TYX CORPORATION

Productivity Enhancement Systems

TPS Server

1 Scope

This document describes the TPS Server component. The TPS Server provides support for RTS monitoring and control as well as support for bi-directional data transfer with the TPS and synchronous operation. The document describes the available services and interfaces for PAWS Studio.

2 Overview

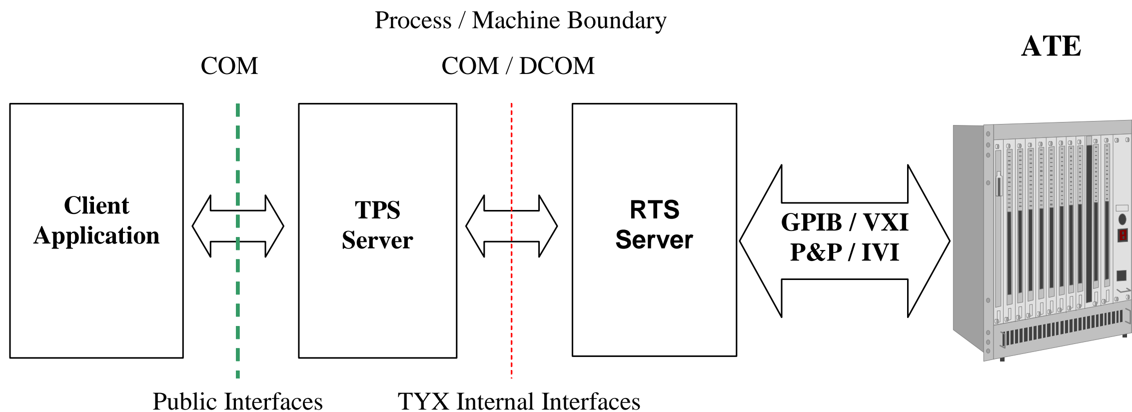
The TPS Server is a component that provides:

- • Access to the RTS Server as an open server component capable of runtime collaboration with other control and monitoring components
- • A stable interface to access the RTS executive functionality insulating the client applications from internal RTS server interface or semantic changes
- • Dispatch interfaces compatible with Visual Basic, C++, C#, Java Script, VB Script bindings
- • A variety of COM events during the operation of the RTS Executive
- • Full configuration support for the RTS Server
- • Support for I/O interception
- • Support for RTS distribution
- • Support for bi-directional data transfer with the ATLAS program
- • Support for both synchronous and asynchronous operation
- • Support for custom I/O resources
- • Access to ATLAS global variables
- • Support to Run an ATLAS block of code

The TPS Server interfaces organize and expose methods for other component objects or clients to interact with the operation of the RTS Executive.

A TPS Server is a client of the RTS Server and a server for its clients. It provides simplified access to RTS Server functionality for clients executing a TPS.

The RTS Server interfaces are subject to change and are designed and maintained for TYX internal usage only. TYX does not recommend the usage of RTS Server interfaces in any client applications.



The TPS Server is designed and maintained by TYX to provide a public and stable interface to the RTS Server. Client applications can use the TPS Server component and interfaces; TYX will maintain and ensure compatibility between the TPS Server and the RTS Server.

3 TPS Server Component

The TPS Server component is essentially a RTS COM adapter. It is externally creatable using the COM API.

The TPS Server component and its interfaces are described in the type library packaged as a resource with **"RtsAx.dll"**. The **"RtsAx.dll"** file is located in "<usr>\tyx\com directory", where <usr> is the directory selected at installation – the default is "C:\usr".

In addition to the functionality provided by the *IRtsControl* interface, the TPS Server provides:

- Support for bi-directional data transfer with the ATLAS program
- Support for both synchronous and asynchronous operation
- Support for custom I/O resources
- Access to ATLAS global variables
- Support to Run an ATLAS block of code

3.1 ITpsServer Interface

The *ITpsServer* interface derives from *IRtsControl* interface [Refer to RTS COM Adapters.doc for an explanation of this interface]. In addition to the functionality exposed by the *IRtsControl* interface it provides properties to control the synchronous operation and the transfer of parameters and results between the client and the TPS.

3.1.1 IDL Description

```
[
	object,
	uuid(3F6B2941-F0DA-11D2-BBB0-00C0268914D3),
	dual,
	helpstring("ITpsServer Interface"),
	pointer_default(unique)
]
interface ITpsServer : IRtsControl
{
	[propget, id(1), helpstring("property Parameters")]
	HRESULT Parameters([out, retval] IDispatch** pVal);
	[propget, id(2), helpstring("property Results")]
	HRESULT Results([out, retval] IDispatch** pVal);
	[propget, id(3), helpstring("property Synchronous")]
	HRESULT Synchronous([out, retval] VARIANT_BOOL* pVal);
	[propput, id(3), helpstring("property Synchronous")]
	HRESULT Synchronous([in] VARIANT_BOOL newVal);
};
```

3.1.2 Parameters Property

Type	Access	Description
IDispatch*	Read-Only	Interface pointer to the PARAMETERS *DataBagResource* object

The *Parameters* property provides support for passing parameters from the client application to the TPS. The property is an interface pointer to the PARAMETERS *DataBagResource* object. The *DataBagResource* object implements a number of additional interfaces: *IIOResource*, *ITextResource*, *IDataBag* and *IDataCollection.* The data access collections can be used to populate the data container with parameters. Please see the documentation the *DataContainer* and the *DataBagResource* components for details. The property is read-only; the access to the *DataBagResource* object is read-write.

The parameters are passed as a collection of named values. On the TPS side the parameters can be retrieved by defining and reading from a text file resource with the name "PARAMETERS". The parameters are exposed in name / value pairs.

3.1.3 Results Property

Type	Access	Description
IDispatch*	Read-Only	Interface pointer to the RESULTS *DataBagResource* object

The *Results* property provides support for returning results from the TPS to the client application. The property is an interface pointer to the RESULTS *DataBagResource* object. The *DataBagResource* object implements a number of additional interfaces: *IIOResource*, *ITextResource*, *IDataBag* and *IDataCollection.* The data access collections can be used to query the data container for results. Please see the documentation the *DataContainer* and the *DataBagResource* components for details. The property is read-only, the access to the *DataBagResource* object is read-write.

The results are passed as a collection of named values. On the TPS side the results can be returned by defining and writing to a text file resource with the name "RESULTS". The results are expected in named values pairs.

3.1.4 Synchronous Property

Type	Access	Description
VARIANT_BOOL	Read-Write	Synchronous / Asynchronous operation mode

The *Synchronous* property controls the operation mode. The default property value is VARIANT_TRUE, i.e. the object operates by default in synchronous mode. In synchronous mode the TPS Server waits before returning from the call until the load, run or unload sequences are complete.

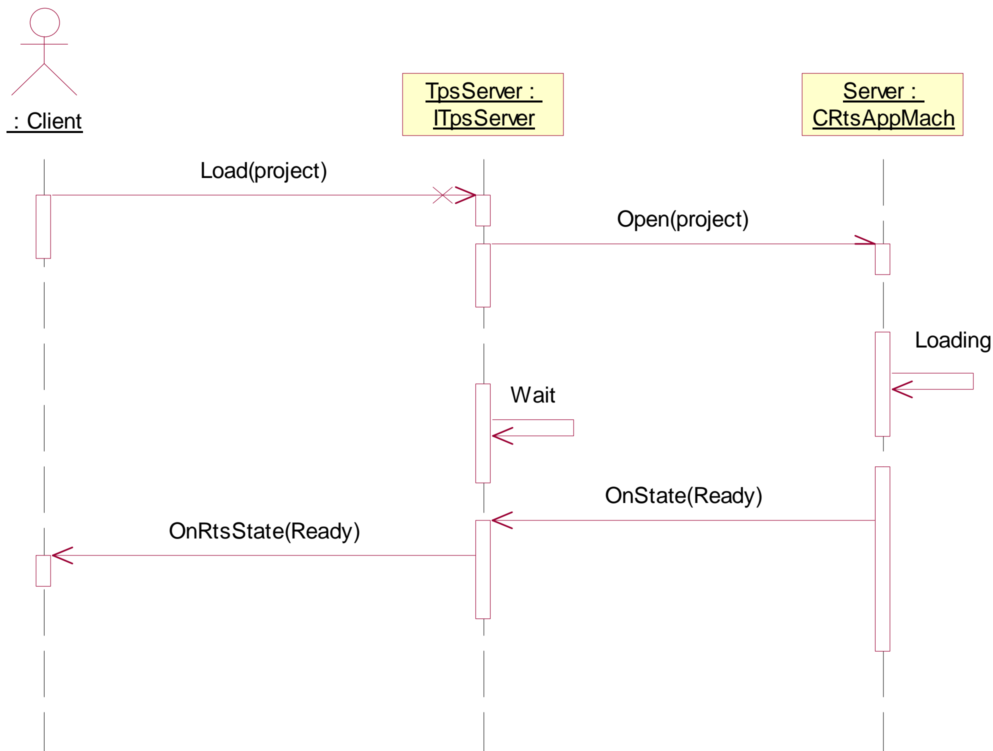


Figure 1 Load Synchronous Sequence Diagram

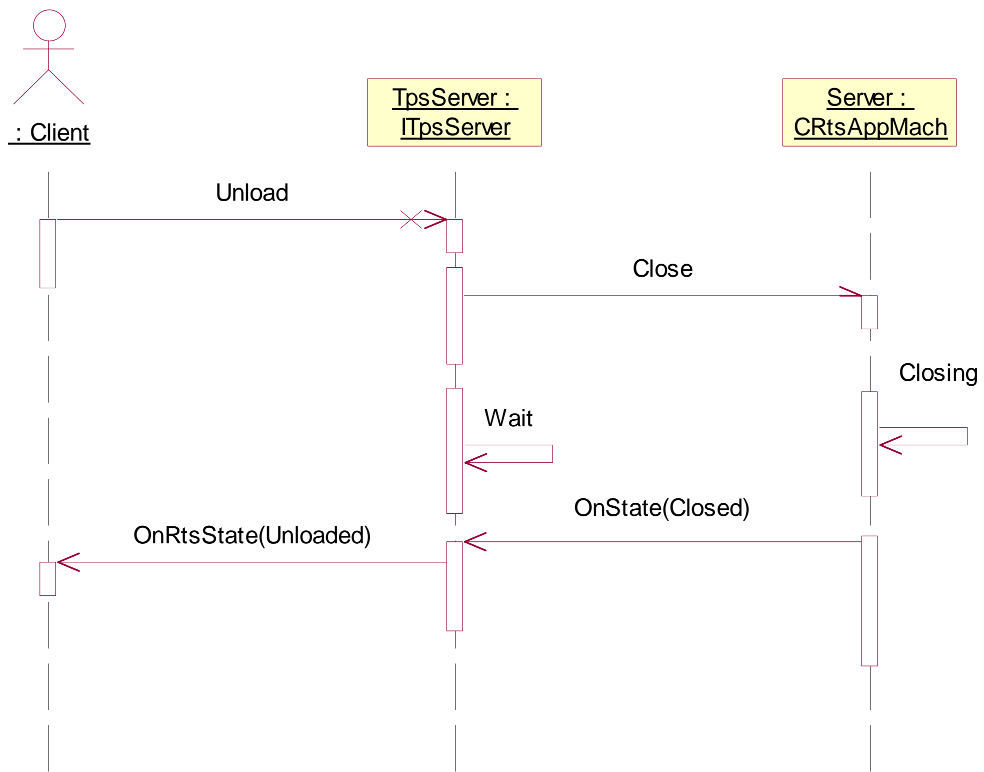


Figure 2 Unload Sequence Diagram

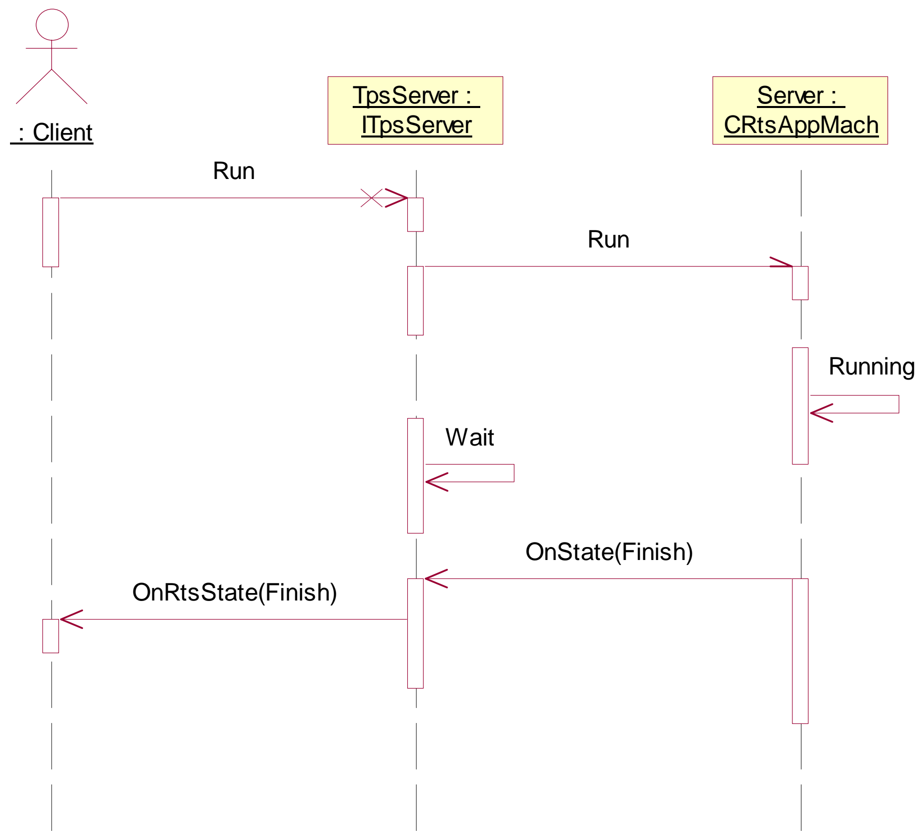


Figure 3 Run Synchronous Sequence Diagram

As shown above, during a synchronous operation the *TpsServer* object waits internally for the started operation to complete. The *Reset* method can be called during a synchronous call to reset the RTS state and return from the synchronous method call immediately. Calling a synchronous method during the execution of another synchronous method will result in erroneous operation.

The synchronous execution mode is designed to be used from simple clients without extensive user interaction to automate the TPS loading, unloading and execution.

3.2 ITpsServerEx Interface

The *ITpsServerEx* interface derives from the *ITpsServer* interface. In addition to the functionality exposed by *ITpsServer* the *ITpsServerEx* interface exposes methods to access predefined global variables, run a particular TPS block and substitute IO resources.

3.2.1 IDL Description

```
[
	object,
	uuid(3F6B2942-F0DA-11D2-BBB0-00C0268914D3),
	dual,
	helpstring("ITpsServer Interface"),
	pointer_default(unique)
]
```

```
interface ITpsServerEx : ITpsServer
{
       [id(11), helpstring("method GetData")]
       HRESULT GetData([in] BSTR strName, [out, retval] VARIANT* pVal);
       [id(12), helpstring("method PutData")]
       HRESULT PutData([in] BSTR strName, [in] VARIANT newVal);
       [id(13), helpstring("method RunBlock")]
       HRESULT RunBlock([in] long lBlockId);
       [id(14), helpstring("method RegisterIOResource")]
       HRESULT RegisterIOResource([in] BSTR sName, [in] IUnknown* pUnkResource);
       [id(15), helpstring("method UnRegisterIOResource")]
       HRESULT UnRegisterIOResource([in] BSTR sName);
};
```

3.2.2 GetData Method

Parameters

Name	Type	Access	Description
strName	BSTR	[in]	Variable name
pVal	VARIANT*	[out, retval]	Pointer to the Variable value

The *GetData* method retrieves the value of the specified predefined variable from the ATLAS program. The available variables depend on the ATLAS subset used. The predefined variables for IEEE.716.89 subsets are `MEASUREMENT, GO, HI, LO, NOGO, MAX-TIME and MANUALINTERVENTION`.

3.2.3 PutData Method

Parameters

Name	Type	Access	Description
strName	BSTR	[in]	Variable name
newVal	VARIANT	[in]	New value

The *PutData* method sets the desired predefined variable to the specified value. The available variables depend on the ATLAS subset used. The predefined variables for IEEE.716.89 subsets are `MEASUREMENT, GO, HI, LO, NOGO, MAX-TIME and MANUALINTERVENTION`.

3.2.4 RunBlock Method

Parameters

Name	Type	Access	Description
lBlockId	long	[in]	Block Number (Block numbers begin from 1….n)

The *RunBlock* method executes the specified TPS block. Two entry points delimit a block. The ATLAS block is identified by the 'E' statement number in the ATLAS program. The ATLAS block can be run in synchronous/asynchronous mode as described by the *Synchronous* property above. The context in which the ATLAS block is run by the RTS Executive is set up through the *RunBlockContext* property described later.

3.2.5 RegisterIOResource Method

Parameters

Name	Type	Access	Description
sName	BSTR	[in]	Name of the IO resource
pUnkResource	IUnknown*	[in]	Interface pointer to the desired resource

The *RegisterIOResource* method replaces the resource specified by the *sName* parameter, with the specified resource. The *pUnkResource* interface pointer parameter must point to a custom I/O resource

object. Custom I/O resource components must implement at least the *IIOResource* and *ITextResource* or *IBinaryResource* interfaces as defined in the I/O Subsystem documents. For details see the I/O Subsystem documentation.

The *RegisterIOResource* method must be called in the “UNLOADED” RTS state. The adapter must be first attached to the RTS Server. The provided resource is used during the following TPS loading. If a resource is already registered an error is reported, to register a new resource call *UnregisterIOResource* before registering the resource.

3.2.6 UnRegisterIOResource Method

Parameters

Name	Type	Access	Description
sName	BSTR	[in]	Name of the IO resource

The *UnRegisterIOResource* method unregisters the registered IO resource specified by the *sName* parameter and restores the default IO resource used by RTS Server.

The *UnRegisterIOResource* method can be called from the “READY”, “FINISH”, “RUNNING”, “HALTED” or “UNLOADED” state. The adapter must be first attached to the RTS Server. It is best to call *UnRegisterIOResource* from the “UNLOADED” state. If *UnRegisterIOResource* is called with a TPS loaded it will take effect only for the next TPS run. Detaching from the server or destroying the adapter object will automatically unregister all registered resources. *UnRegisterIOResource* must be called only before re-registering the resource.
Calling *UnRegisterIOResource* without successfully registering the resource first will generate an error.

3.3 ITpsServerData Interface

The *ITpsServerData* interface is the default interface of the TPS Server object. The *ITpsServerData* interface derives from the *ITpsServerEx* interface. In addition to the functionality exposed by *ITpsServerEx* the *ITpsServerData* interface exposes methods to add and remove watch variables to enhance debug capabilities of the TpsServer.

3.3.1 IDL Description

```
typedef enum RtsAxWatchContext {
       RTSAX_NOTIFY_WHEN_VAR_CHANGES                  = 0,
       RTSAX_NOTIFY_WHEN_VAR_EQUALS_ARG               = 1,
       RTSAX_NOTIFY_WHEN_VAR_DIFFERS_ARG              = 2,
       RTSAX_NOTIFY_AND_HALT_WHEN_VAR_CHANGES         = 3,
       RTSAX_NOTIFY_AND_HALT_WHEN_VAR_EQUALS_ARG      = 4,
       RTSAX_NOTIFY_AND_HALT_WHEN_VAR_DIFFERS_ARG     = 5,
} RtsAxWatchContext;

typedef enum RtsAxRunBlockContext {
       RUN_FROM_BLOCK              = 0,
       RUN_THIS_BLOCK              = 1,
       RUN_TOENDOFCURRENT_BLOCK    = 2,
} RtsAxRunBlockContext;

[
       object,
       uuid(3F6B2943-F0DA-11D2-BBB0-00C0268914D3),
       dual,
       helpstring("ITpsServerData Interface"),
       pointer_default(unique)
]
```

```
interface ITpsServerData : ITpsServerEx
{
        [id(16), helpstring("method AddWatchVariable")]
        HRESULT AddWatchVariable([in] BSTR sName,
                                 [in] IUnknown* pUnkAddressAndTypeInformation,
                                 [in] RtsAxWatchContext eRtsAxWatchContext,
                                 [in] BSTR sArgumentValue,
                                 [in] BSTR sReserved);
        [id(17), helpstring("method RemoveWatchVariable")]
        HRESULT RemoveWatchVariable([in] BSTR sName);
        [id(18), helpstring("method RemoveAllWatchVariables")]
        HRESULT RemoveAllWatchVariables();
        [propput, id(19), helpstring("property Visible")]
        HRESULT Visible([in] VARIANT_BOOL newVal);
        [propget, id(20), helpstring("property RunBlockContext")]
        HRESULT RunBlockContext([out, retval] RtsAxRunBlockContext* pVal);
        [propput, id(20), helpstring("property RunBlockContext")]
        HRESULT RunBlockContext([in] RtsAxRunBlockContext newVal);
        [propget, id(21), helpstring("property EntryBlocks")]
        HRESULT EntryBlocks([out, retval]VARIANT* psaBlockStatements);
};
```

3.3.2 AddWatchVariable Method

Parameters

Name	Type	Access	Description
sName	BSTR	[in]	Name of the variable to be watched.
pUnkAddressAndTypeInformation	IUnknown*	[in]	Interface pointer to the variable's address and data type information
eRtsAxWatchContext	RtsAxWatchContext	[in]	Declares a context to watch a variable.
sArgumentValue	BSTR	[in]	To be used with eRtsAxWatchContext above.
sReserved	BSTR	[in]	Unused.

The AddWatchVariable method is called to add watch variables during debugging a TPS project. Specify a name, Address and Type Information object (described later in the document), valid context (from the enumerations defined) and an argument value.

Symbol	Value	Description
RTSAX_NOTIFY_WHEN_VAR_CHANGES	0	Notify when RTS variable changes.
RTSAX_NOTIFY_WHEN_VAR_EQUALS_ARG	1	Notify when RTS variable equals sArgumentValue.
RTSAX_NOTIFY_WHEN_VAR_DIFFERS_ARG	2	Notify when RTS variable differs from sArgumentValue.
RTSAX_NOTIFY_AND_HALT_WHEN_VAR_CHANGES	3	Notify and halt when RTS variable changes.
RTSAX_NOTIFY_AND_HALT_WHEN_VAR_EQUALS_ARG	4	Notify and halt when RTS variable equals sArgumentValue.
RTSAX_NOTIFY_AND_HALT_WHEN_VAR_DIFFERS_ARG	5	Notify and halt when RTS variable differs from sArgumentValue.

3.3.3 RemoveWatchVariable Method

Parameters

Name	Type	Access	Description

sName	BSTR	[in]	Name of the watch variable that is to be removed.

Call this method with the name of the watch variable that is to be removed.

3.3.4 RemoveAllWatchVariables Method

Call this method to remove all watch variables added while debugging the TPS Project.

3.3.5 Visible Property

Type	Access	Description
VARIANT_BOOL	Write	Controls visibility of the RTS Application

The *Visible* property controls the visibility of the RTS Application. Clients of the TPSServer / TPSServerLite can now control display of the RTS Application through this property.

3.3.6 RunBlockContext Property

Type	Access	Description
VARIANT_BOOL	Read-Write	Used with the RtsAxRunBlockContext.

The *RunBlockContext* property controls the context in which the RTS Executive should run an Entry Block. This context is used by the RTS Executive when the *RunBlock* method is invoked. By default, the TPS Server sets up the *RunBlockContext* to RUN_THIS_BLOCK.

Symbol	Value	Description
RUN_FROM_BLOCK	0	Run the RTS Executive from the Entry Block specified unto the end of the ATLAS program.
RUN_THIS_BLOCK	1	Run the RTS Executive for just the current Entry Block specified.
RUN_TOENDOFCURRENT_BLOCK	2	Run the RTS Executive from its current location until the end of the current Block.

3.3.7 EntryBlocks Property

Type	Access	Description
VARIANT*	Read	Returns a SafeArray of VT_I4 (integers), each integer specifies the Statement Number for the Entry Block.

The *EntryBlocks* property provides access to the Entry Block table of the RTS Executive. Number of elements of the SafeArray returned match exactly to the number of Entry Blocks in the ATLAS program. Each integer value within the SafeArray corresponds to the Entry Block Statement Number in the ATLAS program.

3.4 TpsServer CoClass

3.4.1 IDL Description

```
[
	uuid(3F6B2940-F0DA-11D2-BBB0-00C0268914D3),
	helpstring("TpsServer Class")
]
coclass TpsServer
{
	[default] interface ITpsServerData;
	[default, source] dispinterface _IRtsDataEvents;
```

```
};
```

3.4.2 UML Design

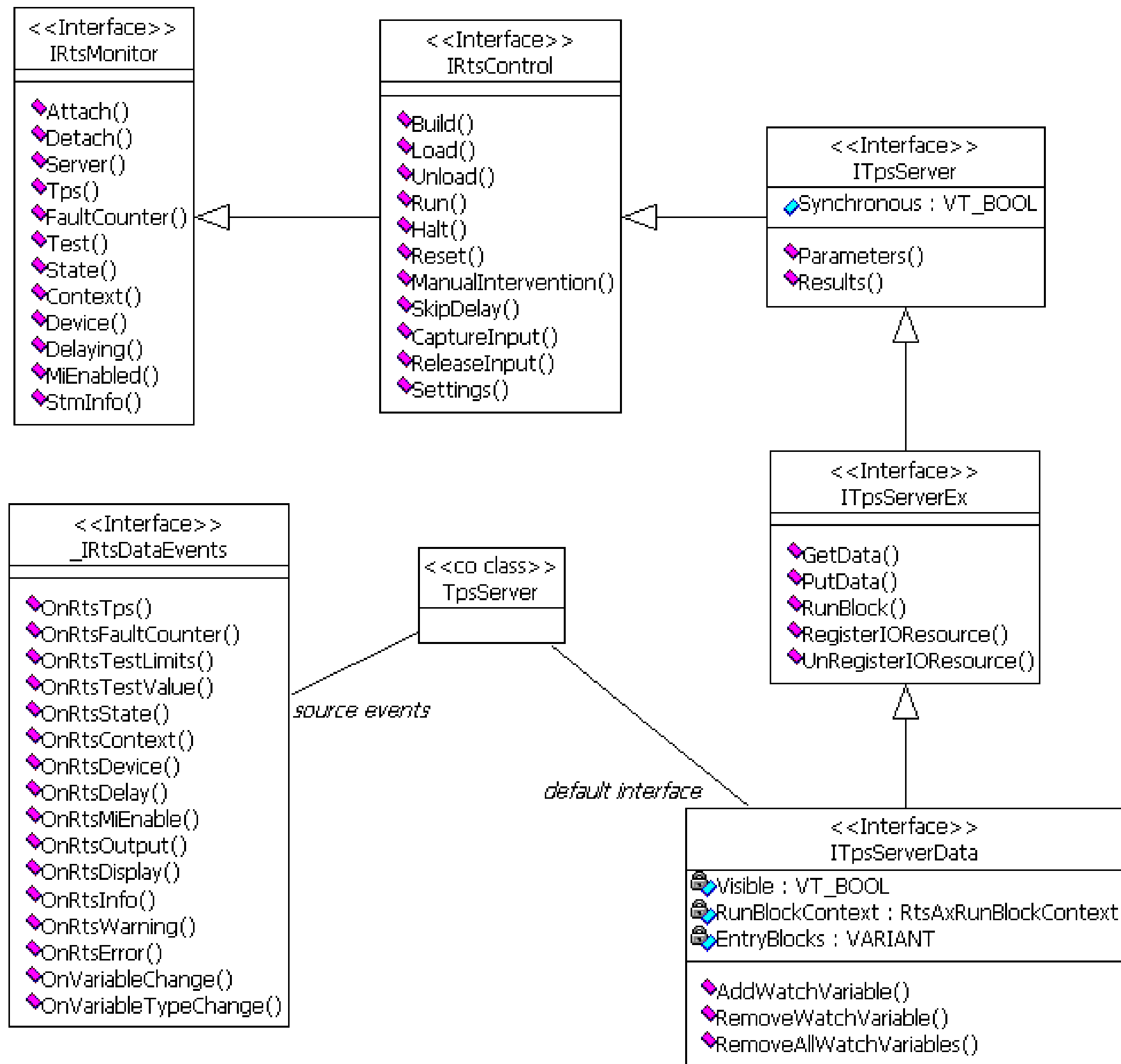


Figure 4 TPS Server class design

3.5 TpsServerLite CoClass

The *TpsServerLite* co-class is a lighter version of the *TpsServer* co-class. Unlike the TpsServer, this co-class does not populate the IOSubSystem with the RESULTS and PARAMETERS data resources.

3.5.1 IDL Description

```
[
	uuid(3F6B2970-F0DA-11D2-BBB0-00C0268914D3),
	helpstring("TpsServerLite Class")
]
coclass TpsServerLite
{
```

```
        [default] interface ITpsServerData;
        [default, source] dispinterface _IRtsDataEvents;
};
```

3.5.2 UML Design

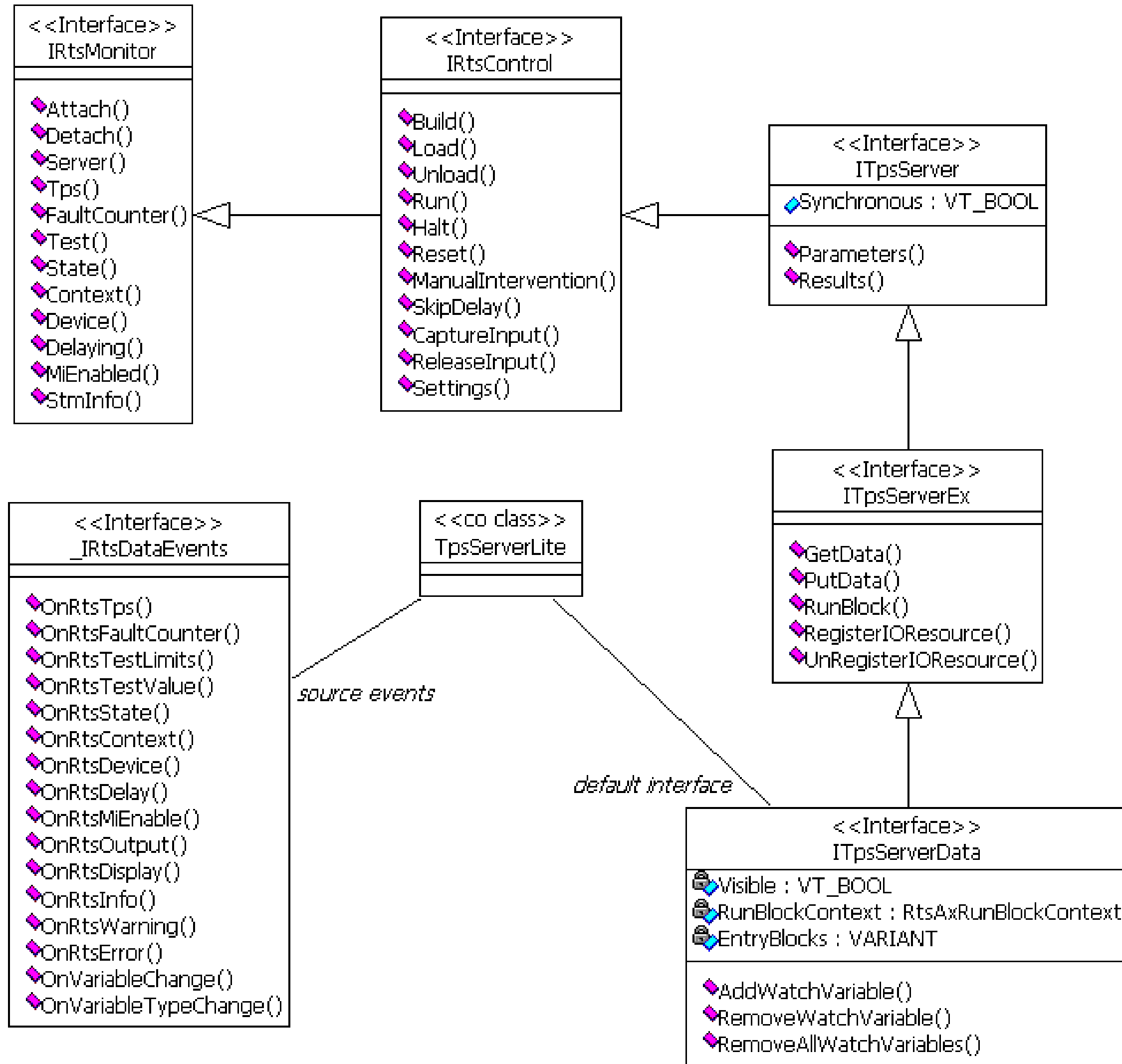


Figure Error! Bookmark not defined. **TpsServerLite class design**

3.6 _IRtsDataEvents Events Interface

The RTS COM adapters fire a number of standard COM events. Using events the server notifies the client when events of interest occur. For implementation, a standard COM solution is used involving connection points and sink interfaces. The client must provide the sink object implementing the required interface. The RTS COM adapters implement the *IConnectionPointContainer* interface.

In order to connect the two objects, the client must call the *Advise* method of the *IConnectionPoint* interface for the connection point of interest. The number of sink objects receiving notification is not subject to any restrictions. To discontinue notification, the client can use the *UnAdvise* method. The mechanism is similar with registering a callback function; in this case it is a callback interface. For details, see the COM specification for outgoing interfaces, connection points and sink objects.

The *_IRtsDataEvents* interface groups the events related to RTS activity.

3.6.1 IDL description

```
[
	uuid(3F6B2909-F0DA-11D2-BBB0-00C0268914D3),
	helpstring("_IRtsDataEvents Interface")
]
dispinterface _IRtsDataEvents
{
	properties:
	methods:
	[id(1), helpstring("method OnRtsTps")]
	HRESULT OnRtsTps([in] BSTR strTps);
	[id(2), helpstring("method OnRtsFaultCounter")]
	HRESULT OnRtsFaultCounter([in] long lFC);
	[id(3), helpstring("method OnRtsTestLimits")]
	HRESULT OnRtsTestLimits([in] IDispatch* pTest);
	[id(4), helpstring("method OnRtsTestValue")]
	HRESULT OnRtsTestValue([in] IDispatch* pTest);
	[id(5), helpstring("method OnRtsState")]
	HRESULT OnRtsState([in] long lState);
	[id(6), helpstring("method OnRtsContext")]
	HRESULT OnRtsContext([in] long lContext);
	[id(7), helpstring("method OnRtsDevice")]
	HRESULT OnRtsDevice([in] BSTR strDevice);
	[id(8), helpstring("method OnRtsDelay")]
	HRESULT OnRtsDelay([in] double dTime);
	[id(9), helpstring("method OnRtsMiEnable")]
	HRESULT OnRtsMiEnable([in] VARIANT_BOOL bEnable);
	[id(10), helpstring("method OnRtsOutput")]
	HRESULT OnRtsOutput([in] BSTR strMsg);
	[id(11), helpstring("method OnRtsDisplay")]
	HRESULT OnRtsDisplay([in] BSTR strMsg);
	[id(12), helpstring("method OnRtsInfo")]
	HRESULT OnRtsInfo([in] BSTR strMsg);
	[id(13), helpstring("method OnRtsWarning")]
	HRESULT OnRtsWarning([in] BSTR strMsg);
	[id(14), helpstring("method OnRtsError")]
	HRESULT OnRtsError([in] BSTR strMsg);
	[id(15), helpstring("method OnVariableChange")]
	HRESULT OnVariableChange([in] BSTR name,
	                         [in] BSTR value,
	                         [in] long vlc);
	[id(16), helpstring("method OnVariableTypeChange")]
	HRESULT OnVariableTypeChange([in] BSTR name,
	                             [in] long value,
	                             [in] long vlc);
};
```

3.6.2 OnRtsTps Event

Parameters

Name	Type	Access	Description
strTps	BSTR	[in]	Full TPS path

The *OnRtsTps* event is fired when the *Tps* property changes. The *strTps* parameter is null when the TPS was unloaded or the loading failed.

3.6.3 OnRtsFaultCount Event

Parameters

Name	Type	Access	Description
lFC	long	[in]	TPS fault counter

The *OnRtsFaultCount* event is fired when the *FaultCount* property changes.

3.6.4 OnRtsTestLimits Event

Parameters

Name	Type	Access	Description
pTest	IDispatch*	[in]	Pointer to the IDispatch interface of the test object.

The *OnRtsTestLimits* event is fired during a test when test limits are available. For a signal-based test, this notification occurs before the actual measurement. The provided test object contains only the limits and dimension information.

3.6.5 OnRtsTestValue Event

Parameters

Name	Type	Access	Description
pTest	IDispatch*	[in]	Pointer to the IDispatch interface of the test object.

The *OnRtsTestValue* event is fired during a test when the comparison is executed. For a signal-based test, this notification occurs after the actual measurement. The provided test object is fully populated.

3.6.6 OnRtsState Event

Parameters

Name	Type	Access	Description
lState	long	[in]	The new state.

The *OnRtsState* event is fired when the RTS state changes.

3.6.7 OnRtsContext Event

Parameters

Name	Type	Access	Description
lContext	long	[in]	The new context.

The *OnRtsContext* event is fired when the RTS context is changed.

3.6.8 OnRtsDevice Event

Parameters

Name	Type	Access	Description
strDevice	BSTR	[in]	The new active device

The *OnRtsDevice* event is fired when the current device changes.

3.6.9 OnRtsDelay Event

Parameters

Name	Type	Access	Description
dTime	double	[in]	Remaining delay amount in seconds

The *OnRtsDelay* event is fired when the RTS is delaying. The *OnRtsDelay* event is fired periodically during a delay notifying the client of the delay progress.

3.6.10 OnRtsMiEnable Event

Parameters

Name	Type	Access	Description
bEnable	VARIANT_BOOL	[in]	Desired action - enable or disable

The *OnRtsMiEnable* event is fired when the Manual Intervention action is enabled or disabled

3.6.11 OnRtsOutput Event

Parameters

Name	Type	Access	Description
strMsg	BSTR	[in]	Text to display

The *OnRtsOutput* event is fired by ATLAS OUTPUT, to 'DISPLAY' statements.

3.6.12 OnRtsDisplay Event

Parameters

Name	Type	Access	Description
strMsg	BSTR	[in]	Text to display

The *OnRtsDisplay* event is fired by using the display function in MACRO or CEM device drivers.

3.6.13 OnRtsInfo Event

Parameters

Name	Type	Access	Description
strMsg	BSTR	[in]	Information message to display

The *OnRtsInfo* event is fired by information messages.

3.6.14 OnRtsWarning Event

Parameters

Name	Type	Access	Description
strMsg	BSTR	[in]	Warning message to display

The *OnRtsWarning* event is fired by warning messages.

3.6.15 OnRtsError Event

Parameters

Name	Type	Access	Description
strMsg	BSTR	[in]	Error message to display

The *OnRtsError* event is fired by error messages.

3.6.16 OnVariableChange Event

Parameters

Name	Type	Access	Description
name	BSTR	[in]	Name of the RTS variable under watch.
value	BSTR	[in]	Value f the RTS variable under watch.
vlc	long	[in]	Address of the AIL instruction that caused the variable value to be changed.

The *OnVariableChange* event is fired when a RTS variable under watch changes.

3.6.17 OnVariableTypeChange Event

Parameters

Name	Type	Access	Description
name	BSTR	[in]	Name of the RTS variable under watch.
value	Long	[in]	Value f the RTS variable under watch.
vlc	long	[in]	Address of the AIL instruction that caused the variable type to be changed.

The *OnVariableTypeChange* event is fired when the type of the variable under watch changes. Value would correspond to a type within the `RtsVarTypes enumeration defined later.`

4 Additional COM Components, Interfaces and Types

4.1 IAddressInformation

The interface is used to define address information and how ATE variables are stored and manipulated in the RTS adapters.

4.1.1 IDL Description

```
[
	object,
	uuid(3F6B2981-F0DA-11D2-BBB0-00C0268914D3),
	dual,
	helpstring("IAddressInformation Interface"),
	pointer_default(unique)
]
interface IAddressInformation : IDispatch
{
	[propget, id(1), helpstring("property Vad")]
	HRESULT Vad([out, retval] long *pVal);
	[propget, id(2), helpstring("property FieldFrom")]
	HRESULT FieldFrom([out, retval] long *pVal);
	[propget, id(3), helpstring("property FieldLength")]
	HRESULT FieldLength([out, retval] long *pVal);
	[id(4), helpstring("method Populate")]
	HRESULT Populate([in] long lVad,
	                 [in] long lFieldFrom,
	                 [in] long lFieldLength);
};
```

4.1.2 Vad Property

Type	Access	Description
Long	Read-Only	Virtual address location of the variable

Use this property to get the virtual address of a variable in the RTS.

4.1.3 FieldFrom Property

Type	Access	Description
Long	Read-Only	Field description start bit

Use this property to define the description start bit. It describes storage of digital fields which is supported by specific ATE subsets.

4.1.4 FieldLength Property

Type	Access	Description
Long	Read-Only	Field length (in bits) of the variable

Use this property to define the Field length of variable in RTS. It describes storage field length of digital fields which is supported by specific ATE subsets.

4.1.5 Populate Method

Parameters

Name	Type	Access	Description
lVad	long	[in]	Virtual address location of the variable
lFieldFrom	long	[in]	Field description start bit
lFieldLength	long	[in]	Field length (in bits) of the variable

Use this method to update variables parameters. *lFieldFrom* and *lFieldLength* are only used in defining storage fields of digital variables which is supported by specific ATE subsets.

4.2 AddressInformation CoClass

This co-class is the entity that maintains the watch variable information between two debug sessions.

4.2.1 IDL Description

```
[
	uuid(3F6B2980-F0DA-11D2-BBB0-00C0268914D3),
	helpstring("AddressInformation Class")
]
coclass AddressInformation
{
	[default] interface IAddressInformation;
};
```

4.2.2 UML Design

Figure Error! Bookmark not defined. **AddressInformation class design**

4.3 IAddressAndTypeInformation

The interface derives from *IAddressInformation*. It further defines the type of the watched variable.

4.3.1 IDL Description

```
typedef enum RtsVarTypes {
	RTSAX_TYPE_BOOL      = 0,
	RTSAX_TYPE_INT       = 1,
	RTSAX_TYPE_REAL      = 2,
	RTSAX_TYPE_TEXT      = 4,
	RTSAX_TYPE_CON89     = 6,
	RTSAX_TYPE_DIGITAL   = 7,
	RTSAX_TYPE_CON85     = 9,
	RTSAX_TYPE_MASK      = 0x0F,
	RTSAX_EXCESS_MASK    = 0xF0,
	RTSAX_SIZE_MASK      = 0xFF00,
} RtsVarTypes;

[
	object,
	uuid(3F6B2991-F0DA-11D2-BBB0-00C0268914D3),
	dual,
	helpstring("IAddressAndTypeInformation Interface"),
	pointer_default(unique)
]
interface IAddressAndTypeInformation : IAddressInformation
{
	[propget, id(5), helpstring("property TypeWord")]
	HRESULT TypeWord([out, retval] unsigned short *pVal);
	[propget, id(6), helpstring("property TypeAsString")]
	HRESULT TypeAsString([out, retval] BSTR *pVal);
	[id(7), helpstring("method Populate")]
	HRESULT Populate([in] long lVad,
	                 [in] long lFieldFrom,
	                 [in] long lFieldLength,
	                 [in] unsigned short ushTypeWord);
};
```

4.3.2 TypeWord Method

Parameters

Name	Type	Access	Description
pVal	unsigned short	[out, retval]	Value of the variable

This method retrieves the value of the variable as a word.

4.3.3 TypeAsString Method

Parameters

Name	Type	Access	Description
pVal	BSTR	[out, retval]	Value pointed to by the variable

This method retrieves the string pointed to by the variable.

4.3.4 Populate Method

Parameters

Name	Type	Access	Description
lVad	long	[in]	Virtual address location of the variable
lFieldFrom	long	[in]	Field description start bit
lFieldLength	long	[in]	Field length (in bits) of the variable
ushTypeWord	Unsigned short	[in]	Value of the variable

The overloaded method of this interface is used to update variables parameters. *lFieldFrom* and *lFieldLength* are only used in defining storage fields of digital variables which are supported by specific ATE subsets.

4.4 AddressAndTypeInformation CoClass

4.4.1 IDL Description

```
[
        uuid(3F6B2990-F0DA-11D2-BBB0-00C0268914D3),
        helpstring("AddressAndTypeInformation Class")
]
coclass AddressAndTypeInformation
{
        [default] interface IAddressAndTypeInformation;
};
```

4.4.2 UML Design

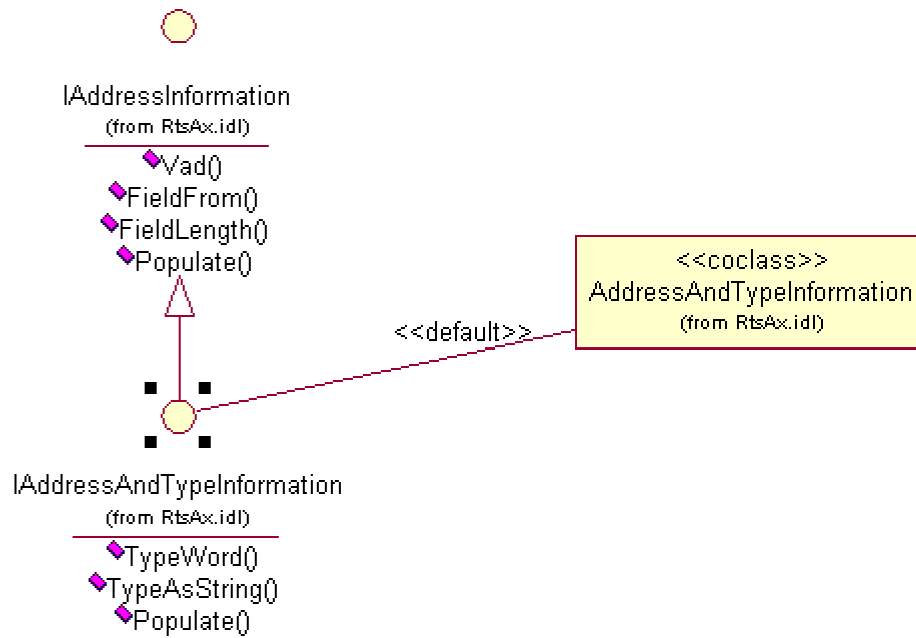


Figure Error! Bookmark not defined. **AddressAndTypeInformation class design**

Note:
All additional interfaces, types and components referred by this document are described in the *COM Utils*, *IOSubsystem* or *RTS COM Adapters* documents.