virtual  ~PlTlTypesConfig();
```

Destructor

Operations:

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlTlTypesConfig );
```

class PlTlTypesConfig (-)

Instances of PlTlTypesConfig handle configuration files which can contain class names which are added to the list of the timeline's known types.

- handleKeyword

```
virtual int handleKeyword(const HString& kywd);
```

Handle keyword. Process "type" keyword

- handleType

```
virtual int handleType();
```

Load TLTypes with type name and TLColorMapper with the type name and associated color.

- operator =

```
PlTlTypesConfig& operator =(const PlTlTypesConfig& orig);
```

Assignment operator

- saveConfig

```
virtual int saveConfig(ostream& str);
```

Save my current config info to the given stream

6.3.13.3.27 PlTlValEvent Class

Overview:

class PlTlValEvent (timeline value event)

This class is a derived TLEvent which is used for displaying values versus time data graphically on a TLEventSubRegion over a specific interval of time. For example, the data may be displayed in histogram or graph form. There is only one instance of this class per subregion for the entire interval over which data is to be displayed.

The primary behavior this class adds is the val() member function, which puts a list of values to be displayed over a specified interval into a collection provided by the user. The exact nature of the objects put in the list depends on derived classes.

Export Control: Public

Inheritance Relationships:

Attributes:

`myIntvl: HEpochInterval`

My entire interval over which data is to be displayed

`myMaxVal: double`

The maximum values to be displayed

`myMinVal: double`

The mininum values to be displayed

`mySubRegionName: HString`

My subregion that I should be displayed on

`myThldVal: double`

The threshold values to be displayed

`myType: HString`

My type, used for filtering, color, etc.

Constructors and Destructor:

`PlTlValEvent(Xtl* );`

Constuctor. Argument is parent Xtl

`PlTlValEvent(const PlTlValEvent& );`

Copy constructor

`virtual  ~PlTlValEvent();`

Destructor

Operations:

- associate

`virtual void associate(TLEventView* , TLEventSubRegion* );`

Called to associate the subregion with the view Overrides TLEvent::associate()

- createView

  ```
  virtual TLEventView* createView(TLEventSubRegion* );
  ```

  Called when a new view is needed - behavior at this level is to return NULL, overriding TLEvent behavior to create TLBoxView. Each derived kind of PlTlValEvent needs to create a particular kind of view.

- DECLARE_LOCAL_CLASS

  ```
  int DECLARE_LOCAL_CLASS(PlTlValEvent );
  ```

  class PlTlValEvent (timeline value event)

  This class is a derived TLEvent which is used for displaying values versus time data graphically on a TLEventSubRegion over a specific interval of time. For example, the data may be displayed in histogram or graph form. There is only one instance of this class per subregion for the entire interval over which data is to be displayed.

  The primary behavior this class adds is the val() member function, which puts a list of values to be displayed over a specified interval into a collection provided by the user. The exact nature of the objects put in the list depends on derived classes.

- draggable

  ```
  virtual void draggable(unsigned int );
  ```

- draggable

  ```
  virtual unsigned int draggable() const;
  ```

  Set/get draggable status. Overridden to prohibit setting to TRUE. Base class behavior is for PlTlValEvents not to be draggable.

- interval

  ```
  virtual void interval(const HEpochInterval& );
  ```

  Get total interval over which I will display data

- interval

  ```
  virtual const HEpochInterval& interval() const;
  ```

  Set total interval over which I will display data

- isAppropriate

  ```
  virtual int isAppropriate(TLEventSubRegion* );
  ```

  Returns TRUE if subregion should have a view on me. Default behavior is to check subregion name against mySubRegionName

- maxVal

  ```
  virtual double maxVal() const;
  ```

  Get maximum value to be displayed

- maxVal

  ```
  virtual void maxVal(double );
  ```

  Set maximum value to be displayed

- minVal

  ```
  virtual void minVal(double );
  ```

  Set minimum value to be displayed

- minVal

  ```
  virtual double minVal() const;
  ```

  Get minimum value to be displayed

- name

  ```
  virtual const HString& name() const;
  ```

  Get name

- operator =

  ```
  PlTlValEvent& operator =(const PlTlValEvent& );
  ```

  Assignment operator

- subRegName

  ```
  virtual const HString& subRegName() const;
  ```

  Set name of subregion I should be displayed on

- subRegName

  ```
  virtual void subRegName(const char* );
  ```

  Get name of subregion I should be displayed on

- thldVal

  ```
  virtual double thldVal() const;
  ```

  Get threshold value to be displayed

- thldVal

  ```
  virtual void thldVal(double );
  ```

  Set threshold value to be displayed

- times

  `virtual int times(HEpochTime& st, HEpochTime& en);`

  Return start and end times of total interval over which I am displaying data.

- type

  `virtual const HString& type() const;`

  Set type for color mapping, etc.

- type

  `virtual void type(const char* );`

  Get type.

- updateVal

  `virtual int updateVal(const HEpochInterval& intvl);`

  Tell my view to update itself over the interval.

- val

  `virtual int val(const HEpochInterval& reqIntvl, HObjCollection& col);`

  Fill the collection with the data that is to be displayed over the requested interval. If data is put into collection, TRUE is returned.

6.3.13.3.28 PlTlValView Class

Overview:

class PlTlValView (value view)

This class is the base class for views that display value versus time information graphically in a timeline subregion.

Export Control: Public

Inheritance Relationships:

Attributes:

`myOvrThldColor: DRGBColor`

My color for displaying over-threshold data values

`myVisIntvl: HEpochInterval`

The currently visible portion of the total interval over which I am displaying data

`myVisRect: HRectangle`

The rectangle on the display which my currently visible interval covers

Constructors and Destructor:

`PlTlValView(TLEvent* );`

Constructor. Argument is owning TLEvent.

`PlTlValView(const PlTlValView& );`

Copy constructor

`virtual  ~PlTlValView();`

destructor

Operations:

- accDrag

  `virtual unsigned int accDrag();`

  Determine if I can be dragged

- adjPos

  `virtual void adjPos(long , long );`

  Adjust my position

- containedBy

  `virtual unsigned int containedBy(HRectangle* ) const;`

  Determine if I am contained by the given rectangle

- contains

  `virtual unsigned int contains(HRectangle* ) const;`

  Determine if I contain the given rectangle

- DECLARE_LOCAL_CLASS

  `int DECLARE_LOCAL_CLASS(PlTlValView );`

  class PlTlValView (value view)

  This class is the base class for views that display value versus time information graphically in a timeline subregion.

- erase

```
virtual void erase();
```

Erase my whole view

- findEventRegion

```
virtual TLEventSubRegion* findEventRegion();
```

Get the subregion I am on,

- getPos

```
virtual void getPos();
```

Get my position

- intersects

```
virtual unsigned int intersects(const HRectangle* ) const;
```

Determine if I intersect with the given rectangle

- intersects

```
virtual unsigned int intersects(unsigned long x, unsigned long
y) const;
```

Determine if I intersect with the given point

- operator =

```
PlTlValView& operator =(const PlTlValView& );
```

Assignment operator

- ovrThldColor

```
virtual const DRGBColor& ovrThldColor() const;
```

Get my over-threshold color

- ovrThldColor

```
virtual void ovrThldColor(const DRGBColor& col);
```

Set my over-threshold color

- pixel

```
virtual unsigned long pixel(double val);
```

Determine the pixel location corresponding to the input value by doing a linear interpolation based on myVisRect and the max and min values to be displayed as obtained from the associated PlTlValEvent

- recalcVisIntvl

```
virtual void recalcVisIntvl(TLTimeScaleAbs* );
```

Recalculate my visible interval

- recalcVisRect

```
virtual void recalcVisRect(TLTimeScaleAbs* );
```

Recalculate my visible rectangle

- recalc

```
virtual void recalc(TLTimeScaleAbs* );
```

Recalculate my visible interval and my current bounding rectangle on display based on my total interval and the given timescale.

- repaintSpecifics

```
virtual int repaintSpecifics(const HObjCollection& );
```

Repaint what I am displaying using the data in the HObjCollection.

- repaint

```
virtual int repaint();
```

Repaint by getting data over myVisIntvl from my associated PlTlValEvent and then calling repaintSpecifics overrides base class function

- savePos

```
virtual void savePos();
```

Save my position

- selectView

```
virtual void selectView();
```

Select me.

- unselectView

```
virtual void unselectView();
```

Unselect me.

- updateVal

```
virtual int updateVal(const HEpochInterval& intvl);
```

Update value over the given interval.

- visInterval

```
virtual const HEpochInterval& visInterval();
```

Return my visible interval

6.3.13.3.29 PlTlWindow Class

Overview:

class PlTlWindow (-)

PlTlWindow are main windows used by the PlTlAppl class.

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
PlTlWindow();
```

Default constructor

```
virtual  ~PlTlWindow();
```

Destructor

Operations:

- createDisplay

```
virtual int createDisplay();
```

Create the display.

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlTlWindow );
```

class PlTlWindow (-)

PlTlWindow are main windows used by the PlTlAppl class.

- initIcon

```
virtual void initIcon();
```

Create a planning timeline icon. Overrides DWindow::initIcon

6.3.13.3.30 PlTlXtl Class

Overview:

class PlTlXtl (planning timeline)

Instances of PlTlXtl are planning timeline displays. A PlTlXtl is the main application class for the timeline. It is responsible for setting up communication with the name server process to communicate with a system resource model(SRM) which contains a resource pool from which it gets the resource states to display.

Export Control: Public

Inheritance Relationships:

Attributes:

`myBitFile: HString`

My bitmap file that contains path and file name for stipple bitmap

`myDefsConfig: PlTlDefsConfig*`

My configuration file for various default values.

`myDispIntvl: HEpochInterval`

My displayed interval

`myParent: PlTlAppl*`

My parent PlTlAppl

`myPlTlStatusBar: PlTlStatusBar*`

My PlTlXtl Status bar

`myPlTlZmBar: PlTlZmBar*`

My PlTlXtl zoom bar

`myStplConfig: PlTlStplConfig*`

My configuration file for stipple files.

`myTypesConfig: PlTlTypesConfig*`

My configuration file for types.

Constructors and Destructor:

```
PlTlXtl(PlTlAppl* );
```

Constructor. Agument is parent appl.

```
PlTlXtl(const PlTlXtl& orig);
```

Copy constructor

```
virtual  ~PlTlXtl();
```

Destructor

Operations:

- changeViewedRs

```
virtual int changeViewedRs(const HObjCollection& rsToDelete);
```

Change my viewed resources to match new set provided by PlTlAppl. Argument is list of currently-viewed resources that I don't want to see anymore.

- clearInternals

```
virtual void clearInternals();
```

Prepare for a configuration change, ask all entities to clear so we can totally reconfigure

- configBasicDefaults

```
virtual int configBasicDefaults();
```

Set up basic defaults like region colors and time interval

- configStpl

```
virtual int configStpl();
```

Set up possible event types and their associated stipple file

- configSubRegions

```
virtual int configSubRegions();
```

Set up a subregion and a label for each resource that is to be displayed on the timeline

- configTypes

```
virtual int configTypes();
```

Set up possible event types and their associated colors

- createDefsConfig

```
virtual PlTlDefsConfig* createDefsConfig();
```

Create my config file for loading/saving various defaults

- createDisplay

```
virtual int createDisplay();
```

Create the display. Overrides Xtl :: createDisplay to add createStatusBar

- createMainReg

```
virtual int createMainReg();
```

Create main region. Overrides Xtl :: createMainReg to return PlTlMainRegion type instead of TLMainRegion type

- createNewPtlEvents

```
virtual void createNewPtlEvents(RResource* rs, unsigned int key,
const char* nm, const HEpochInterval& spanOfUpdate);
```

Create new PtlEvents

- createNewPtlEvent

```
virtual PlTlSimpleEvent* createNewPtlEvent();
```

Create a new PtlEvent

- createNewStplEvents

```
virtual void createNewStplEvents(RResource* rs, unsigned int
key, const char* nm, const HEpochInterval& spanOfUpdate);
```

Create new stipple events

- createNewStplEvent

```
virtual PlTlStplEvent* createNewStplEvent(RResource* rs);
```

Create a new stipple event

- createPanel

```
virtual int createPanel();
```

Create the PlTlXtl main panel and menus. This adds facilities to permit interactive changes to configuration, colors, and resources.

- createScalableFonts

```
virtual DFontXScalable* createScalableFonts();
```

Create default scalable fonts

- createStatusBar

```
virtual int createStatusBar();
```

Create an PlTlXtl status bar for display messages

- createStplConfig

```
virtual PlTlStplConfig* createStplConfig();
```

Create my config file for loading/saving types and stipple pattern mappings

- createSubRegion

```
virtual int createSubRegion(const HString& resNm, DFontX* font);
```

Create up a subregion and a label for the given name to be displayed on the timeline

- createTypesConfig

```
virtual PlTlTypesConfig* createTypesConfig();
```

Create my config file for loading/saving types and type-to-color mappings

- createZoomBar

```
virtual int createZoomBar();
```

Create an PlTlXtl zoom bar for horizontal scrolling

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlTlXtl );
```

class PlTlXtl (planning timeline)

Instances of PlTlXtl are planning timeline displays. A PlTlXtl is the main application class for the timeline. It is responsible for setting up communication with the name server process to communicate with a system resource model(SRM) which contains a resource pool from which it gets the resource states to display.

- defsConfig

```
virtual PlTlDefsConfig* defsConfig();
```

Get my default configuration information

- determineConfig

```
virtual int determineConfig();
```

Determine initial configuration setup

- dispIntvl

```
virtual void dispIntvl(const HEpochInterval& intvl);
```

Set my display interval

- dispIntvl

```
virtual const HEpochInterval& dispIntvl() const;
```

Get my display interval

- doAttachments

```
virtual int doAttachments();
```

Set up the proper attachments. Overridden to operate on the PlTlXtl zoombar

- doManages

```
virtual int doManages();
```

Manage the timeline area. Overridden to manage the PlTlXtl zoom bar

- getFont

```
virtual DFontX* getFont(const char* family, const char* style,
int size);
```

Obtain a label font of the desired family, style, and size

- getPlTlPanel

```
virtual PlTlPanel* getPlTlPanel();
```

Get the PlTlPanel - I do not pass on ownership

- initialize

```
virtual int initialize();
```

Initialize the timeline display overrides Xtl::initialize

- loadDefaults

```
virtual int loadDefaults();
```

Load my default values by creating the default configuration file and asking it to configure

- operator =

```
PlTlXtl& operator =(const PlTlXtl& orig);
```

Assignment operator

- ptlStatusBar

```
virtual PlTlStatusBar* ptlStatusBar();
```

Get the PlTlXtl zoom bar area

- ptlZmBar

```
virtual PlTlZmBar* ptlZmBar();
```

Get the PlTlXtl zoom bar area

- reconfigure

```
virtual int reconfigure();
```

Reconfigure myself for a new config file

- removeImpactedPtlEvents

  ```
  virtual void removeImpactedPtlEvents(unsigned int key, const
  char* rsrcNm, const HEpochInterval& spanOfUpdate);
  ```

  Remove impacted PtlEvents

- removeImpactedStplEvents

  ```
  virtual void removeImpactedStplEvents(unsigned int key, const
  char* rsrcNm, const HEpochInterval& spanOfUpdate);
  ```

  Remove impacted stipple events

- resize

  ```
  virtual int resize(int newWidth, int newHeight);
  ```

  Override resize to take of PlTlXtl zoom bar

- saveConfig

  ```
  virtual int saveConfig(const char* filenm);
  ```

  Save my configuration to the given file

- setDefBoundsAndSizes

  ```
  virtual int setDefBoundsAndSizes();
  ```

  Set the default width and height of the principal timeline regions

- snapToConfig

  ```
  virtual int snapToConfig();
  ```

  Override snapToConfig to take care of PlTlXtl zoom bar

- snapToInterval

  ```
  virtual void snapToInterval(const HEpochInterval& );
  ```

  Override snapToInterval to take care of PlTlXtl zoom bar

- stateUpdate

  ```
  virtual void stateUpdate(RUpdState* anUpd);
  ```

  state Update is called when my noid gets notification of a state update. I get the list of updates.

- switchTotalIntvl

  ```
  virtual int switchTotalIntvl(const HEpochInterval& intvl);
  ```

  Switch my overall interval to the given interval. I will display the same amount of time I am showing now (if I can), but it will start at the beginning of the new total interval.

- updStates

```
virtual void updStates(RResource* rs, const char* rsrcNm,
unsigned int key, RUpdState* thisUpd);
```

Update a resource that is being displayed as a set of states

6.3.13.3.31 PlTlZmBar Class

Overview:

class PlTlZmBar (-)

Instances of PlTlZmBar are zoom bars which know how to pop up a plan interval window.

Export Control: Public

Inheritance Relationships:

Attributes:

```
myDurDrag: int
```

My drag flag. TRUE during drag, FALSE otherwise

```
myMenu: DOptionMenu*
```

My option menu

```
myOptionBtns: HObjList
```

My list of push buttons on the option menu kept so I can destroy them during reconfiguration

```
myScrollbar: PlTlZmBarCbScrollbar*
```

My scrollbar

```
mySliderPup: PlTlSliderPup*
```

My slider popup

```
myXtl: Xtl*
```

My owning xtl

Constructors and Destructor:

```
PlTlZmBar(Xtl* );
```

Constructor. Argument is my parent Xtl.

```
PlTlZmBar(const PlTlZmBar& orig);
```

Copy constructor

```
virtual  ~PlTlZmBar();
```

Destructor

Operations:

- adjustSlider

  ```
  virtual int adjustSlider(const HEpochInterval& dispIntvl, const HEpochInterval& totIntvl);
  ```

  Adjust slider position and length to match relationship between given displayed interval and given total interval

- adjustToNewIntvl

  ```
  virtual int adjustToNewIntvl(int durSecs);
  ```

  Adjust scrollbar to reflect new interval duration and snap timeline to new displayed interval. I make sure the new interval stays within my total interval.

- clear

  ```
  virtual void clear();
  ```

  clear.

- computeSliderIntvl

  ```
  virtual void computeSliderIntvl(HEpochInterval& intvl);
  ```

  Compute display interval based on current slider position. Return it in intvl

- createOptionMenu

  ```
  virtual int createOptionMenu();
  ```

  Create option menu

- createScrollbar

  ```
  virtual int createScrollbar();
  ```

  Create scrollbar

- createSliderPup

  ```
  virtual int createSliderPup();
  ```

  Create slider popup, and hide it for now

- DECLARE_LOCAL_CLASS

  ```
  int DECLARE_LOCAL_CLASS(PlTlZmBar );
  ```

  class PlTlZmBar (-)

  Instances of PlTlZmBar are zoom bars which know how to pop up a plan interval window.

- fillOptionMenu

  ```
  virtual int fillOptionMenu();
  ```

  Fill the option menu from the defaults file

- getDefsConfig

  ```
  virtual PlTlDefsConfig* getDefsConfig();
  ```

  Return the default configuration class

- init

  ```
  virtual int init();
  ```

  Initialize. Create the form and its child scrollbar and option menu

- operator =

  ```
  PlTlZmBar& operator =(const PlTlZmBar& orig);
  ```

  Assignment operator

- positionWidgets

  ```
  virtual int positionWidgets();
  ```

  Position widgets

- reconfigure

  ```
  virtual int reconfigure();
  ```

  Reconfigure. Reinitialize the option menu buttons

- scrollDragCb

  ```
  virtual void scrollDragCb(PlTlZmBarCbScrollbar* );
  ```

  Callback when scrollbar is dragged

- scrollValChgCb

  ```
  virtual void scrollValChgCb(PlTlZmBarCbScrollbar* );
  ```

  Callback when scrollbar value is changed

- showSelCb

  ```
  virtual void showSelCb(PlTlZmBarCbPushBtn* btn);
  ```

  Callback when new show percentage is selected

6.3.14 Planning_Workbench_Wb Class Category

6.3.14.1 Overview

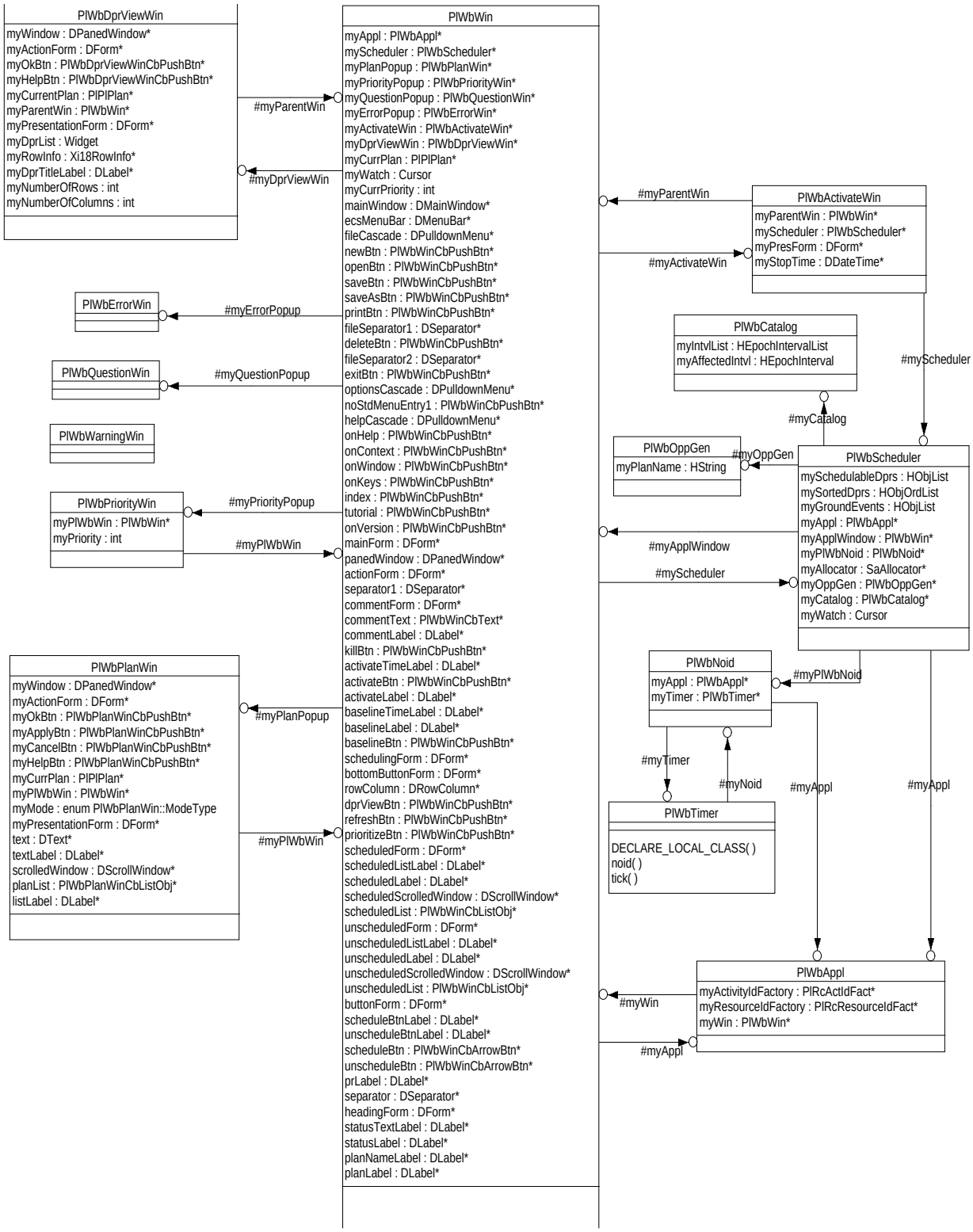


Figure 6.3.14.1-1 Planning_Workbench_Wb

6.3.14.2 Planning_Workbench_Wb Classes

6.3.14.3 PlWbActivateWin Class

Overview:

class PlWbActivateWin (-)

Instances of PlWbActivateWin are dialogs for activating a portion of a plan.

Export Control: Public

Inheritance Relationships:

Attributes:

`myParentWin: PlWbWin*`

My parent window

`myPresForm: DForm*`

My presentation area form

`myScheduler: PlWbScheduler*`

My scheduler

`myStopTime: DDateTime*`

My stop time

Constructors and Destructor:

`PlWbActivateWin(PlWbWin* parentWin, PlWbScheduler* ssiScheduler);`

Constructor.

`PlWbActivateWin(const PlWbActivateWin& orig);`

Copy constructor

`virtual ~PlWbActivateWin();`

Destructor

Operations:

- applyChanges

`virtual void applyChanges();`

Apply changes. tell PlWbScheduler to switch to the new total interval overrides DApplyDialog::applyChanges()

- createActionControls

  ```
  virtual int createActionControls();
  ```

  Modify my action area controls

- createDispObjs

  ```
  virtual int createDispObjs();
  ```

  Create the display objects that make up my presentation area

- createPopup

  ```
  virtual int createPopup(const char* name, DWidget* parent, Arg*
  arglist, Cardinal argcount);
  ```

  Create the DPopup. This overides DPopup::createPopup()

- createPresentationArea

  ```
  virtual int createPresentationArea();
  ```

  Make my presentation area

- createStopTime

  ```
  virtual int createStopTime();
  ```

  Create the stop time item

- DECLARE_LOCAL_CLASS

  ```
  int DECLARE_LOCAL_CLASS(PlWbActivateWin );
  ```

  class PlWbActivateWin (-)

  Instances of PlWbActivateWin are dialogs for activating a portion of a plan.

- init

  ```
  virtual int init();
  ```

  Initialize.

- manageAllChildren

  ```
  virtual void manageAllChildren();
  ```

  Ask all of pieces to manageChild

- operator =

  ```
  PlWbActivateWin& operator =(const PlWbActivateWin& orig);
  ```

Assignment operator

- reconfigure

  `virtual int reconfigure();`

  Reconfigure. Reinitialize the available and viewed lists

- setStopTimePos

  `virtual void setStopTimePos();`

  Set the position of the stop time item

- setStopTime

  `virtual int setStopTime();`

  Initialize the stop time with the current time.

- show

  `virtual void show();`

  Show me on the desktop. Override to set stop time to current total interval.

6.3.14.3.1 PlWbAppl Class

Overview:

class PlWbAppl (-)

Instances of PlWbAppl are derived PlDiAppl's capable of using its noid to request resources, plan names and activities from the resource model.

Export Control: Public

Inheritance Relationships:

Inherits from PlDiAppl

Attributes:

`myActivityIdFactory: PlRcActIdFact*`

My activity id factory

`myResourceIdFactory: PlRcResourceIdFact*`

My resource id factory

`myWin: PlWbWin*`

My window

Constructors and Destructor:

```
PlWbAppl(const PlWbAppl& orig);
```

Copy constructor

```
PlWbAppl();
```

Default constructor

```
virtual  ~PlWbAppl();
```

Destructor

Operations:

- agent

```
virtual PlIpAgent* agent();
```

Get my agent

- allocRemove

```
virtual int allocRemove(const char* planName, const
HEpochInterval& when, int rsId, int actId);
```

Remove an allocation against the specified parameters

- baselinePlan

```
virtual int baselinePlan(const char* name);
```

Baseline the plan in the database

- closeAppl

```
virtual void closeAppl();
```

close application. Overrides PlDiAppl::closeAppl

- connectionMade

```
virtual int connectionMade();
```

Do the rest of the initializing when a connection is made

- copyPlan

```
virtual int copyPlan(const char* srcPlanName, const char*
dstPlanName, HEpochInterval* intvl);
```

Ask the SRM to copy the source plan to the destination plan

- createEnviron

```
virtual int createEnviron();
```

Set up my environment

- createIdFactories

```
virtual int createIdFactories();
```

Create the ID generator factories

- createMshNoid

```
virtual int createMshNoid();
```

Create a connection to the message handler

- createPlan

```
virtual int createPlan(const char* name);
```

Ask the SRM to create a new plan

- createSchedulableResources

```
virtual int createSchedulableResources();
```

Create new schedulable resources (PlRcResource) and add them to the schedulable resource pool.

- create

```
virtual int create(const char* appClass, int& argc, char** argv,
XrmOptionDescList options, Cardinal numOptions);
```

Create me

- DECLARE_CLASS

```
int DECLARE_CLASS(PlWbAppl );
```

class PlWbAppl (-)

Instances of PlWbAppl are derived PlDiAppl's capable of using its noid to request resources, plan names and activities from the resource model.

- deletePlan

```
virtual int deletePlan(const char* name);
```

Ask the SRM to delete a plan

- initCommunications

```
virtual int initCommunications();
```

Initialize communications. Overide PlClAppl behavior to delay creation of SRM noid.

- init

```
virtual int init();
```

Initialize me

- makeWindow

```
virtual DWindow* makeWindow();
```

Make a main window

- newSrmNoid

```
virtual PlClSrmNoid* newSrmNoid();
```

Create the planning workbench SRM noid

- operator =

```
PlWbAppl& operator =(const PlWbAppl& orig);
```

Assignment operator

- savePlan

```
virtual int savePlan(const char* name);
```

Save plan to the database

- shutDown

```
virtual void shutDown();
```

clean up

- updateActivatedDprs

```
virtual int updateActivatedDprs(HObjCollection& dprCollection);
```

Update activated DPRs in the database

6.3.14.3.2 PlWbCatalog Class

Overview:

Class PlWbCatalog (resource model change notifier)

The PlWbCatalog is a local process update catalog for updating the Scheduling System Interface pools.

PlWbCatalog sets up registration of changes of resources, their states, activity changes, and plan changes. It also has protocol for flushing these change notices.

Export Control: Public

Inheritance Relationships:

Attributes:

`myAffectedIntvl: HEpochInterval`

My affected interval

`myIntvlList: HEpochIntervalList`

My interval list

Constructors and Destructor:

`PlWbCatalog(const PlWbCatalog& orig);`

Copy constructor

`PlWbCatalog();`

Default constructor

`virtual  ~PlWbCatalog();`

Destructor

Operations:

- affectedIntvl

  `virtual int affectedIntvl(HEpochInterval& when);`

  Get the affected interval on this PlWbCatalog.

- DECLARE_CLASS

  `int DECLARE_CLASS(PlWbCatalog );`

  Class PlWbCatalog (resource model change notifier)

  The PlWbCatalog is a local process update catalog for updating the Scheduling System Interface pools.

  PlWbCatalog sets up registration of changes of resources, their states, activity changes, and plan changes. It also has protocol for flushing these change notices.

- init

  `virtual int init();`

  Initialize this PlWbCatalog.

- operator =

  `PlWbCatalog& operator =(const PlWbCatalog& orig);`

  Assignment operator

- registerAllocChg

```
virtual int registerAllocChg(int actId, RAll* all, int
cloneFlag);
```

Register change in an allocation state.

- registerAllocNew

```
virtual int registerAllocNew(int actId, RAll* all, int
cloneFlag);
```

Register a new allocation.

- registerAllocRem

```
virtual int registerAllocRem(int actId, RAll* all, int
cloneFlag);
```

Register a removed allocation.

- registerRsStateChg

```
virtual int registerRsStateChg(int rsId, const char* planName,
const HEpochInterval& when);
```

Register change in a resource state.

6.3.14.3.3 PlWbDprViewWin Class

Overview:

class PlWbDprViewWin (-)

Instances of PlWbDprViewWin are dialogs for display a list of DPRs.

Export Control: Public

Inheritance Relationships:

Attributes:

```
myActionForm: DForm*
```

My action region form.

```
myCurrentPlan: PlPlPlan*
```

My current plan

```
myDprList: Widget
```

My DPR list widget

```
myDprTitleLabel: DLabel*
```

My DPR list title label

`myHelpBtn: PlWbDprViewWinCbPushBtn*`

My help button.

`myNumberOfColumns: int`

My number of columns in my list

`myNumberOfRows: int`

My number of rows in my list

`myOkBtn: PlWbDprViewWinCbPushBtn*`

My ok button.

`myParentWin: PlWbWin*`

My parent window.

`myPresentationForm: DForm*`

My presentation form

`myRowInfo: Xi18RowInfo*`

My DPR list info

`myWindow: DPanedWindow*`

My top level paned manager, holds all of my display elements.

Constructors and Destructor:

`PlWbDprViewWin(PlWbWin* parentWin, DXCMapAbs* );`

Constructor.

`PlWbDprViewWin(const PlWbDprViewWin& orig);`

Copy constructor

`virtual  ~PlWbDprViewWin();`

Destructor

Operations:

- createActionArea

  `virtual int createActionArea();`

  Make my action area, that will contain my buttons.

- createActionControls

```
virtual int createActionControls();
```

Make the action controls.

- createDisplay

```
virtual int createDisplay();
```

Make my display.

- createHelp

```
virtual int createHelp();
```

Make my help button.

- createOk

```
virtual int createOk();
```

Make my ok button.

- createPanedWindow

```
virtual int createPanedWindow();
```

Make my enclosing DPanedWindow.

- createPopup

```
virtual int createPopup(const char* name, DWidget* parent, Arg*
arglist, Cardinal argcount);
```

Create the DPopup. This overides DPopup::createPopup()

- createPresentationArea

```
virtual int createPresentationArea();
```

Make the presentation area.

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlWbDprViewWin );
```

class PlWbDprViewWin (-)

Instances of PlWbDprViewWin are dialogs for display a list of DPRs.

- helpCb

```
virtual void helpCb(PlWbDprViewWinCbPushBtn* );
```

Callback when the help button is hit.

- init

```
virtual int init();
```

Initialize.

- installCallbacks

  ```
  virtual void installCallbacks();
  ```

  This member installs the callbacks. It appears that the base class incorrectly sets the no match callback.

- manageAllChildren

  ```
  virtual void manageAllChildren();
  ```

  Ask all of my pieces to manageChild().

- okCb

  ```
  virtual void okCb(PlWbDprViewWinCbPushBtn* );
  ```

  Callback when the ok button is hit.

- operator =

  ```
  PlWbDprViewWin& operator =(const PlWbDprViewWin& orig);
  ```

  Assignment operator

- plan

  ```
  virtual int plan(PlPlPlan* aplan);
  ```

  Set my plan from a given plan object

- plan

  ```
  virtual int plan(const char* planName);
  ```

  Set my plan from a given plan name

- plan

  ```
  virtual PlPlPlan* plan();
  ```

  Get my plan

- populateList

  ```
  virtual int populateList();
  ```

  Populate DPR list.

- positionHelp

  ```
  virtual void positionHelp();
  ```

  Position my help button.

- positionOk

```
virtual void positionOk();
```

Position my ok button.

- refreshList

```
virtual void refreshList();
```

Refresh display for DPR list.

- sizeActionForm

```
virtual int sizeActionForm();
```

After the action area controls are build, set the size of myActionForm. Default behavior is to set myself to the height of my buttons.

- updateList

```
virtual void updateList();
```

Update the contents of the DPR list.

- updateView

```
virtual int updateView();
```

Redraw the popup when any model attributes change.

- update

```
virtual void update();
```

Update all the window contents.

6.3.14.3.4 PlWbErrorWin Class

Overview:

class PlWbErrorWin (Planning Workbench Error Popup)

General purpose window for posting errors.

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
PlWbErrorWin();
```

Constructor.

```
virtual  ~PlWbErrorWin();
```

Default Destructor.

Operations:

- DECLARE_CLASS

```
int DECLARE_CLASS(PlWbErrorWin );
```

class PlWbErrorWin (Planning Workbench Error Popup)

General purpose window for posting errors.

- init

```
virtual int init(DWidget* parent);
```

Initialize message box.

- show

```
virtual int show(const char* msg);
```

Popup message window with given message string.

6.3.14.3.5 PlWbNoid Class

Overview:

class PlWbNoid (derived PlClSrmNoid)

Instances of PlWbNoids handle planning workbench related client noid messages.

Export Control: Public

Inheritance Relationships:

Inherits from PlClSrmNoid

Attributes:

```
myAppl: PlWbAppl*
```

My appl

```
myTimer: PlWbTimer*
```

My timer

Constructors and Destructor:

```
PlWbNoid(PlWbAppl* appl);
```

Constructor with application class

```
PlWbNoid();
```

Default Constructor

```
PlWbNoid(const PlWbNoid& orig);
```

Copy constructor

```
virtual  ~PlWbNoid();
```

Destructor

Operations:

- appl

```
virtual void appl(PlWbAppl* appl);
```

Give me visibility to my application.

- connEstablished

```
virtual int connEstablished();
```

When connection is established, initialize.

- DECLARE_CLASS

```
int DECLARE_CLASS(PlWbNoid );
```

class PlWbNoid (derived PlClSrmNoid)

Instances of PlWbNoids handle planning workbench related client noid messages.

- handleAllocRem

```
virtual int handleAllocRem(HMsgConn* , HAddress* , SMsgAllocRem*
);
```

Method to handle a message indicating whether an allocation remove that I requested was successful.

- handlePlanNames

```
virtual int handlePlanNames(HMsgConn* , HAddress* ,
SMsgPlanNmsRsp* msg);
```

Handle plan names by giving then to the parent

- handlePlanNew

```
virtual int handlePlanNew(HMsgConn* , HAddress* , SMsgPlanNew* );
```

Method to handle new plan name

- handlePlanSnap

```
virtual int handlePlanSnap(HMsgConn* , HAddress* , SMsgPlanSnap*
msg);
```

Handle plan snap message

- handlePlClMsgPlanNew

```
virtual int handlePlClMsgPlanNew(HMsgConn* con, HAddress* adr,
PlClMsgPlanNew* msg);
```

Method to handle derived new plan message.

- init

```
virtual int init();
```

Initialize the application.

- installActChg

```
virtual int installActChg(RActivity* act);
```

Install a changed activity

- installActDel

```
virtual int installActDel(int actId);
```

Delete an activity

- installActNew

```
virtual int installActNew(RActivity* act);
```

Install a new activity

- installAllocUpd

```
virtual int installAllocUpd(RUpdAllocAbs* anUpdate);
```

Install an updated allocation.

- installPlanClear

```
virtual int installPlanClear(const char* planName,
HEpochInterval* intvl, HVList* rsIds);
```

Clear a plan

- installPlanDel

```
virtual int installPlanDel(const char* planName);
```

Delete a plan

- installPlanNew

```
virtual int installPlanNew(const char* planName, HVList* rsIds);
```

Install a new plan

- operator =

```
PlWbNoid& operator =(const PlWbNoid& orig);
```

Assignment operator

- requestPlanNew

```
virtual int requestPlanNew(const char* planName, HVList* rsIds);
```

Override the base class. We ignore the rsId list.

- tick

```
virtual void tick();
```

Fire my timer.

6.3.14.3.6 PlWbOppGen Class

Overview:

class PlWbOppGen

Instances of PlWbOppGen are opportunity generators for the production planning workbench.

Export Control: Public

Inheritance Relationships:

Attributes:

```
myPlanName: HString
```

The plan for which I am generating opportunities.

Constructors and Destructor:

```
PlWbOppGen();
```

Default constructor

```
virtual  ~PlWbOppGen();
```

Default destructor

Operations:

- createDprOpportunities

  ```
  virtual int createDprOpportunities(RActivity* act, RSchRqst*
  req);
  ```

  Attempt to create a scheduling opportunity for this request from the DPR activity information

- createGroundEventOpportunities

  ```
  virtual int createGroundEventOpportunities(RActivity* act,
  RSchRqst* req);
  ```

  Attempt to create a scheduling opportunity for this request from the DPR activity information

- DECLARE_LOCAL_CLASS

  ```
  int DECLARE_LOCAL_CLASS(PlWbOppGen );
  ```

  class PlWbOppGen

  Instances of PlWbOppGen are opportunity generators for the production planning workbench.

- earliestComputerSchOpp

  ```
  const RSchOpp earliestComputerSchOpp(PlRcComputer* computer,
  const HEpochTime& earliestStartTime, double dprDuration);
  ```

  Find a scheduling opportunity for the supplied computer starting later than the given start time for the given duration.

- earliestStringSchOpp

  ```
  const RSchOpp earliestStringSchOpp(PlRcString* string, const
  HEpochTime& earliestStartTime, double dprDuration);
  ```

  Find a scheduling opportunity for one of the supplied computers starting later than the given start time for the given duration.

- init

  ```
  virtual int init();
  ```

  Initialization

- opportunities

  ```
  virtual int opportunities(RActivity* act, RSchRqst* req);
  ```

  Attempt to create a scheduling opportunity for this request from the activity information

- pastOpportunities

  ```
  virtual int pastOpportunities(RActivity* act, RSchRqst* req);
  ```

Attempt to create a scheduling opportunity for this request from the activity's historical information

- planName

  `virtual HString& planName();`

  Get the plan name

- planName

  `virtual void planName(const char* plan);`

  Set the plan name

6.3.14.3.7 PlWbPlanWin Class

Overview:

class PlWbPlanWin (Plan keeper and popup)

A PlWbPlanWin is a dialog which allows selection of a plan.

Export Control: Public

Inheritance Relationships:

Attributes:

`listLabel: DLabel*`

My list label

`myActionForm: DForm*`

My action region form.

`myApplyBtn: PlWbPlanWinCbPushBtn*`

My apply button.

`myCancelBtn: PlWbPlanWinCbPushBtn*`

My cancel button.

`myCurrPlan: PlPlPlan*`

My currently selected plan

`myHelpBtn: PlWbPlanWinCbPushBtn*`

My help button.

`myMode: enum PlWbPlanWin::ModeType`

My current mode

`myOkBtn: PlWbPlanWinCbPushBtn*`

My ok button.

`myPlWbWin: PlWbWin*`

My parent window.

`myPresentationForm: DForm*`

My presentation form

`myWindow: DPanedWindow*`

My top level paned manager, holds all of my display elements.

`planList: PlWbPlanWinCbListObj*`

My plan list

`scrolledWindow: DScrollWindow*`

My scrolled window

`textLabel: DLabel*`

My text label

`text: DText*`

My text field

Constructors and Destructor:

`PlWbPlanWin(DXCMapAbs* );`

Default constructor

`PlWbPlanWin(const PlWbPlanWin& orig);`

Copy constructor

`virtual  ~PlWbPlanWin();`

Destructor

Operations:

- applyCb

  `virtual void applyCb(PlWbPlanWinCbPushBtn* , XtPointer );`

  Callback when the apply button is hit.

- applyChanges

```
virtual void applyChanges();
```

Apply changes

- apply

```
virtual void apply(XtPointer callData);
```

Callback for the apply button

- cancelCb

```
virtual void cancelCb(PlWbPlanWinCbPushBtn* , XtPointer );
```

Callback when the cancel button is hit.

- cancel

```
virtual void cancel(XtPointer callData);
```

Callback for the cancel button

- createActionArea

```
virtual int createActionArea();
```

Make my action area, that will contain my buttons.

- createActionControls

```
virtual int createActionControls();
```

Make the action controls.

- createApply

```
virtual int createApply();
```

Make my apply button.

- createCancel

```
virtual int createCancel();
```

Make my cancel button.

- createDisplay

```
virtual int createDisplay();
```

Make my display.

- createHelp

```
virtual int createHelp();
```

Make my help button.

- createOk

```
virtual int createOk();
```

Make my ok button.

- createPanedWindow

```
virtual int createPanedWindow();
```

Make my enclosing DPanedWindow.

- createPopup

```
virtual int createPopup(const char* name, DWidget* parent, Arg*
arglist, Cardinal argcount);
```

Create the DPopup. This overides DPopup::createPopup()

- createPresentationArea

```
virtual int createPresentationArea();
```

Make the presentation area.

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlWbPlanWin );
```

class PlWbPlanWin (Plan keeper and popup)

A PlWbPlanWin is a dialog which allows selection of a plan.

- defaultActionCallback

```
static void defaultActionCallback(Widget parent, XtPointer
clientData, XtPointer callData);
```

Callback for the list default action.

- defaultAction

```
virtual void defaultAction(XtPointer callData);
```

Routine called when the for the list default action (double click, etc.)

- deselectAllItems

```
virtual int deselectAllItems();
```

Deselect all items in the list.

- getSelectedPlan

```
virtual PlPlPlan* getSelectedPlan();
```

Get the selected plan from the list.

- getTextString

```
int getTextString(HString& textString) const;
```

Get the text from the text field.

- helpCb

```
virtual void helpCb(PlWbPlanWinCbPushBtn* , XtPointer );
```

Callback when the help button is hit.

- init

```
virtual int init(PlWbWin* parent);
```

Initialize.

- installCallbacks

```
virtual void installCallbacks();
```

This member installs the callbacks. It appears that the base class incorrectly sets the no match callback.

- manageAllChildren

```
virtual void manageAllChildren();
```

Ask all of my pieces to manageChild().

- noMatch

```
virtual void noMatch(XtPointer callData);
```

Callback if the text does not match a list element

- okCb

```
virtual void okCb(PlWbPlanWinCbPushBtn* , XtPointer );
```

Callback when the ok button is hit.

- operator =

```
PlWbPlanWin& operator =(const PlWbPlanWin& orig);
```

Assignment operator

- plan

```
virtual int plan(PlPlPlan* aplan);
```

Set my plan from a given plan object

- plan

```
virtual int plan(const char* planName);
```

Set my plan from a given plan name

- plan

  ```
  virtual PlPlPlan* plan();
  ```

  Get my plan

- positionApply

  ```
  virtual void positionApply();
  ```

  Position my apply button.

- positionCancel

  ```
  virtual void positionCancel();
  ```

  Position my cancel button.

- positionHelp

  ```
  virtual void positionHelp();
  ```

  Position my help button.

- positionOk

  ```
  virtual void positionOk();
  ```

  Position my ok button.

- selectItem

  ```
  virtual int selectItem(const char* item);
  ```

  Select an item from the list.

- setTextString

  ```
  virtual int setTextString(const char* textString);
  ```

  Sets the string displayed in the DText displayable, but also invokes the displayable's modifyVerifyCallback, valueChangedCallback, sets the insert cursor position to the beginning of the string, and lastly, invokes the displayable's motionVerifyCallback routine.

- showMode

  ```
  virtual int showMode(enum PlWbPlanWin::ModeType );
  ```

  Manage my dialog according to the mode.

- singleSelectionCallback

  ```
  static void singleSelectionCallback(Widget parent, XtPointer
  clientData, XtPointer callData);
  ```

Static member that is called by Xt when an item in the list is selected. This converts from the procedural to the oo domain. The routine translates from a static C routine (required by Motif) into a virtual C++ routine. Callback for the list single selection.

- singleSelection

  ```
  virtual void singleSelection(XtPointer callData);
  ```

  Routine called when the list detects a single selection

- sizeActionForm

  ```
  virtual int sizeActionForm();
  ```

  After the action area controls are build, set the size of myActionForm. Default behavior is to set myself to the height of my buttons.

- updateView

  ```
  virtual int updateView();
  ```

  Redraw the popup when any model attributes change.

- update

  ```
  virtual void update();
  ```

  Update all the dialog contents.

6.3.14.3.8 PlWbPriorityWin Class

Overview:

class PlWbPriorityWin (Priority selection popup)

A PlWbPriorityWin is a window which allows selection of a priority for producion requests.

Export Control: Public

Inheritance Relationships:

Attributes:

```
myPlWbWin: PlWbWin*
```

My parent window.

```
myPriority: int
```

My priority

Constructors and Destructor:

```
PlWbPriorityWin(const PlWbPriorityWin& orig);
```

Copy constructor

```
PlWbPriorityWin();
```

Constructor

```
virtual  ~PlWbPriorityWin();
```

Destructor

Operations:

- apply

```
virtual void apply(XtPointer callData);
```

Callback for apply button

- cancel

```
virtual void cancel(XtPointer callData);
```

Callback for cancel button

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlWbPriorityWin );
```

class PlWbPriorityWin (Priority selection popup)

A PlWbPriorityWin is a window which allows selection of a priority for producion requests.

- init

```
virtual int init(PlWbWin* parent);
```

Initialize

- installCallbacks

```
virtual void installCallbacks();
```

This member installs the callbacks.

- ok

```
virtual void ok(XtPointer callData);
```

Callback for ok button

- operator =

```
PlWbPriorityWin& operator =(const PlWbPriorityWin& orig);
```

Assignment operator

- priority

```
virtual int priority(int value);
```

Set my priority

- priority

```
virtual int priority();
```

Get my priority

- show

```
virtual int show();
```

Manage my dialog according to the mode.

6.3.14.3.9 PlWbQuestionWin Class

Overview:

class PlWbQuestionWin (Planning Workbench Question Popup)

General purpose window for posing questions

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
PlWbQuestionWin();
```

Constructor.

```
virtual  ~PlWbQuestionWin();
```

Default Destructor.

Operations:

- DECLARE_CLASS

```
int DECLARE_CLASS(PlWbQuestionWin );
```

class PlWbQuestionWin (Planning Workbench Question Popup)
General purpose window for posing questions

- init

  `virtual int init(DWidget* parent);`
  Initialize message box.

- show

  `virtual int show(const char* msg);`
  Popup message window with given message string.

6.3.14.3.10 PlWbScheduler Class

Overview:
class PlWbScheduler
The PlWbScheduler is the scheduler for the production planning workbench.

Export Control: Public

Inheritance Relationships:

Attributes:

`myAllocator: SaAllocator*`
My allocator

`myApplWindow: PlWbWin*`
My appl window

`myAppl: PlWbAppl*`
My appl

`myCatalog: PlWbCatalog*`
My update catalog

`myGroundEvents: HObjList`
List of Ground Events created from the selected activities.

`myOppGen: PlWbOppGen*`
My opportunity generator

myPlWbNoid: PlWbNoid*

My resource model client noid

mySchedulableDprs: HObjList

List of valid, schedulable DPR activities from the selected activities.

mySortedDprs: HObjOrdList

Sorted list of DPRs. This relies on the DPR compare operation.

myWatch: Cursor

My cursor

Constructors and Destructor:

PlWbScheduler(const PlWbScheduler&);

Copy constructor (Disallows copying)

PlWbScheduler(PlWbAppl* appl);

Default constructor

virtual ~PlWbScheduler();

Default destructor

Operations:

- activate

  virtual int activate(const RPlan* plan, const HEpochInterval& intvl);

  Submit jobs to processing for the given interval of the currently selected plan.

- allocator

  virtual SaAllocator* allocator();

  Get my allocator

- allocator

  virtual int allocator(SaAllocator* allocator);

  Set my allocator

- calculateDprSuccessors

  virtual void calculateDprSuccessors();

  Calculate the number of DPR successors for each of the DPRs

- DECLARE_LOCAL_CLASS

  ```
  int DECLARE_LOCAL_CLASS(PlWbScheduler );
  ```

  class PlWbScheduler

  The PlWbScheduler is the scheduler for the production planning workbench.

- init

  ```
  virtual int init();
  ```

  Initialize

- killJobs

  ```
  virtual int killJobs();
  ```

  Kill AUTOSYS jobs.

- operator =

  ```
  PlWbScheduler& operator =(const PlWbScheduler& );
  ```

  Assignment operator (Disallows assignments)

- reSchedule

  ```
  virtual int reSchedule(const char* planName, HObjCollection&
  col);
  ```

  Reschedule activities

- schedulePastJobs

  ```
  virtual int schedulePastJobs(const char* planName);
  ```

  Schedule jobs which have already executed.

- schedule

  ```
  virtual int schedule(const char* planName, HObjCollection& col);
  ```

  Schedule activities

- setDprPriorities

  ```
  virtual void setDprPriorities(RPlan* currentPlan);
  ```

  Iterate through the schedulable DPRS, setting the priorities to the parent production request priority for this plan.

- setDprTaskLevels

  ```
  virtual void setDprTaskLevels();
  ```

  Calculate the "task level" and the earliest scheduled start time for each of the schedulable DPRs.

- setWatch

  ```
  virtual void setWatch(int flag);
  ```

  Set my cursor to watch (if TRUE) or back to normal (if FALSE). Overridden from base class to avoid core dump when there is no window.

- sortDprs

  ```
  virtual void sortDprs();
  ```

  Sort the DPRs and insert them into the sorted DPR list.

- unSchedule

  ```
  virtual int unSchedule(const char* planName, HObjCollection& col);
  ```

  Unschedule activities

- updatePlan

  ```
  virtual void updatePlan(RPlan* thePlan);
  ```

  Update the list of production requests for this plan.

- validate

  ```
  virtual void validate(HObjCollection& selectedActs);
  ```

  Check selected activites to make sure their dependencies can be met and store schedulable activities in the DPR and Ground Event lists

6.3.14.3.11 PlWbTimer Class

Overview:

class PlWbTimer (-)

Instances of PlWbTimer are timers used for scheduling timeouts and control.

Export Control: Public

Inheritance Relationships:

Attributes:

```
myNoid: PlWbNoid*
```

My resource model client noid

Constructors and Destructor:

```
PlWbTimer(const PlWbTimer& orig);
```

Copy constructor

```
PlWbTimer(long microSeconds);
```

Default constructor

```
virtual  ~PlWbTimer();
```

Destructor

Operations:

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlWbTimer );
```

class PlWbTimer (-)

Instances of PlWbTimer are timers used for scheduling timeouts and control.

- noid

```
virtual void noid(PlWbNoid* n);
```

Set my noid.

- operator =

```
PlWbTimer& operator =(const PlWbTimer& orig);
```

Assignment operator

- tick

```
virtual void tick();
```

Override base tick() behavior.

6.3.14.3.12 PlWbWarningWin Class

Overview:

class PlWbWarningWin (Planning Workbench Error Popup)

General purpose window for posting errors.

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
PlWbWarningWin();
```

Constructor.

```
virtual  ~PlWbWarningWin();
```

Default Destructor.

Operations:

- DECLARE_CLASS

```
int DECLARE_CLASS(PlWbWarningWin );
```

class PlWbWarningWin (Planning Workbench Error Popup)
General purpose window for posting errors.

- init

```
virtual int init(DWidget* parent);
```

Initialize message box.

- show

```
virtual int show(const char* msg);
```

Popup message window with given message string.

6.3.14.3.13 PlWbWin Class

Overview:
class PlWbWin (A derived DWindow)
This class represents the main Motif Window for the production planning workbench.

Export Control: Public

Inheritance Relationships:

Attributes:

```
actionForm: DForm*
```

My action form

activateBtn: PlWbWinCbPushBtn*

My activate button

activateLabel: DLabel*

My activate label

activateTimeLabel: DLabel*

My activation time label

baselineBtn: PlWbWinCbPushBtn*

My baseline button

baselineLabel: DLabel*

My baseline label

baselineTimeLabel: DLabel*

My baseline time label

bottomButtonForm: DForm*

My bottom button form

buttonForm: DForm*

My button form

commentForm: DForm*

My comment form

commentLabel: DLabel*

My comment label

commentText: PlWbWinCbText*

My comment text field

deleteBtn: PlWbWinCbPushBtn*

My delete button

dprViewBtn: PlWbWinCbPushBtn*

My view DPR button

ecsMenuBar: DMenuBar*

My menu bar

exitBtn: PlWbWinCbPushBtn*

My exit button

`fileCascade: DPulldownMenu*`

My file cascade button

`fileSeparator1: DSeparator*`

My first file menu separator

`fileSeparator2: DSeparator*`

My first file menu separator

`headingForm: DForm*`

My heading form

`helpCascade: DPulldownMenu*`

My help cascade button

`index: PlWbWinCbPushBtn*`

My index button

`killBtn: PlWbWinCbPushBtn*`

My kill button

`mainForm: DForm*`

My main form

`mainWindow: DMainWindow*`

My main window

`myActivateWin: PlWbActivateWin*`

My activation window

`myAppl: PlWbAppl*`

My appl

`myCurrPlan: PlPlPlan*`

My current plan

`myCurrPriority: int`

My current default priority for production requests

`myDprViewWin: PlWbDprViewWin*`

My activation window

`myErrorPopup: PlWbErrorWin*`

My error window

myPlanPopup: PlWbPlanWin*

My plan window

myPriorityPopup: PlWbPriorityWin*

My priority window

myQuestionPopup: PlWbQuestionWin*

My question window

myScheduler: PlWbScheduler*

My scheduler

myWatch: Cursor

My display cursor

newBtn: PlWbWinCbPushBtn*

My new button

noStdMenuEntry1: PlWbWinCbPushBtn*

My options submenu button

onContext: PlWbWinCbPushBtn*

My on context help button

onHelp: PlWbWinCbPushBtn*

My on help button

onKeys: PlWbWinCbPushBtn*

My on keys help button

onVersion: PlWbWinCbPushBtn*

My on version button

onWindow: PlWbWinCbPushBtn*

My on window help button

openBtn: PlWbWinCbPushBtn*

My open button

optionsCascade: DPulldownMenu*

My options cascade button

panedWindow: DPanedWindow*

My paned window

planLabel: DLabel*

My plan label

planNameLabel: DLabel*

My plan name label

printBtn: PlWbWinCbPushBtn*

My print button

prioritizeBtn: PlWbWinCbPushBtn*

My prioritize button

prLabel: DLabel*

My production request label

refreshBtn: PlWbWinCbPushBtn*

My refresh button

rowColumn: DRowColumn*

My button rowColumn

saveAsBtn: PlWbWinCbPushBtn*

My save as button

saveBtn: PlWbWinCbPushBtn*

My save button

scheduleBtnLabel: DLabel*

My schedule button label

scheduleBtn: PlWbWinCbArrowBtn*

My schedule button

scheduledForm: DForm*

My scheduled form

scheduledLabel: DLabel*

My scheduled label

scheduledListLabel: DLabel*

My scheduled list

scheduledList: PlWbWinCbListObj*

My scheduled list

scheduledScrolledWindow: DScrollWindow*

My scheduled scrolled window

schedulingForm: DForm*

My scheduling form

separator1: DSeparator*

My first separator

separator: DSeparator*

My other separator

statusLabel: DLabel*

My status label

statusTextLabel: DLabel*

My status text label

tutorial: PlWbWinCbPushBtn*

My tutorial button

unscheduleBtnLabel: DLabel*

My unschedule button label

unscheduleBtn: PlWbWinCbArrowBtn*

My unschedule button

unscheduledForm: DForm*

My unscheduled form

unscheduledLabel: DLabel*

My unscheduled label

unscheduledListLabel: DLabel*

My unscheduled list label

unscheduledList: PlWbWinCbListObj*

My unscheduled list

unscheduledScrolledWindow: DScrollWindow*

My unscheduled scrolled window

Constructors and Destructor:

PlWbWin(const PlWbWin& orig);

Copy constructor

```
PlWbWin(PlWbAppl* appl);
```

Default Constructor

```
virtual  ~PlWbWin();
```

Default Destructor

Operations:

- activateBtnCb

```
virtual void activateBtnCb(PlWbWinCbPushBtn* , XtPointer cbs);
```

Activate button callback

- baselineBtnCb

```
virtual void baselineBtnCb(PlWbWinCbPushBtn* , XtPointer cbs);
```

Baseline button callback

- commentChangedCb

```
virtual void commentChangedCb(PlWbWinCbText* , XtPointer );
```

Comment callback

- copyPlan

```
virtual int copyPlan(const char* srcName, const char* dstName);
```

Copy the current plan.

- createActivateWin

```
virtual PlWbActivateWin* createActivateWin();
```

Instatiate a window for activating a portion of the plan.

- createDisplay

```
virtual int createDisplay();
```

Overriden from base class to provide window display creation.

- createDprViewWin

```
virtual int createDprViewWin();
```

Instantiate a DPR view window.

- createErrorWin

```
virtual int createErrorWin();
```

Instantiate a popup error window.

- createPlanWin

```
virtual int createPlanWin();
```

Instantiate a plan selection window.

- createPlan

```
virtual int createPlan(const char* name);
```

Create a new plan.

- createPriorityWin

```
virtual int createPriorityWin();
```

Instantiate a priority selection window.

- createQuestionWin

```
virtual int createQuestionWin();
```

Instantiate a popup question window.

- DECLARE_LOCAL_CLASS

```
int DECLARE_LOCAL_CLASS(PlWbWin );
```

class PlWbWin (A derived DWindow)

This class represents the main Motif Window for the production planning workbench.

- deleteBtnCb

```
virtual void deleteBtnCb(PlWbWinCbPushBtn* , XtPointer cbs);
```

Delete button callback

- deletePlan

```
virtual int deletePlan(const char* name);
```

Delete a plan.

- dprViewBtnCb

```
virtual void dprViewBtnCb(PlWbWinCbPushBtn* , XtPointer cbs);
```

DPR view button callback

- exitBtnCb

```
virtual void exitBtnCb(PlWbWinCbPushBtn* , XtPointer cbs);
```

Exit button callback

- formatScheduledListString

```
int formatScheduledListString(char* string, PlAcPr*
productionRequest);
```

Format a string for the unscheduled list.

- formatUnscheduledListString

```
int formatUnscheduledListString(char* string, PlAcPr*
productionRequest);
```

Format a string for the unscheduled list.

- getActList

```
virtual int getActList(PlWbWinCbListObj* list, HObjCollection&
objCol);
```

Get the list of selected activities.

- getAllActs

```
virtual int getAllActs(PlWbWinCbListObj* list, HObjCollection&
objCol);
```

Get the list of all activities.

- initIcon

```
virtual void initIcon();
```

Overriden from base class to set my personal icon.

- init

```
virtual int init();
```

Initialization

- killBtnCb

```
virtual void killBtnCb(PlWbWinCbPushBtn* , XtPointer cbs);
```

Kill button callback

- newBtnCb

```
virtual void newBtnCb(PlWbWinCbPushBtn* , XtPointer cbs);
```

New button callback

- openBtnCb

```
virtual void openBtnCb(PlWbWinCbPushBtn* , XtPointer cbs);
```

Open button callback

- openPlan

```
virtual int openPlan();
```

Open the active plan if it is available. Else, open a candidate or new plan.

- operator =

```
PlWbWin& operator =(const PlWbWin& orig);
```

Assignment operator

- parseCommandLine

```
virtual int parseCommandLine();
```

Parse the command line.

- plan

```
virtual PlPlPlan* plan();
```

Get the current plan

- plan

```
virtual int plan(const char* planName);
```

Set the current plan.

- postError

```
virtual int postError(const char* messageString);
```

Popup the error window with a given message.

- postQuestion

```
virtual int postQuestion(const char* messageString);
```

Popup the question window with a given question.

- prioritizeBtnCb

```
virtual void prioritizeBtnCb(PlWbWinCbPushBtn* , XtPointer cbs);
```

Priority button callback

- priority

```
virtual int priority(int value);
```

Set the current priority for production requests.

- priority

```
virtual int priority();
```

Get the current priority for production requests

- refreshBtnCb

```
virtual void refreshBtnCb(PlWbWinCbPushBtn* , XtPointer cbs);
```

Refresh button callback

- removeSelectedItems

```
virtual int removeSelectedItems(PlWbWinCbListObj* list);
```

Remove selected items from the list.

- rescheduleCurrentPlan

```
virtual int rescheduleCurrentPlan();
```

Reschedule current plan.

- saveAsBtnCb

```
virtual void saveAsBtnCb(PlWbWinCbPushBtn* , XtPointer cbs);
```

Save as button callback

- saveBtnCb

```
virtual void saveBtnCb(PlWbWinCbPushBtn* , XtPointer cbs);
```

Save button callback

- savePlan

```
virtual int savePlan(const char* name);
```

Save a plan to the database.

- scheduleBtnCb

```
virtual void scheduleBtnCb(PlWbWinCbArrowBtn* , XtPointer cbs);
```

Schedule button callback

- scheduleCurrentPlan

```
virtual int scheduleCurrentPlan();
```

Schedule a plan from the list of production requests for the current plan. This provides a way of initially scheduling a plan and it is assumed that no planning has taken place for the current plan for the current session. Contrast this with scheduledBtnCb below.

- schedulePastJobs

```
virtual int schedulePastJobs();
```

Schedule on the current plan jobs which have already executed.

- setup

```
virtual int setup();
```

Setup display.

- setWatch

```
virtual void setWatch(int flag);
```

Set my cursor to watch (if TRUE) or back to normal (if FALSE). Overridden from base class to avoid core dump when there is no window.

- shutDown

```
virtual void shutDown();
```

Shutdown the window. Make sure open plans have been saved.

- unscheduleBtnCb

```
virtual void unscheduleBtnCb(PlWbWinCbArrowBtn* , XtPointer
cbs);
```

Unschedule button callback

- updateActivationStatus

```
int updateActivationStatus();
```

Update activation related widgets

- updateForm

```
virtual int updateForm();
```

Update the contents of the widgets including lists and textfields.

- updatePriorities

```
virtual int updatePriorities();
```

Update the priorities of the selected production request(s) in the scheduled list.

- update

```
virtual int update();
```

Update the main windows and dialogs