1 Scope

This document describes RTS COM Adapters. The RTS COM Adapters provide support for RTS monitoring and control. The document describes the available services, COM components and interfaces for PAWS Studio.

2 Overview

The RTS COM Adapters are a set of components that:

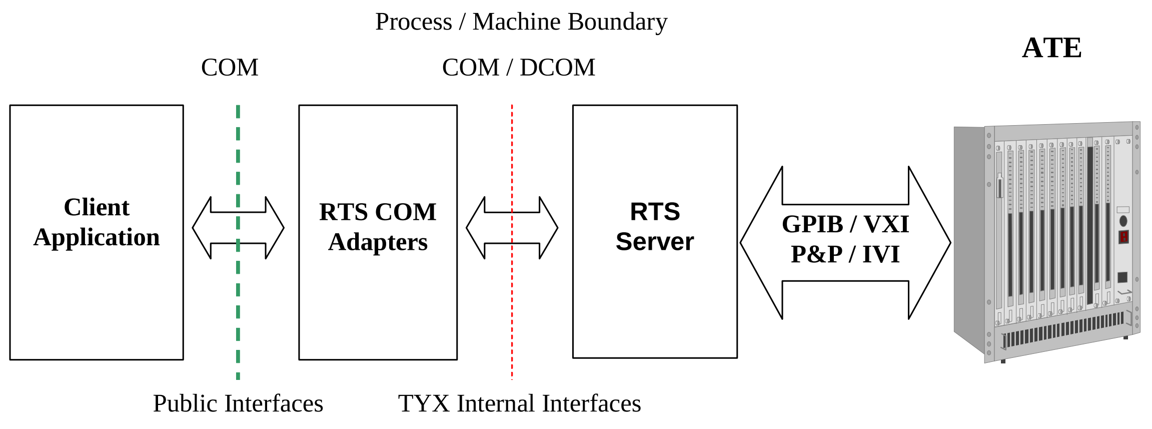
- • Provide access to the RTS Server as an open server component capable of runtime collaboration with other control and monitoring components
- • Provide a stable interface to access the RTS executive functionality insulating the client applications from internal RTS server interface or semantic changes
- • Employ dispatch interfaces compatible with Visual Basic, C++, C#, Java Script, VB Script bindings
- • Generate a variety of COM events during the operation of the RTS Executive
- • Provide full configuration support for the RTS Server
- • Provide support for I/O interception
- • Support RTS distribution

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

An RTS COM Adapter 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 RTS COM Adapters are designed and maintained by TYX to provide a public and stable interface to the RTS Server. Client applications can use the RTS COM Adapters components and interfaces; TYX will maintain and ensure compatibility between the RTS COM Adapters and the RTS Server.



Each RTS COM Adapter is specialized to provide a well-defined set of services and functionality. Two or more adapters can be attached to the same RTS Server to combine the functionality provided individually.

3 RTS COM Adapters Base Interfaces

All RTS COM Adapters implement one or more of the following base interfaces. Please see the documentation for the specific adapter for details. The RTS COM Adapters Base 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”.

3.1 IRtsMonitor Interface

The *IRtsMonitor* interface groups methods and properties to monitor the RTS Server. It provides complete read-only access to the state of the RTS Server. The RTS Server must be controlled by other means. *IRtsMonitor* is a dual interface, derives from *IDispatch* and supports automation fully.

3.1.1 IDL Description

```
typedef enum RtsAxState {
      RTSAX_STATE_DETACHED        = -1,
      RTSAX_STATE_UNLOADED        = 0,
      RTSAX_STATE_READY           = 1,
      RTSAX_STATE_RUNNING         = 2,
      RTSAX_STATE_HALTED          = 3,
      RTSAX_STATE_FINISH          = 4,
} RtsAxState;

typedef enum RtsAxContext {
      RTSAX_CTX_DETACHED          = -1,
      RTSAX_CTX_UNLOADED          = 0,
      RTSAX_CTX_INIT              = 1,
      RTSAX_CTX_TPS               = 2,
      RTSAX_CTX_TERM              = 3,
} RtsAxContext;

[
      object,
      uuid(3F6B2901-F0DA-11D2-BBB0-00C0268914D3),
      dual,
      helpstring("IRtsMonitor Interface"),
      pointer_default(unique)
]
interface IRtsMonitor : IDispatch
{
      [id(DISPID_ATTACH), helpstring("method Attach")]
      HRESULT Attach([in, optional] VARIANT varServer);
      [id(DISPID_DETACH), helpstring("method Detach")]
      HRESULT Detach();
      [propget, id(DISPID_SERVER), helpstring("property Server")]
      HRESULT Server([out, retval] IDispatch** pVal);
      [propget, id(DISPID_TPS), helpstring("property Tps")]
      HRESULT Tps([out, retval] BSTR* pVal);
      [propget, id(DISPID_FAULTCOUNTER), helpstring("property FaultCounter")]
      HRESULT FaultCounter([out, retval] long* pVal);
      [propget, id(DISPID_TEST), helpstring("property Test")]
      HRESULT Test([out, retval] IDispatch** pVal);
      [propget, id(DISPID_STATE), helpstring("property State")]
      HRESULT State([out, retval] long* pVal);
      [propget, id(DISPID_CONTEXT), helpstring("property Context")]
      HRESULT Context([out, retval] long* pVal);
      [propget, id(DISPID_DEVICE), helpstring("property Device")]
      HRESULT Device([out, retval] BSTR* pVal);
      [propget, id(DISPID_DELAYING), helpstring("property Delaying")]
```

```
        HRESULT Delaying([out, retval] VARIANT_BOOL* pVal);
        [propget, id(DISPID_MIENABLED), helpstring("property MiEnabled")]
        HRESULT MiEnabled([out, retval] VARIANT_BOOL* pVal);
        [propget, id(DISPID_STMINFO), helpstring("property StmInfo")]
        HRESULT StmInfo([out, retval] IDispatch** pVal);
};
```

3.1.2 Attach Method

Parameters

Name	Type	Access	Description
varServer	VARIANT	[in, optional]	RTS Server or the host computer.

The *Attach* method connects the adapter object to the RTS Server specified by the *varServer* parameter. The possible types and values of the *varServer* VARIANT parameter are:

1. 1. VT_EMPTY or VT_NULL – in this case the adapter object will connect to the RTS Server running on the local machine. If there is no RTS Server running a new instance will be created on the local machine.
2. 2. VT_ERROR – this case occurs from scripting languages when the optional parameter is omitted. The adapter object will connect to the RTS Server running on the local machine. If there is no RTS Server running a new instance will be created on the local machine.
3. 3. VT_UNKNOWN or VT_DISPATCH – the provided interface pointer must point to the RtsServer component. Such a pointer can be obtained using the *Server* property of another adapter object.
4. 4. VT_BSTR – the parameter is interpreted as the physical location of the RTS Server to connect to. The location can be specified as Windows computer name or as IP address – for example ‘TYXATE’ or ‘192.168.100.14’. If the RTS Server is not running on the specified machine a new instance will be created. DCOM security settings have to be set to allow RTS Server creation and access on the remote machine for all authorized users. Event notifications from RTS Server to the client machine must also be allowed. For details please see the Microsoft DCOM security settings documentation. If the string parameter is null or empty the local machine will be used as host.
5. 5. All other types will generate an error.

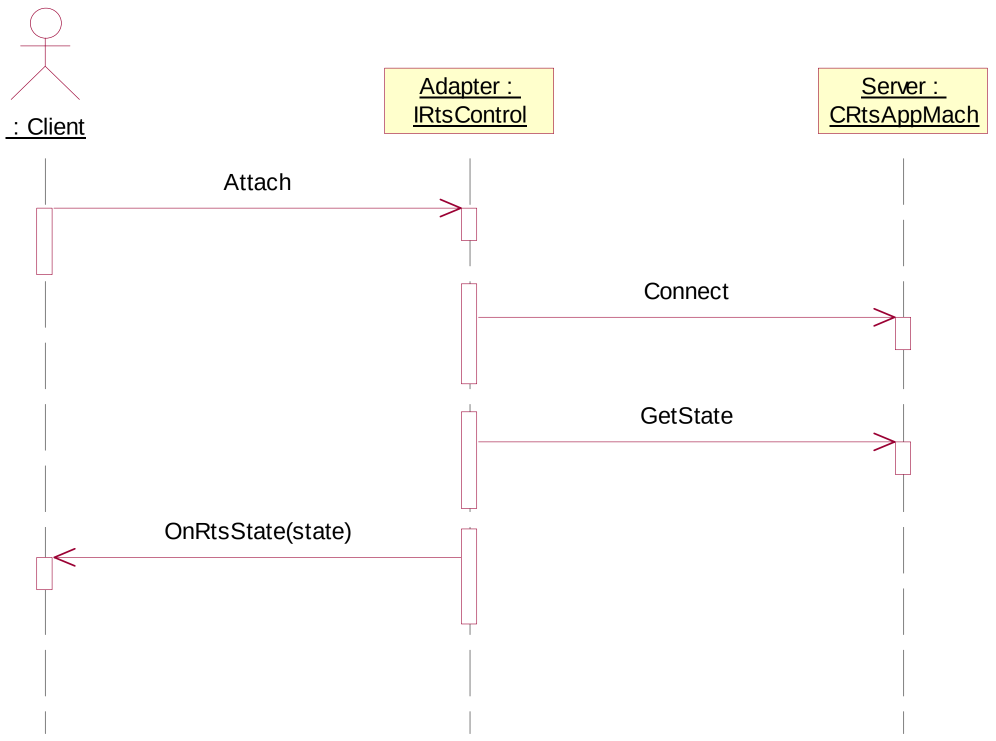


Figure 1 Attach Sequence Diagram

As depicted in the sequence diagram above the adapter object will fire an *OnRtsState* event with the current server state after establishing the connection.
The *Attach* method does not change the configuration or state of the RTS Server. If a server is already attached and an *Attach* method is called the adapter object will first *Detach* from the existing server.

3.1.3 Detach Method

The *Detach* method detaches the RTS Server from the adapter object. It is intended to be called only after a successful *Attach* but it can be safely called even if *Attach* was not called or was called but failed. The *Detach* method is called automatically when the adapter object is destroyed, consequently calling *Detach* before releasing the adapter object is not mandatory.

3.1.4 Server Property

Type	**Access**	**Description**
IDispatch*	Read-Only	Interface pointer to RTS Server

The Server property contains an *IDispatch* interface pointer to the RTS Server the adapter object is connected to. The property is read-only; to change it the *Attach* method can be used. The obtained interface pointer can be used only for attaching purposes. All RTS Server interfaces are for TYX internal use only; as the product evolves they are maintained and changed periodically.

3.1.5 Tps Property

Type	**Access**	**Description**
BSTR	Read-Only	TPS full file path.

The *Tps* property contains the full path to the currently loaded TPS. The property is NULL when no project is loaded or the adapter object is disconnected. The property is read-only, to change it a control interface must to be used to load a different TPS using the *Load* method. An *OnRtsTps* event is fired when the *Tps* property changes.

3.1.6 FaultCounter Property

Type	**Access**	**Description**
long	`Read-Only`	The number of faults occurred during the current execution

The *FaultCounter* property indicates how many faults, or NOGO conditions, were encountered during the current execution. An execution is considered complete when the RTS Finish state is reached. The fault counter is incremented for each NOGO condition generated by compare, verify or prove ATLAS statements. The fault counter is reset to 0 every time the program execution is restarted. The *FaultCounter* property is read-only, only the RTS server can modify it. The reported value is 0 when the adapter object is detached from the RTS Server. An *OnRtsFaultCounter* event is fired when the *FaultCounter* property changes.

3.1.7 Test Property

Type	**Access**	**Description**
IDispatch*	`Read-Only`	Pointer to the *IDispatch* interface of the test object.

The *Test* property provides access to the last test object. The *IDispatch* interface pointer points to an *AtlasTest* object also exposing the *IAtlasTest* interface. The test object contains information about the last test such as the limits, the measurement and the result. For details please see the *AtlasTest* component documentation provided in the “COM Utilities” section. The content of the test object is modified dynamically by the RTS; a “deep copy” is required to store a copy of the test object. During the test execution the content of the test object is collected. Before the measurement only the test limits are available. The *OnRtsTestLimits* and *OnRtsTestValue* events are fired when the test object is changed. This property is read-only; it can be changed only by the RTS Server. A null interface pointer is returned when the adapter object is disconnected from the RTS Server.

3.1.8 State Property

Type	**Access**	**Description**
long	`Read-Only`	The state of the RTS COM Adapter

The *State* property provides access to the state of the adapter object. The property is read-only; it can be changed only by the RTS Server or by using the *Attach / Detach* methods. An *OnRtsState* event is fired when the *State* property changes.

When the adapter is connected to the server its state is numerically the same with the state of the RTS Server. The possible state values are specified by the `RtsAxState` enumeration contained by the IDL description and the RtsAx type library.

Symbol	**Value**	**Description**
`RTSAX_STATE_DETACHED`	`-1`	Adapter object is disconnected from RTS Server
`RTSAX_STATE_UNLOADED`	`0`	No TPS was loaded or TPS was unloaded
`RTSAX_STATE_READY`	`1`	TPS loaded, server initialized and ready for execution
`RTSAX_STATE_RUNNING`	`2`	Server is running TPS
`RTSAX_STATE_HALTED`	`3`	TPS execution is halted
`RTSAX_STATE_FINISH`	`4`	TPS execution complete, FINISH or TERMINATE reached

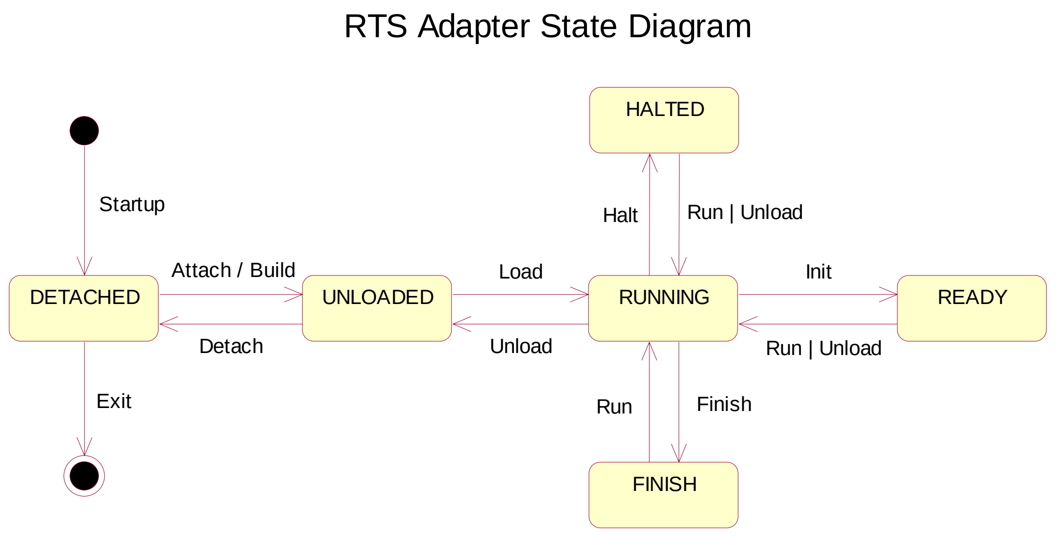


Figure 2 RTS COM Adapter State Diagram

3.1.9 Context Property

Type	**Access**	**Description**
long	`Read-Only`	Context of execution

The *Context* property provides access to the execution context of the adapter object. The property is read-only; it can be changed only by the RTS Server or by using the *Attach / Detach* methods. The execution context provides additional information about the complete state of the RTS Server. For example the server can go thru the *Running* and *Halted* states during the TPS execution or during the TPS initialization or termination. The actions to respond to a warning or error condition or an external event can take into consideration both the state and the execution context of the RTS Server. The same event may be handled differently if it occurs during TPS initialization, execution or termination. An *OnRtsContext* event is fired when the *Context* property changes.
When the adapter is connected to the server its context is numerically the same with the context of the RTS Server. The possible state values are specified by the `RtsAxContext` enumeration contained by the IDL description and the RtsAx type library.

Symbol	**Value**	**Description**
RTSAX_ **CTX_DETACHED**	-1	Adapter object is disconnected from RTS Server
RTSAX_ **CTX_UNLOADED**	0	The RTS Server is in TPS unloaded context
RTSAX_ **CTX_INIT**	1	The RTS Server is in TPS initialization context
RTSAX_ **CTX_TPS**	2	The RTS Server is in TPS execution context
RTSAX_	3	The RTS Server is in TPS unloading or termination context

3.1.10 Device Property

Type	Access	Description
BSTR	Read-Only	The current or last device accessed by RTS

The *Device* property provides access to the name of the last device accessed by the RTS. The property is read-only; it can be changed only by the RTS Server. The *Device* property is changed each time the RTS changes the current device, before the actual device access. An *OnRtsDevice* event is fired whenever the *Device* property changes. When the adapter object is disconnected from the RTS Server the *Device* property value is null.

3.1.11 Delaying Property

Type	Access	Description
VARIANT_BOOL	Read-Only	Indicates if the RTS Server is delaying

The *Delaying* property indicates if the RTS is delaying. The property is designed to be used to enable or disable the "SkipDelay" button in the GUI. An *OnRtsDelay* event is fired when the *Delaying* property changes. The *Delaying* property is read-only; it can be changed only by the RTS Server. The RTS Server can be instructed to skip the delay using the "SkipDelay" method in a control interface. When the adapter object is disconnected from the RTS Server the *Delaying* property value is VARIANT_FALSE.

3.1.12 MiEnabled Property

Type	Access	Description
VARIANT_BOOL	Read-Only	Indicates if the manual intervention is enabled

The *MiEnabled* property indicates if the "Manual Intervention" action is enabled. The property is designed to be used to enable or disable the "Manual Intervention" button in the GUI. An *OnRtsMiEnable* event is fired when the *MiEnabled* property changes. The *MiEnabled* property is read-only; it can be changed only by the RTS Server. When the adapter object is disconnected from the RTS Server the *MiEnabled* property value is VARIANT_FALSE.

3.1.13 StmInfo Property

Type	Access	Description
IDispatch*	Read-Only	An interface pointer to the StmInfo object

The *StmInfo* property provides access to the statement information object maintained by the RTS Server. The *IDispatch* interface pointer points to a *StmInfo* object also exposing the *IStmInfo* interface. The statement information object contains information about the current ATLAS statement test such as the statement number, the ATLAS module, the ATLAS verb, etc. For details please see the *StmInfo* component documentation provided in the "COM Utilities" section. The content of the statement information object is modified dynamically by the RTS; a "deep copy" is required to store a copy of the test object. There are no events fired when the statement information object changes. The *StmInfo* property is provided to support client applications in characterizing accurately the TPS execution point when events of interest occur. This property is read-only; it can be changed only by the RTS Server. A null interface pointer is returned when the adapter object is disconnected from the RTS Server; the client must handle the null pointer safely.

3.1.14 Dispatch Identifiers

```
const DISPID DISPID_ATTACH              = 31;
```

```
        const DISPID DISPID_DETACH                  = 32;
        const DISPID DISPID_SERVER                  = 33;
        const DISPID DISPID_TPS                     = 34;
        const DISPID DISPID_FAULTCOUNTER            = 35;
        const DISPID DISPID_TEST                    = 36;
        const DISPID DISPID_STATE                   = 37;
        const DISPID DISPID_CONTEXT                 = 38;
        const DISPID DISPID_DEVICE                  = 39;
        const DISPID DISPID_DELAYING                = 40;
        const DISPID DISPID_MIENABLED               = 41;
        const DISPID DISPID_STMINFO                 = 42;
```

3.2 IRtsControl Interface

The *IRtsControl* interface groups methods and properties to control the RTS Server. The *IRtsControl* interface derives from *IRtsMonitor* interface and provides unrestricted access to the functionality exposed by the RTS Server. *IRtsControl* is a control or full access interface implemented as a dual interface that derives indirectly from *IDispatch* and supports automation.

3.2.1 IDL Description

```
[
        object,
        uuid(3F6B2902-F0DA-11D2-BBB0-00C0268914D3),
        dual,
        helpstring("IRtsControl Interface"),
        pointer_default(unique)
]
interface IRtsControl : IRtsMonitor
{
        [id(DISPID_BUILD), helpstring("method Build")]
        HRESULT Build([in, optional] VARIANT varHost);
        [id(DISPID_LOAD), helpstring("method Load")]
        HRESULT Load([in] BSTR strTps);
        [id(DISPID_UNLOAD), helpstring("method Unload")]
        HRESULT Unload();
        [id(DISPID_RUN), helpstring("method Run")]
        HRESULT Run();
        [id(DISPID_HALT), helpstring("method Halt")]
        HRESULT Halt();
        [id(DISPID_RESET), helpstring("method Reset")]
        HRESULT Reset();
        [id(DISPID_MANINTV), helpstring("method ManualIntervention")]
        HRESULT ManualIntervention();
        [id(DISPID_SKIPDLY), helpstring("method SkipDelay")]
        HRESULT SkipDelay();
        [id(DISPID_CAPINPUT), helpstring("method CaptureInput")]
        HRESULT CaptureInput([in, optional] VARIANT varRsrc);
        [id(DISPID_RELINPUT), helpstring("method ReleaseInput")]
        HRESULT ReleaseInput();
        [propget, id(DISPID_SETTINGS), helpstring("property Settings")]
        HRESULT Settings([out, retval] IRtsSettings** pVal);
};
```

3.2.2 Build Method

Parameters

Name	**Type**	**Access**	**Description**
varHost	VARIANT	[in, optional]	Name or IP address of the host computer

The *Build* method reconfigures the RTS Server object as specified by the *Settings* property and connects the adapter object to the RTS Server specified by the *varHost* parameter. The *Build* method is designed to create the RTS Server and configure it before connecting the adapter. If the RTS Server is already running it can be reconfigured only if the TPS is unloaded. To connect to the RTS Server without reconfiguring it, use the *Attach* method exposed by the *IRtsMonitor* interface.

The possible types and values of the *varHost* VARIANT parameter are:

1. 1. VT_EMPTY or VT_NULL – in this case the adapter object will connect to the RTS Server running on the local machine. If there is no RTS Server running a new instance will be created on the local machine.
2. 2. VT_ERROR – this case occurs from scripting languages when the optional parameter is omitted. The adapter object will connect to the RTS Server running on the local machine. If there is no RTS Server running a new instance will be created on the local machine.
3. 3. VT_BSTR – the parameter is interpreted as the physical location of the RTS Server to connect to. The location can be specified as Windows computer name or as IP address – for example 'TYXATE' or '192.168.100.14'. If the RTS Server is not running on the specified machine a new instance will be created. DCOM security settings have to be set to allow RTS Server creation and access on the remote machine for all authorized users. Event notifications from RTS Server to the client machine must also be allowed. For details please see the Microsoft DCOM security settings documentation. If the string parameter is null or empty the local machine will be used as host.
4. 4. All other types will generate an error.

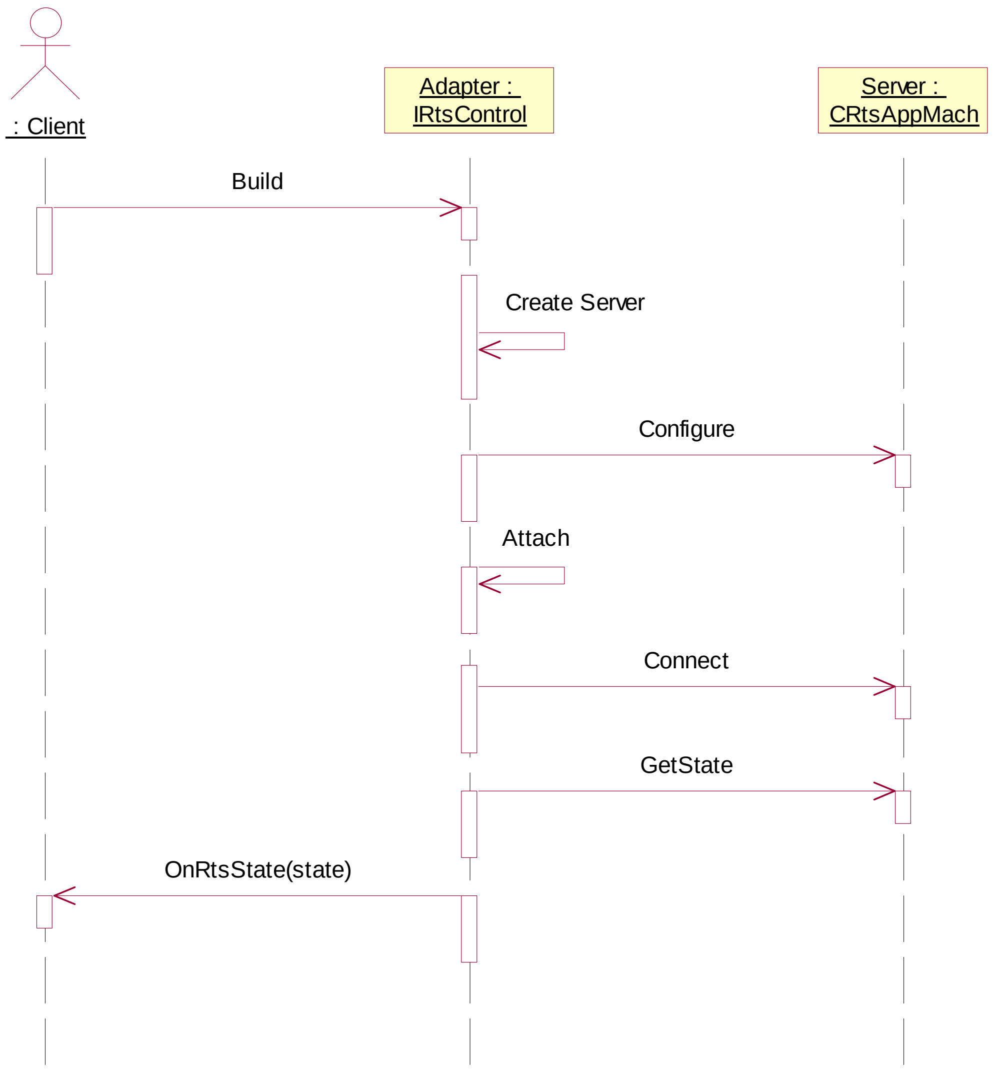


Figure 3 Build Sequence Diagram

Build or *Attach* methods must be called before calling any other control method. The *Settings* property is an exception; it can be used to specify the desired configuration before calling the *Build* method.
The *Build* method provide clients a way to start up and *Attach* to the RTS by providing its own (or default) settings.

3.2.3 Load Method

Parameters

Name	Type	Access	Description
strTps	BSTR	[in]	Full path to the PAWS project file to open

The *Load* method loads the specified TPS project file. The RTS executive loads the required binary files and executes the *INITIALIZE* macro. The UNC (Uniform Naming Convention) can be used to specify a TPS from a file system shared by remote computers. The RTS executive can open and run a TPS in a read-only environment such as a CD-ROM or a file system with read-only permissions, the temporary files will be created in the system temporary directory.

As shown, the *Load* method is asynchronous; it starts the loading process and returns immediately. The adapter object fires a state change event when the loading process is complete. The transition to the "READY" state signals that the loading is complete and the RTS is ready to start the TPS execution. A context change event is also fired when loading is complete, the context transitions from "INITIALIZE" to "TPS".

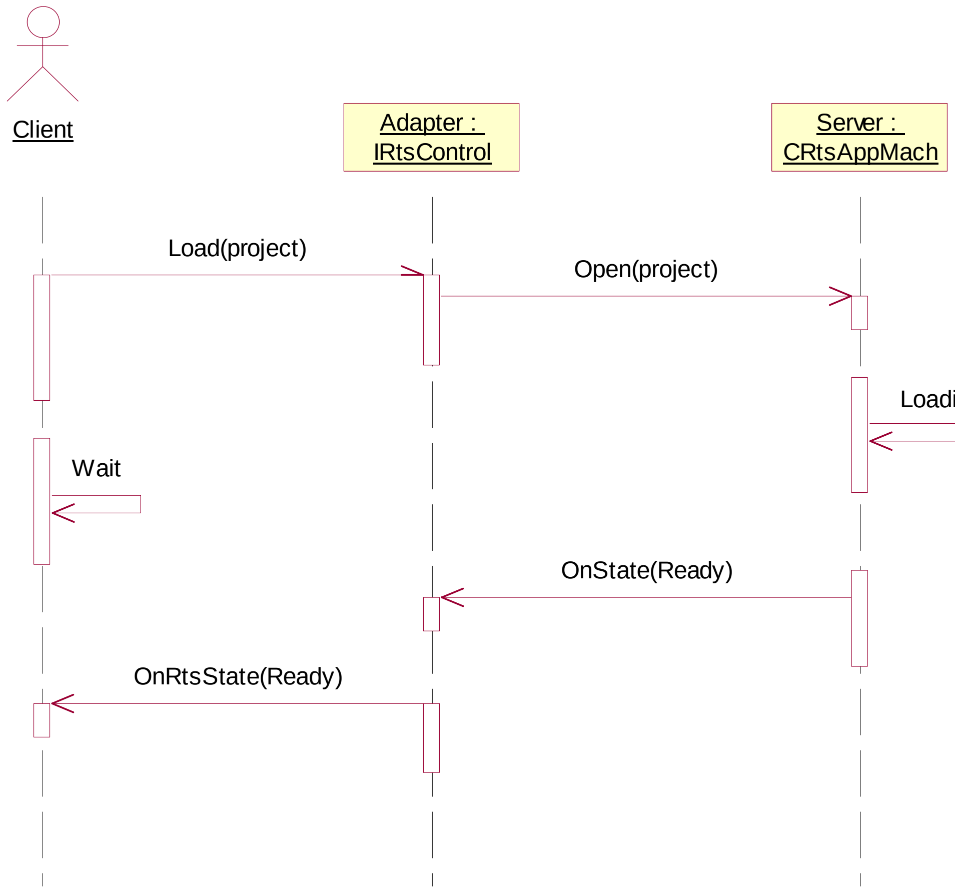


Figure 4 Load Sequence Diagram

The *Load* method can be called only from the "UNLOADED" state after the adapter was attached to the RTS Server by first calling either *Attach* or *Build*. Not doing so will result in an error.

3.2.4 Unload Method

The *Unload* method starts the TPS unloading sequence. During the unloading sequence the RTS executive runs the *TERMINATE* macro and unloads the binary files in use. The unload sequence frees all resources, such as files, GPIB and/or VXI buses, and P&P and/or IVI session handles. *Unload* can be used to shutdown the RTS gracefully.

As shown, the *Unload* method is asynchronous; it starts the unloading process and returns immediately. The adapter object fires a state change event when the unloading process is complete. The transition to the "UNLOADED" state signals that the unloading is complete. A context change event is also fired when loading is complete, the context transitions from "TPS" to "TERMINATE" and then to "UNLOADED".

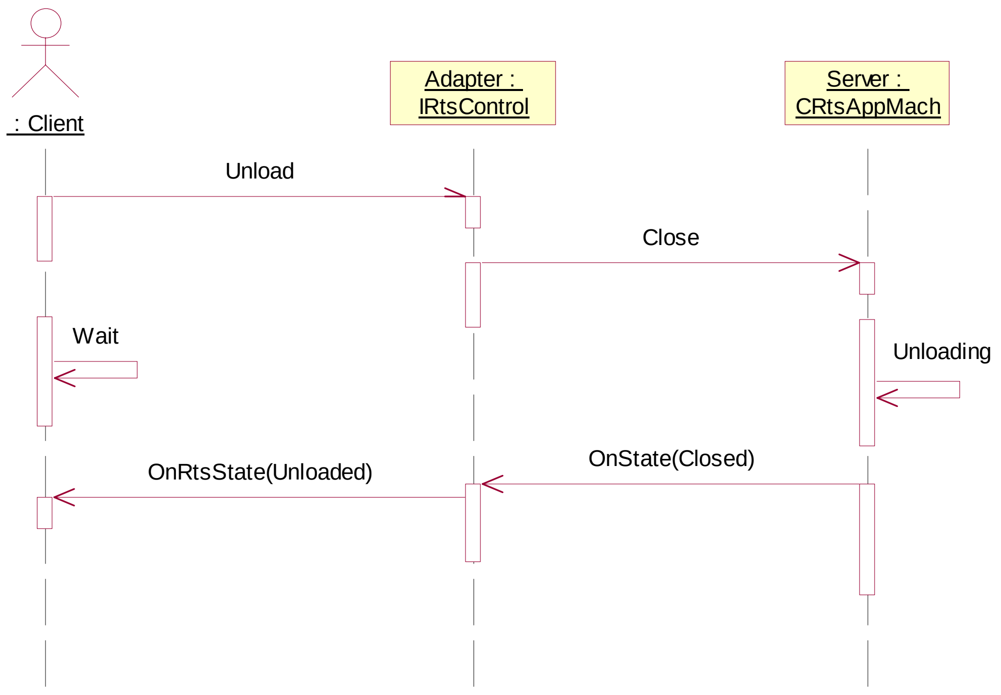


Figure 5 Unload Sequence Diagram

The *Unload* method can be called from the "READY", "FINISH", "HALTED" or "RUNNING" states. The adapter must be attached to the RTS Server and with a TPS loaded. Errors will be reported if the RTS Server is not attached or no TPS is loaded.

3.2.5 Run Method

The *Run* method starts or continues the execution of a TPS program. As shown in ***Figure 6*** the *Run* method is asynchronous, it starts the running process and returns immediately. The adapter object fires a state change event when the execution is complete or interrupted. At the end of the execution the RTS transitions to the "FINISH", "HALTED" or "UNLOADED" state.

The *Run* method can be called from the "READY", "FINISH" or "HALTED" states. The adapter must be attached to the RTS Server with a TPS loaded and not be in the "RUNNING" state already. Errors will be reported otherwise.

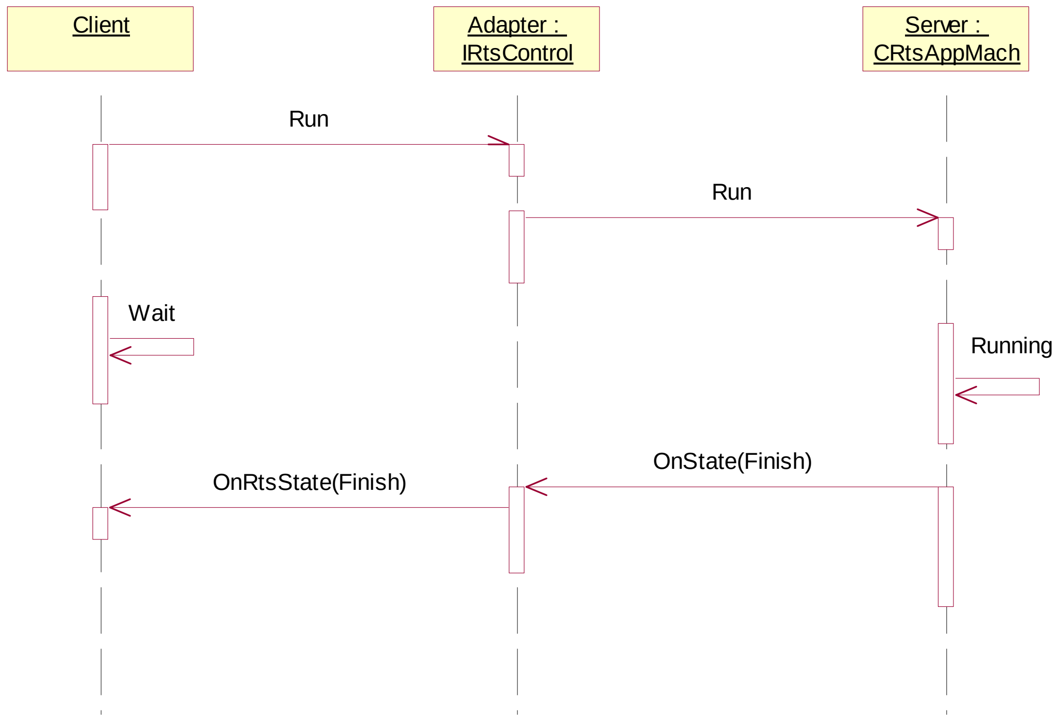


Figure 6 Run Sequence Diagram

3.2.6 Halt Method

The *Halt* method stops the execution of a running TPS program. As shown, the *Halt* method is asynchronous, it posts a halt requests to the RTS Server and returns immediately. The RTS executive halts at the next atomic instruction. The adapter object fires a state change event when the RTS transitions to the "HALTED" state.

The *Halt* method can be called only from the "RUNNING" state. The adapter must be attached to the RTS Server with a TPS loaded and running. Errors will be reported otherwise.

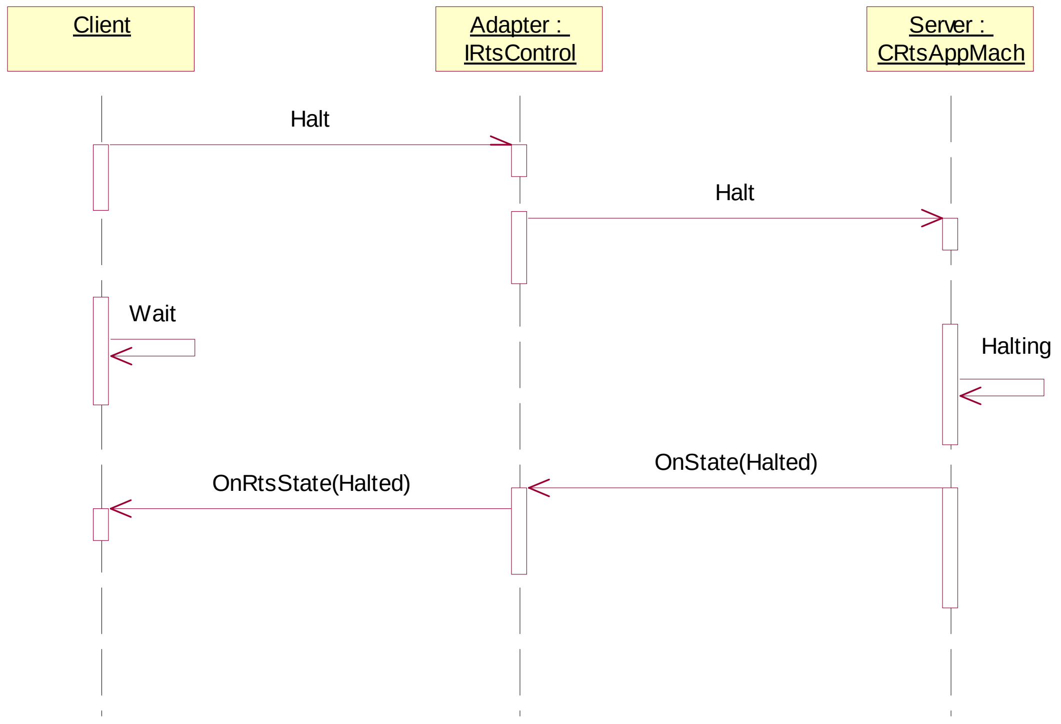


Figure 7 Halt Sequence Diagram

3.2.7 Reset Method

The *Reset* method starts the ATE and TPS reset sequence. During the reset sequence the RTS executive runs the *RESETALL* macro and resets all busses and drivers. In addition, the TPS reset sequence sets the execution pointer to the beginning of the TPS program, no ATLAS variables are changed. *Reset* can be used to end the TPS execution and reset the ATE and TPS in emergencies or for a clean restart. The reset sequence ends with a transition to the "FINISH" state.

The *Reset* method is asynchronous; it starts the reset sequence and returns immediately. The adapter object fires a state change event when the reset sequence is complete. The transition to the "FINISH" state signals that the reset is complete.

The *Reset* method can be called from the "READY", "FINISH", "RUNNING" or "HALTED" states. The adapter must be attached to the RTS Server and with a TPS loaded. Errors will be reported otherwise.

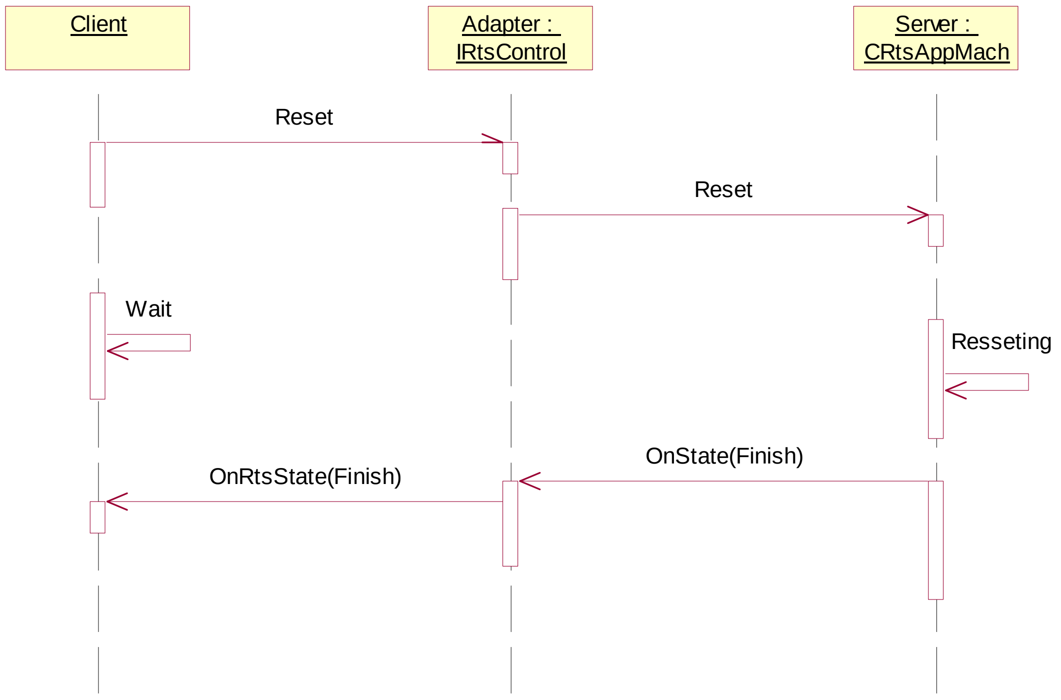


Figure 8 Reset Sequence Diagram

3.2.8 ManualIntervention Method

The *ManualIntervention* method sets the “MANUALINTERVENTION” ATLAS variable to true. This action is used for ATLAS statements such as `MONITOR` or events based on manual intervention when the RTS executive waits for the “MANUALINTERVENTION” ATLAS variable to be set to true.

The *ManualIntervention* method can be called from the “READY”, “FINISH”, “RUNNING” or “HALTED” states. The adapter must be attached to the RTS Server, with a TPS loaded. Errors will be reported otherwise.

3.2.9 SkipDelay Method

The *SkipDelay* method interrupts the current delay and continues the execution.

The *SkipDelay* method can be called from the “RUNNING” state. The adapter must be attached to the RTS Server and with a TPS loaded. Errors will be reported otherwise. If the RTS is not currently executing an ATLAS Delay statement and *SkipDelay* is called no errors are reported.

3.2.10 CaptureInput Method

Parameters

Name	Type	Access	Description
varRsrc	VARIANT	[in, optional]	The desired input resource

The *CaptureInput* method replaces the input resource used by RTS Server with the resource specified by the *varRsrc* parameter.

The possible types and values of the optional *varRsrc* VARIANT parameter are:

1. 1. VT_EMPTY or VT_NULL – in this case a default INPUT resource is created and used in the client process. When the RTS Server is running on a remote host this will display input dialog on the client machine; by default the input dialog is displayed on the remote machine, where the RTS Server is running. The input dialog appearance and behavior is identical, it only runs in the client machine and process.
2. 2. VT_ERROR – this case occurs from scripting languages when the optional parameter is omitted. The behavior is the same as for case 1.
3. 3. VT_UNKNOWN or VT_DISPATCH – the provided interface pointer must point to a custom I/O resource object. Custom I/O resource components must implement at least the *IIOResource* and *ITextResource* interfaces as defined in the I/O Subsystem documents. The *IStdIO* interface can also be implemented to gain access to data such as the prompt or the default value. For details see the I/O Subsystem documentation.
4. 4. VT_BSTR – the parameter is interpreted as the ProgID of the I/O resource to use, for example "CustomServer.InputResource". The specified resource will be created on the client machine in the client process. The I/O resource class GUID can also be specified enclosed in curly brackets, for example "{3F6B2A10-F0DA-11D2-BBB0-00C0268914D3}". Custom I/O resources must meet the same requirements as for case 3.
5. 5. All other types will return an Error.

The *CaptureInput* method must be called from the "UNLOADED" state. The adapter must be first attached to the RTS Server. The provided input resource is used during the next TPS loading. Call *CaptureInput* before loading the TPS. If *CaptureInput* is called with a TPS already loaded it will take effect only for the next TPS. Errors will be returned when the custom I/O resource does not satisfy required interfaces. When the input is already captured an error is reported, to recapture the input call *ReleaseInput* first.

3.2.11 ReleaseInput Method

The *ReleaseInput* method restores the input resource used by RTS Server. It 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 this method from the "UNLOADED" state. If *ReleaseInput* is called with a TPS loaded it will take effect only for the next TPS. Detaching from the server or destroying the adapter object will automatically release a captured input. Calling *ReleaseInput* without successfully capturing the input first will generate an error.

3.2.12 Settings Property

Type	**Access**	**Description**
`IRtsSettings*`	`Read-Only`	Interface pointer to the RTS settings object.

The *Settings* property provides access to the settings and options of the RTS Server. The settings object is available before attaching the RTS Server. The content of the settings object is used to configure the RTS Server when the *Build* method is used for attachment. While the RTS Server is attached, all options and settings exposed by the *IRtsSettings* interface reflect and access directly the options and settings of the RTS Server. The property is read-only; the interface pointer cannot be changed. The content of the settings object however is read-write it can be changed without restrictions.

3.2.13 Dispatch Identifiers

```
const DISPID DISPID_BUILD               = 51;
const DISPID DISPID_LOAD                = 52;
const DISPID DISPID_UNLOAD              = 53;
const DISPID DISPID_RUN                 = 54;
const DISPID DISPID_HALT                = 55;
const DISPID DISPID_RESET               = 56;
const DISPID DISPID_MANINTV             = 57;
const DISPID DISPID_SKIPDLY             = 58;
const DISPID DISPID_CAPINPUT            = 59;
const DISPID DISPID_RELINPUT            = 60;
const DISPID DISPID_SETTINGS            = 61;
```

3.3 IRtsAxMonitor Interface

The *IRtsAxMonitor* interface derives from *IRtsMonitor*. In addition to the methods and properties exposed by the *IRtsMonitor* interface, the *IRtsAxMonitor* interface exposes properties to control the visual appearance and behavior of the ActiveX control.

3.3.1 IDL Description

```
[
	object,
	uuid(3F6B2911-F0DA-11D2-BBB0-00C0268914D3),
	dual,
	helpstring("IRtsAxMonitor Interface"),
	pointer_default(unique)
]
interface IRtsAxMonitor : IRtsMonitor
{
	[propget, id(DISPID_CONFIGURABLE), helpstring("property Configurable")]
	HRESULT Configurable([out, retval] VARIANT_BOOL* pVal);
	[propput, id(DISPID_CONFIGURABLE), helpstring("property Configurable")]
	HRESULT Configurable([in] VARIANT_BOOL newVal);
	[propput, id(DISPID_DSPBACKCOLOR),
	          helpstring("property Display BackColor")]
	HRESULT DspBackColor([in]OLE_COLOR clr);
	[propget, id(DISPID_DSPBACKCOLOR),
	         helpstring("property Display BackColor")]
	HRESULT DspBackColor([out,retval]OLE_COLOR* pclr);
	[propput, id(DISPID_DSPFORECOLOR),
	         helpstring("property Display ForeColor")]
	HRESULT DspForeColor([in]OLE_COLOR clr);
	[propget, id(DISPID_DSPFORECOLOR),
	         helpstring("property Display ForeColor")]
	HRESULT DspForeColor([out,retval]OLE_COLOR* pclr);
	[propget, id(DISPID_DSPERRORCOLOR),
	         helpstring("property Display ErrorColor")]
	HRESULT DspErrorColor([out, retval] OLE_COLOR* pVal);
	[propput, id(DISPID_DSPERRORCOLOR),
	         helpstring("property Display ErrorColor")]
	HRESULT DspErrorColor([in] OLE_COLOR newVal);
	[propget, id(DISPID_DSPWARNCOLOR),
	         helpstring("property Display WarningColor")]
	HRESULT DspWarnColor([out, retval] OLE_COLOR* pVal);
	[propput, id(DISPID_DSPWARNCOLOR),
	         helpstring("property Display WarningColor")]
	HRESULT DspWarnColor([in] OLE_COLOR newVal);
	[propget, id(DISPID_DSPINFOCOLOR),
	         helpstring("property Display InfoColor")]
	HRESULT DspInfoColor([out, retval] OLE_COLOR* pVal);
	[propput, id(DISPID_DSPINFOCOLOR),
	         helpstring("property Display InfoColor")]
	HRESULT DspInfoColor([in] OLE_COLOR newVal);
	[propget, id(DISPID_DSPLINES), helpstring("property Display Lines")]
	HRESULT DspLines([out, retval] long* pVal);
	[propput, id(DISPID_DSPLINES), helpstring("property Display Lines")]
	HRESULT DspLines([in] long newVal);
	[propputref, id(DISPID_DSPFONT), helpstring("property Display Font")]
	HRESULT DspFont([in]IFontDisp* pFont);
	[propput, id(DISPID_DSPFONT), helpstring("property Display Font")]
	HRESULT DspFont([in]IFontDisp* pFont);
	[propget, id(DISPID_DSPFONT), helpstring("property Display Font")]
	HRESULT DspFont([out, retval]IFontDisp** ppFont);
};
```

3.3.2 Configurable Property

Type	Access	Description
VARIANT_BOOL	Read-Write	Controls if run-time configuration is enabled

The *Configurable* property controls if the run-time configuration is enabled or not. When set to true the run-time configuration bitmap is visible, when set to false the run-time configuration bitmap is hidden.

3.3.3 DspBackColor Property

Type	Access	Description
OLE_COLOR	Read-Write	Controls the background color of the output window

3.3.4 DspForeColor Property

Type	Access	Description
OLE_COLOR	Read-Write	Controls the text foreground color for the output window

3.3.5 DspErrorColor Property

Type	Access	Description
OLE_COLOR	Read-Write	Controls the foreground color for error messages

3.3.6 DspWarnColor Property

Type	Access	Description
OLE_COLOR	Read-Write	Controls the foreground color for warning messages

3.3.7 DspInfoColor Property

Type	Access	Description
OLE_COLOR	Read-Write	Controls the foreground color for information messages

3.3.8 DspLines Property

Type	Access	Description
Long	Read-Write	Controls the maximum number of lines stored for the output window

3.3.9 DspFont Property

Type	Access	Description
IFontDisp*	Read-Write	Controls the font used by the output window

3.3.10 Dispatch Identifiers

```
const DISPID DISPID_CONFIGURABLE        = 71;
const DISPID DISPID_DSPBACKCOLOR        = 72;
const DISPID DISPID_DSPFORECOLOR        = 73;
const DISPID DISPID_DSPERRORCOLOR       = 74;
const DISPID DISPID_DSPWARNCOLOR        = 75;
const DISPID DISPID_DSPINFOCOLOR        = 76;
const DISPID DISPID_DSPLINES            = 77;
const DISPID DISPID_DSPFONT             = 78;
```

3.4 RTSMonitor CoClass

3.4.1 IDL Description

```
[
        uuid(3F6B2910-F0DA-11D2-BBB0-00C0268914D3),
        helpstring("RtsMonitor Class")
]
coclass RtsMonitor
{
        [default] interface IRtsAxMonitor;
        [default, source] dispinterface _IRtsBaseEvents;
};
```

3.4.2 UML Design

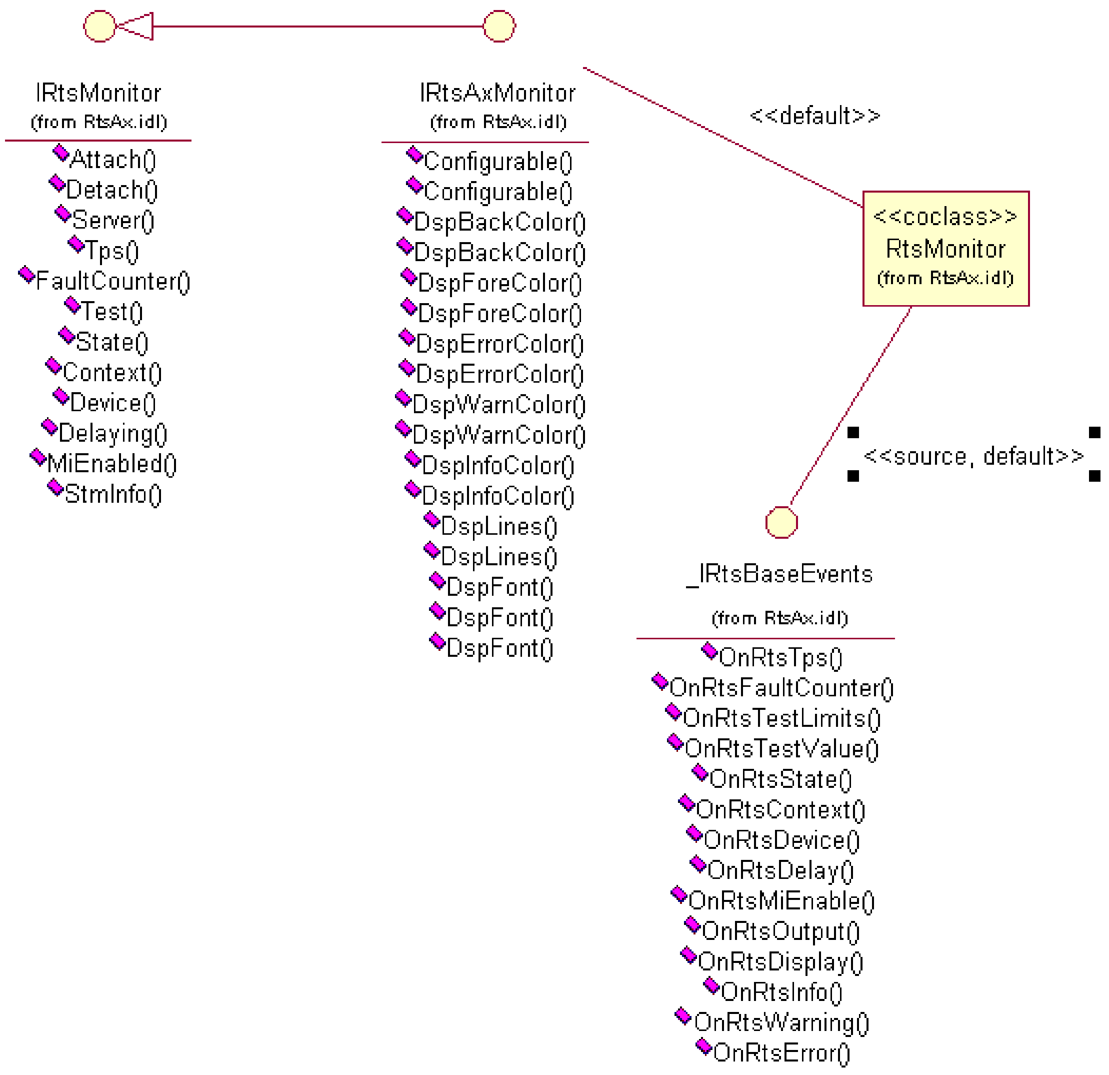
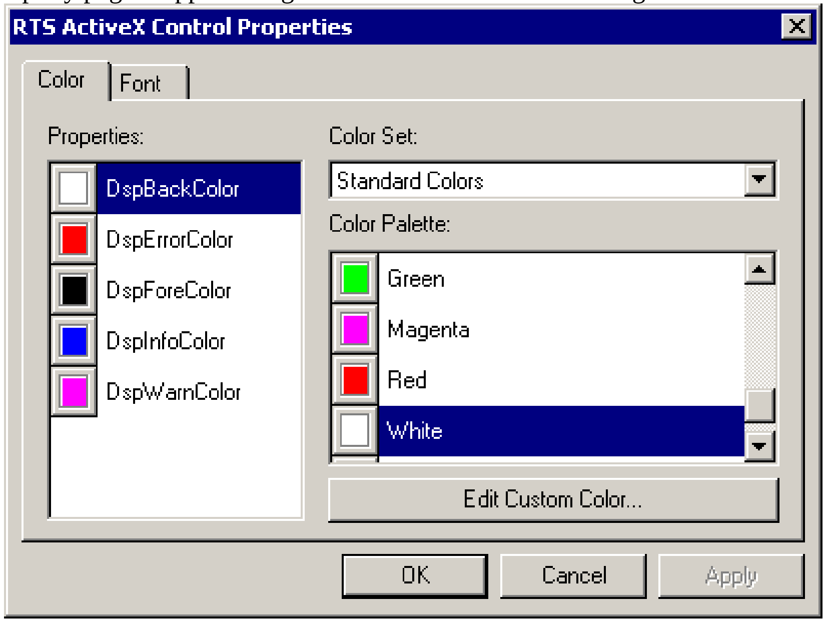


Figure Error! Bookmark not defined. **RTSMonitor class design**

3.4.3 Property Pages

The provided property pages support design time as well as run-time configuration.


RTS ActiveX Control Properties

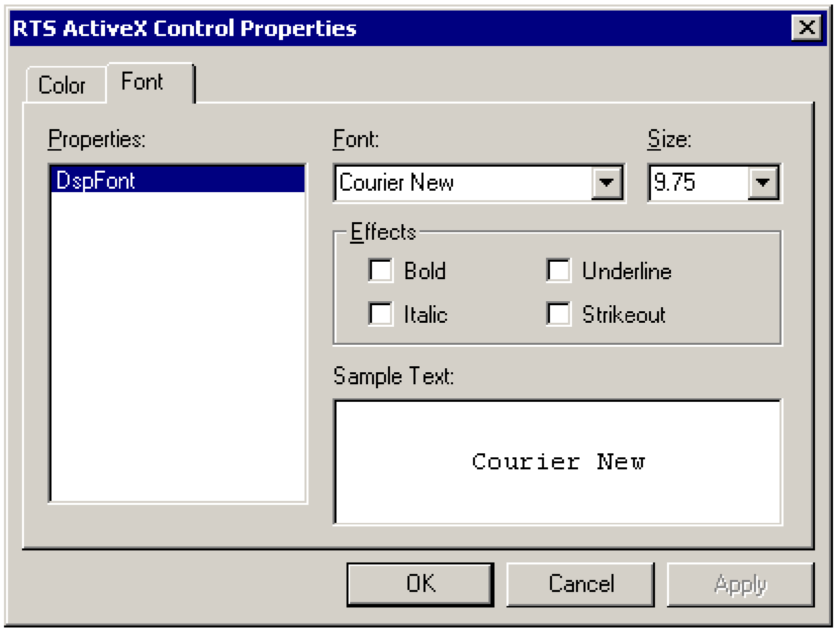
Color | Font

Properties:
- ⬜ DspBackColor
- 🟥 DspErrorColor
- ⬛ DspForeColor
- 🟦 DspInfoColor
- 🟪 DspWarnColor

Color Set: Standard Colors

Color Palette:
- 🟩 Green
- 🟪 Magenta
- 🟥 Red
- ⬜ White

Figure 10 Color Property Page


RTS ActiveX Control Properties

Color | Font

Properties: DspFont

Font: Courier New

Size: 9.75

Effects:
- ☐ Bold
- ☐ Underline
- ☐ Italic
- ☐ Strikeout

Sample Text: Courier New

Figure 11 Font Property Page

3.5 IRtsAxControl Interface

The *IRtsAxControl* interface derives from *IRtsControl* interface. In addition to methods and properties exposed by the *IRtsControl* interface, the *IRtsAxControl* interface exposes a property to control whether or not the control object is configurable.

3.5.1 IDL Description

```
[
	object,
	uuid(3F6B2921-F0DA-11D2-BBB0-00C0268914D3),
	dual,
	helpstring("IRtsAxControl Interface"),
	pointer_default(unique)
]
interface IRtsAxControl : IRtsControl
{
	[propget, id(DISPID_CONFIGURABLE), helpstring("property Configurable")]
	HRESULT Configurable([out, retval] VARIANT_BOOL* pVal);
	[propput, id(DISPID_CONFIGURABLE), helpstring("property Configurable")]
	HRESULT Configurable([in] VARIANT_BOOL newVal);
};
```

3.5.2 Configurable Property

Type	**Access**	**Description**
VARIANT_BOOL	Read-Write	Controls if run-time configuration is enabled

The *Configurable* property controls if the run-time configuration is enabled or not. When set to true the run-time configuration bitmap is visible, when set to false the run-time configuration bitmap is hidden.

3.6 RtsControl CoClass

3.6.1 IDL Description

```
[
	uuid(3F6B2920-F0DA-11D2-BBB0-00C0268914D3),
	helpstring("RtsControl Class")
]
coclass RtsControl
{
	[default] interface IRtsAxControl;
	[default, source] dispinterface _IRtsBaseEvents;
};
```

3.6.2 UML Design

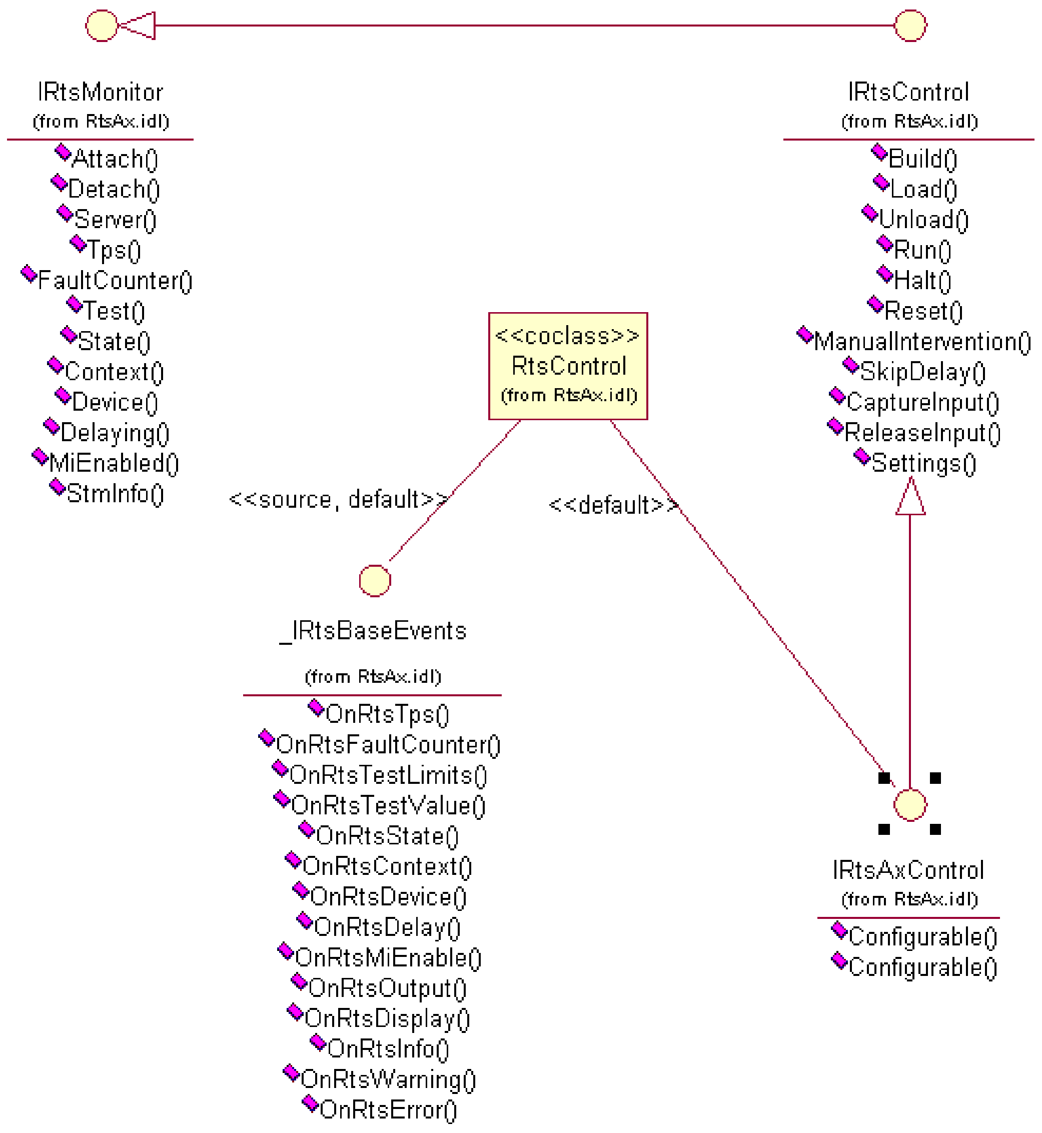
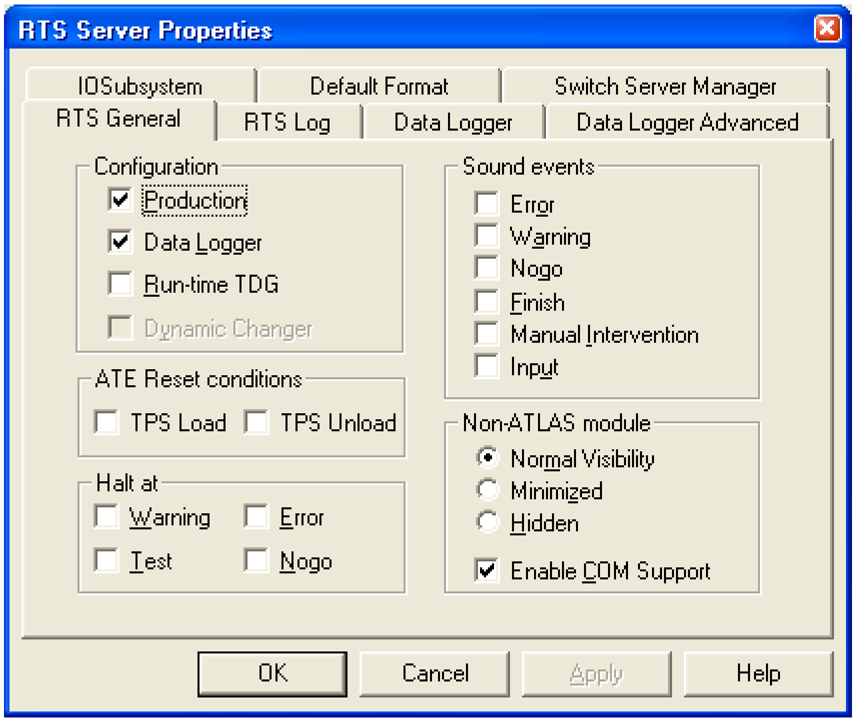


Figure Error! Bookmark not defined. **RTSControl class design**

3.6.3 PropertyPages



Configuration
- [✓] Production
- [✓] Data Logger
- [ ] Run-time TDG
- [ ] Dynamic Changer

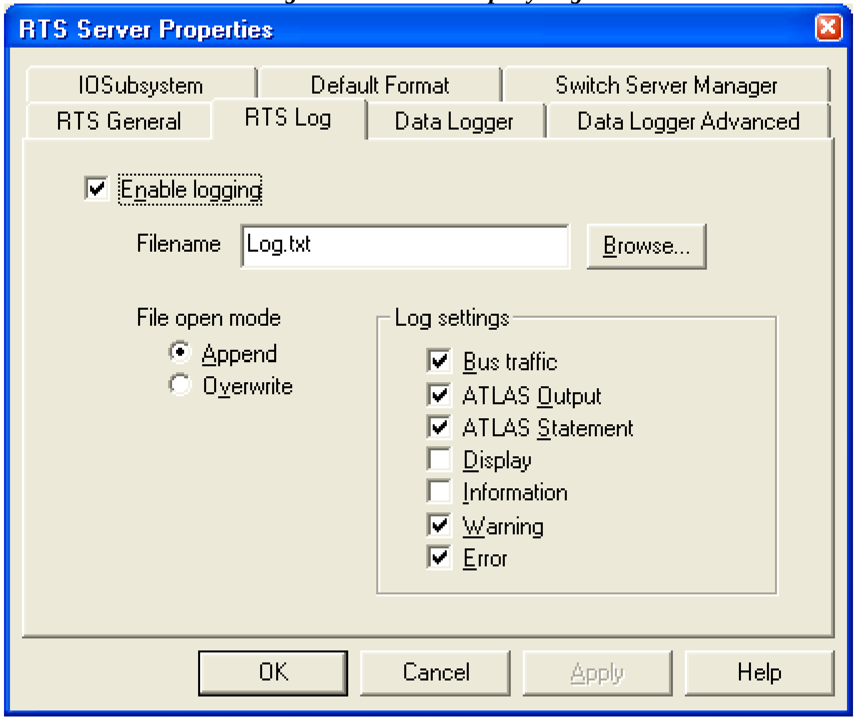
ATE Reset conditions
- [ ] TPS Load
- [ ] TPS Unload

Halt at
- [ ] Warning
- [ ] Error
- [ ] Test
- [ ] Nogo

Sound events
- [ ] Error
- [ ] Warning
- [ ] Nogo
- [ ] Finish
- [ ] Manual Intervention
- [ ] Input

Non-ATLAS module
- [✓ Normal Visibility]
- Minimized
- Hidden
- [✓] Enable COM Support

Figure 13 General Property Page



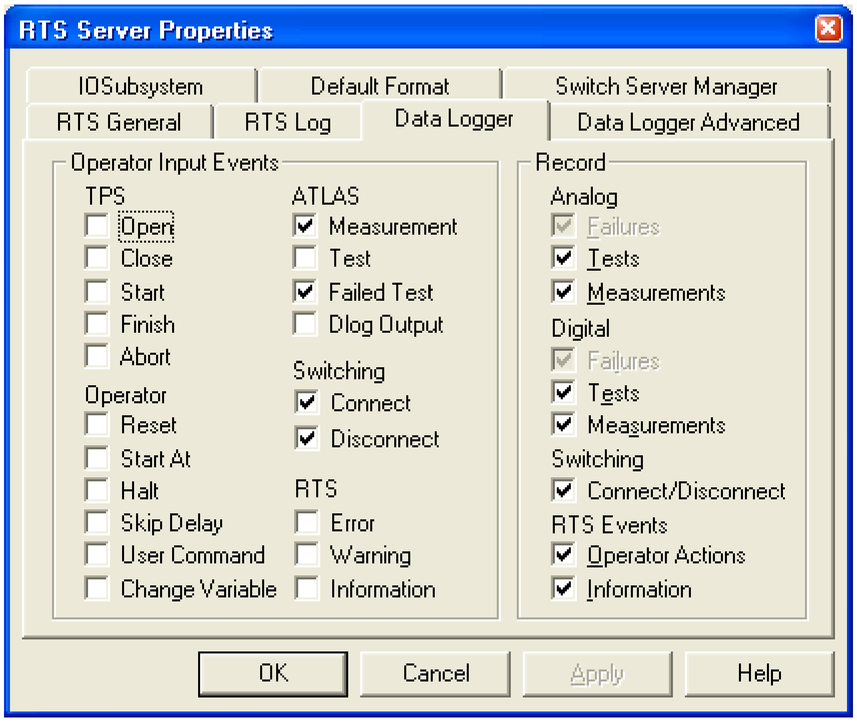
[✓] Enable logging

Filename: Log.txt

File open mode
- [✓ Append]
- Overwrite

Log settings
- [✓] Bus traffic
- [✓] ATLAS Output
- [✓] ATLAS Statement
- [ ] Display
- [ ] Information
- [✓] Warning
- [✓] Error

Figure 14 RTS Log Property Page


RTS Server Properties

IOSubsystem | Default Format | Switch Server Manager | RTS General | RTS Log | Data Logger | Data Logger Advanced

Operator Input Events

TPS
- [ ] Open
- [ ] Close
- [ ] Start
- [ ] Finish
- [ ] Abort

Operator
- [ ] Reset
- [ ] Start At
- [ ] Halt
- [ ] Skip Delay
- [ ] User Command
- [ ] Change Variable

ATLAS
- [✓] Measurement
- [ ] Test
- [✓] Failed Test
- [ ] Dlog Output

Switching
- [✓] Connect
- [✓] Disconnect

RTS
- [ ] Error
- [ ] Warning
- [ ] Information

Record

Analog
- [✓] Failures
- [✓] Tests
- [✓] Measurements

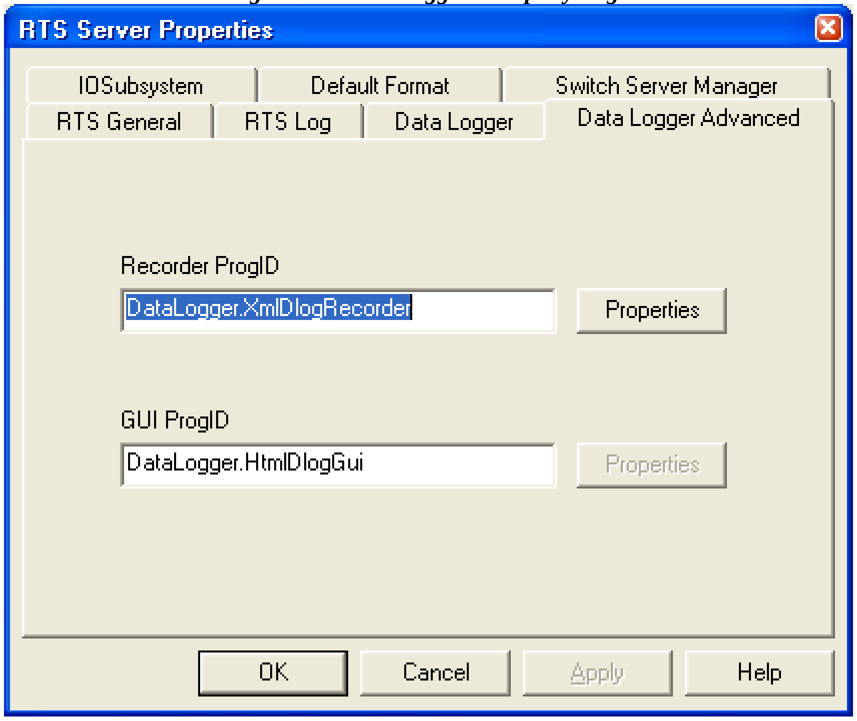
Digital
- [✓] Failures
- [✓] Tests
- [✓] Measurements

Switching
- [✓] Connect/Disconnect

RTS Events
- [✓] Operator Actions
- [✓] Information

OK | Cancel | Apply | Help

Figure 15 "Data Logger" Property Page


RTS Server Properties

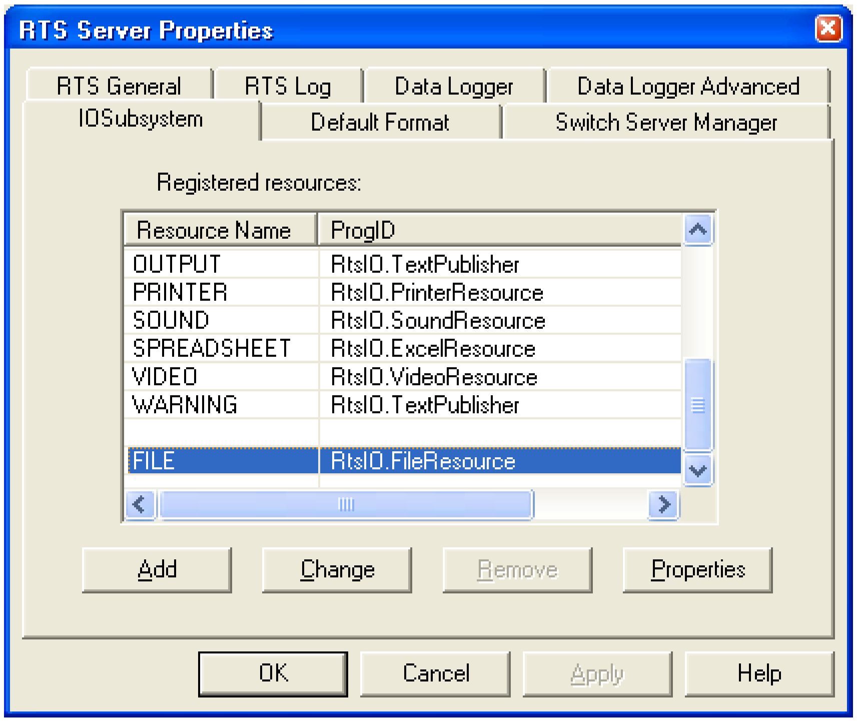
IOSubsystem | Default Format | Switch Server Manager | RTS General | RTS Log | Data Logger | Data Logger Advanced

Recorder ProgID: DataLogger.XmlDlogRecorder — Properties

GUI ProgID: DataLogger.HtmlDlogGui — Properties

OK | Cancel | Apply | Help

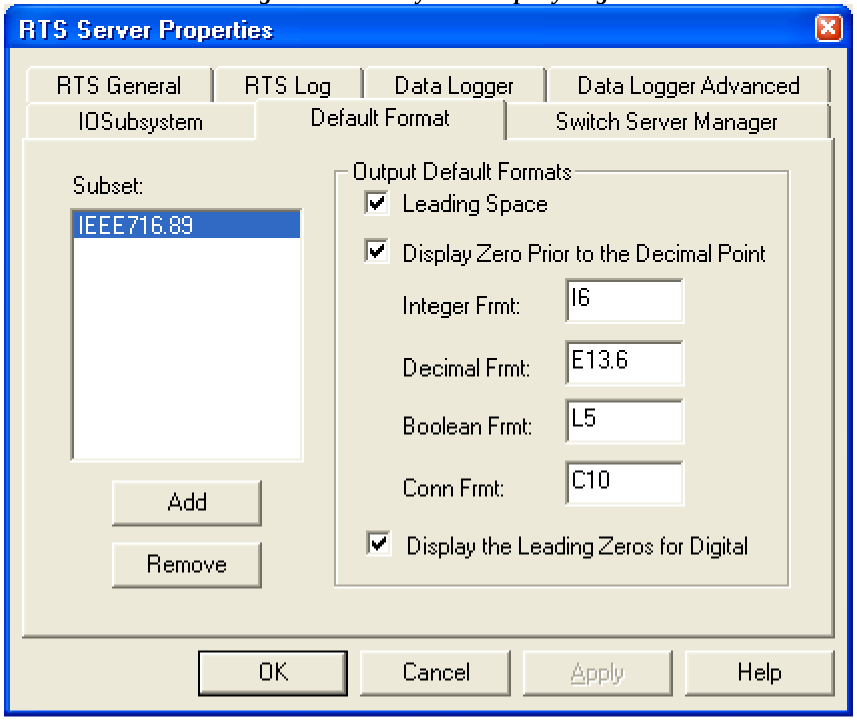
Figure 16 Data Logger Advanced Property Page



Registered resources:

Resource Name	ProgID
OUTPUT	RtsIO.TextPublisher
PRINTER	RtsIO.PrinterResource
SOUND	RtsIO.SoundResource
SPREADSHEET	RtsIO.ExcelResource
VIDEO	RtsIO.VideoResource
WARNING	RtsIO.TextPublisher
FILE	RtsIO.FileResource

Figure 17 IOSubsystem Property Page



Subset:

IEEE716.89

Output Default Formats

[✓] Leading Space

[✓] Display Zero Prior to the Decimal Point

Integer Frmt: I6

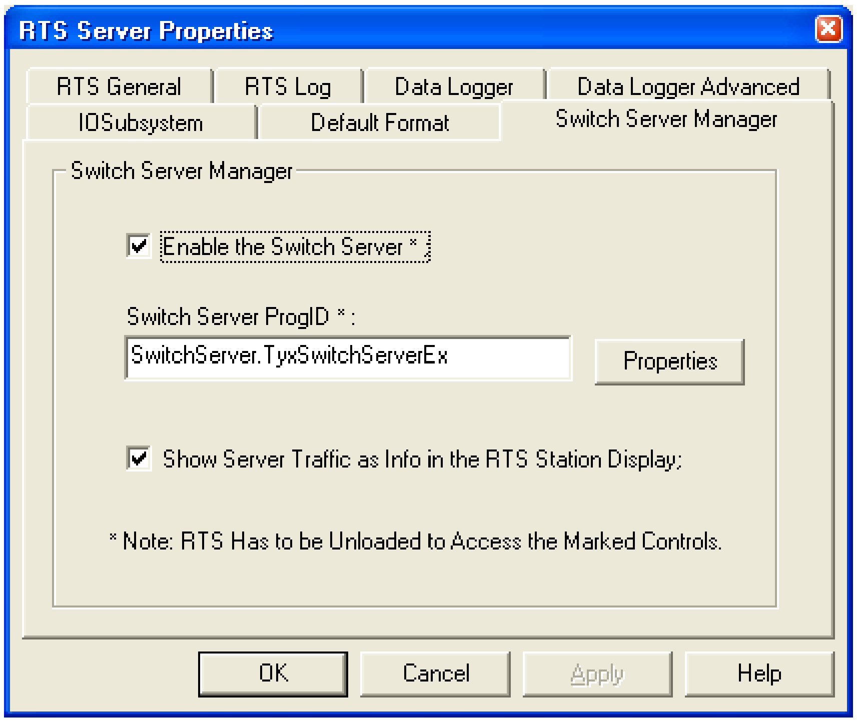
Decimal Frmt: E13.6

Boolean Frmt: L5

Conn Frmt: C10

[✓] Display the Leading Zeros for Digital

Figure 18 DefaultFormat Property Page



Switch Server Manager

[✓] Enable the Switch Server * :

Switch Server ProgID * : SwitchServer.TyxSwitchServerEx

[✓] Show Server Traffic as Info in the RTS Station Display:

* Note: RTS Has to be Unloaded to Access the Marked Controls.

Figure 19 SwitchServerManager Property Page

3.7 IRtsAxCombo Interface

The *IRtsAxCombo* interface derives from *IRtsControl* interface. In addition to methods and properties exposed by the *IRtsControl* interface, the *IRtsAxCombo* interface exposes properties to control the visual appearance of the ActiveX control.

3.7.1 IDL Description

```
[
	object,
	uuid(3F6B2931-F0DA-11D2-BBB0-00C0268914D3),
	dual,
	helpstring("IRtsAxCombo Interface"),
	pointer_default(unique)
]
interface IRtsAxCombo : IRtsControl
{
	[propget, id(1), helpstring("property RtsControl")]
	HRESULT RtsControl([out, retval] IRtsAxControl** pVal);
	[propget, id(2), helpstring("property RtsMonitor")]
	HRESULT RtsMonitor([out, retval] IRtsAxMonitor** pVal);
	[propget, id(DISPID_CONFIGURABLE), helpstring("property Configurable")]
	HRESULT Configurable([out, retval] VARIANT_BOOL* pVal);
	[propput, id(DISPID_CONFIGURABLE), helpstring("property Configurable")]
	HRESULT Configurable([in] VARIANT_BOOL newVal);
	[propput, id(DISPID_DSPBACKCOLOR),
		  helpstring("property Display BackColor")]
	HRESULT DspBackColor([in]OLE_COLOR clr);
	[propget, id(DISPID_DSPBACKCOLOR),
		  helpstring("property Display BackColor")]
	HRESULT DspBackColor([out,retval]OLE_COLOR* pclr);
	[propput, id(DISPID_DSPFORECOLOR),
                  helpstring("property Display ForeColor")]
        HRESULT DspForeColor([in]OLE_COLOR clr);
        [propget, id(DISPID_DSPFORECOLOR),
                  helpstring("property Display BackColor")]
        HRESULT DspForeColor([out,retval]OLE_COLOR* pclr);
        [propget, id(DISPID_DSPERRORCOLOR),
                  helpstring("property Display ErrorColor")]
        HRESULT DspErrorColor([out, retval] OLE_COLOR* pVal);
        [propput, id(DISPID_DSPERRORCOLOR),
                  helpstring("property Display ErrorColor")]
        HRESULT DspErrorColor([in] OLE_COLOR newVal);
        [propget, id(DISPID_DSPWARNCOLOR),
                  helpstring("property Display WarningColor")]
        HRESULT DspWarnColor([out, retval] OLE_COLOR* pVal);
        [propput, id(DISPID_DSPWARNCOLOR),
                  helpstring("property Display WarningColor")]
        HRESULT DspWarnColor([in] OLE_COLOR newVal);
        [propget, id(DISPID_DSPINFOCOLOR),
                  helpstring("property Display InfoColor")]
        HRESULT DspInfoColor([out, retval] OLE_COLOR* pVal);
        [propput, id(DISPID_DSPINFOCOLOR),
                  helpstring("property Display InfoColor")]
        HRESULT DspInfoColor([in] OLE_COLOR newVal);
        [propget, id(DISPID_DSPLINES), helpstring("property Display Lines")]
        HRESULT DspLines([out, retval] long* pVal);
        [propput, id(DISPID_DSPLINES), helpstring("property Display Lines")]
        HRESULT DspLines([in] long newVal);
        [propputref, id(DISPID_DSPFONT), helpstring("property Display Font")]
        HRESULT DspFont([in]IFontDisp* pFont);
        [propput, id(DISPID_DSPFONT), helpstring("property Display Font")]
        HRESULT DspFont([in]IFontDisp* pFont);
        [propget, id(DISPID_DSPFONT), helpstring("property Display Font")]
        HRESULT DspFont([out, retval]IFontDisp** ppFont);
};
```

3.7.2 RtsControl Property

Type	**Access**	**Description**
`IRtsAxControl*`	`Read-Only`	Interface to the inner RTS Control object

3.7.3 RtsMonitor Property

Type	**Access**	**Description**
`IRtsAxMonitor*`	`Read-Only`	Interface to the inner RTS Monitor object

3.7.4 Configurable Property

Type	**Access**	**Description**
`VARIANT_BOOL`	`Read-Write`	Controls if run-time configuration is enabled

The *Configurable* property controls if the run-time configuration is enabled or not. When set to true the run-time configuration bitmap is visible, when set to false the run-time configuration bitmap is hidden.

3.7.5 DspBackColor Property

Type	**Access**	**Description**
`OLE_COLOR`	`Read-Write`	Controls the background color of the output window

3.7.6 DspForeColor Property

Type	Access	Description
OLE_COLOR	Read-Write	Controls the text foreground color for the output window

3.7.7 DspErrorColor Property

Type	Access	Description
OLE_COLOR	Read-Write	Controls the foreground color for error messages

3.7.8 DspWarnColor Property

Type	Access	Description
OLE_COLOR	Read-Write	Controls the foreground color for warning messages

3.7.9 DspInfoColor Property

Type	Access	Description
OLE_COLOR	Read-Write	Controls the foreground color for information messages

3.7.10 DspLines Property

Type	Access	Description
Long	Read-Write	Controls the maximum number of lines stored for the output window

3.7.11 DspFont Property

Type	Access	Description
IFontDisp*	Read-Write	Controls the font used by the output window

3.7.12 Dispatch Identifiers

```
	const DISPID DISPID_CONFIGURABLE	= 71;
	const DISPID DISPID_DSPBACKCOLOR	= 72;
	const DISPID DISPID_DSPFORECOLOR	= 73;
	const DISPID DISPID_DSPERRORCOLOR	= 74;
	const DISPID DISPID_DSPWARNCOLOR	= 75;
	const DISPID DISPID_DSPINFOCOLOR	= 76;
	const DISPID DISPID_DSPLINES		= 77;
	const DISPID DISPID_DSPFONT		= 78;
```

3.8 RtsCombo CoClass

3.8.1 IDL Description

```
[
	uuid(3F6B2930-F0DA-11D2-BBB0-00C0268914D3),
	helpstring("RtsCombo Class")
]
coclass RtsCombo
{
	[default] interface IRtsAxCombo;
	[default, source] dispinterface _IRtsBaseEvents;
};
```

3.8.2 UML Design

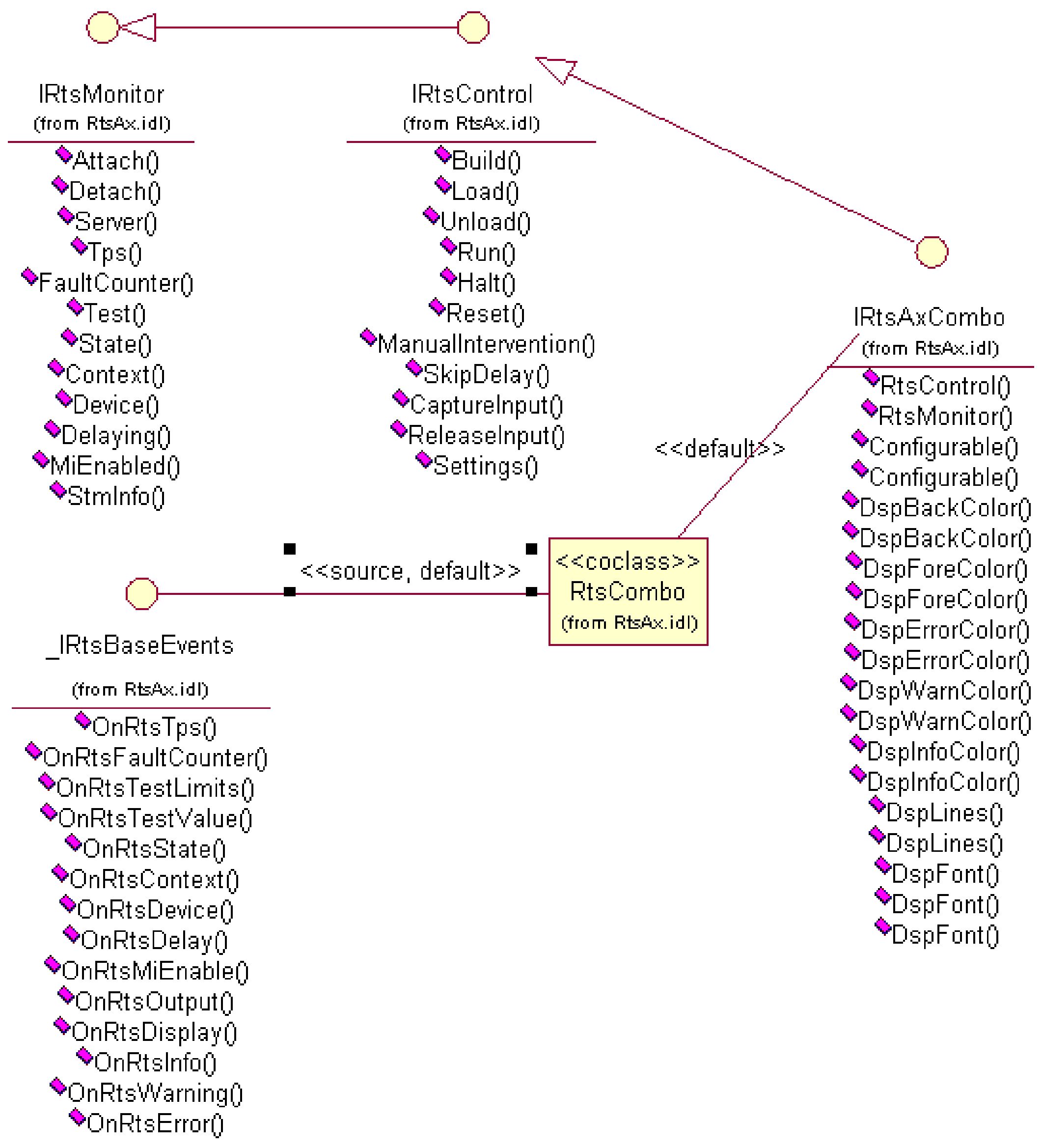


Figure Error! Bookmark not defined. **RTSCombo class design**

3.8.3 Property Pages

Combining the features of the *RtsMonitor* and *RtsControl* components the property pages made available by the *RtsCombo* component are the sum of the property pages exposed by *RtsMonitor* and *RtsControl*.

3.9 _IRtsBaseEvents 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 *_IRtsBaseEvents* interface groups the events related to RTS activity.

3.9.1 IDL description

```
[
	uuid(3F6B2908-F0DA-11D2-BBB0-00C0268914D3),
	helpstring("_IRtsBaseEvents Interface")
]
dispinterface _IRtsBaseEvents
{
	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);
};
```

3.9.2 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.

3.9.3 OnRtsFaultCount Event

Parameters

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

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

3.9.4 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.

3.9.5 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.

3.9.6 OnRtsState Event

Parameters

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

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

3.9.7 OnRtsContext Event

Parameters

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

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

3.9.8 OnRtsDevice Event

Parameters

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

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

3.9.9 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.

3.9.10 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

3.9.11 OnRtsOutput Event

Parameters

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

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

3.9.12 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.

3.9.13 OnRtsInfo Event

Parameters

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

The *OnRtsInfo* event is fired by information messages.

3.9.14 OnRtsWarning Event

Parameters

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

The *OnRtsWarning* event is fired by warning messages.

3.9.15 OnRtsError Event

Parameters

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

The *OnRtsError* event is fired by error messages.

4 Additional COM Components, Interfaces and Types

4.1 IRtsSettings interface

4.1.1 IDL Description

```
typedef enum RtsConfigOptions {
        OPT_CFG_SIMULATOR   = 0x0000,
        OPT_CFG_PRODUCTION  = 0x0001,
        OPT_CFG_DATALOGGER  = 0x0002,
        OPT_CFG_RTDG        = 0x0004,
        OPT_CFG_DYNCHANGER  = 0x0008,
} RtsConfigOptions;

typedef enum RtsResetAtOptions {
        OPT_RESET_AT_LOAD   = 0x0001,
        OPT_RESET_AT_UNLOAD = 0x0002,
} RtsResetAtOptions;

typedef enum RtsHaltAtOptions {
        OPT_HALT_AT_WARNING = 0x0001,
        OPT_HALT_AT_ERROR   = 0x0002,
        OPT_HALT_AT_NOGO    = 0x0004,
        OPT_HALT_AT_TEST    = 0x0008,
} RtsHaltAtOptions;

typedef enum RtsLogOptions {
        OPT_LOG_OUTPUT      = 0x0001,
        OPT_LOG_STATEMENT   = 0x0002,
        OPT_LOG_DISPLAY     = 0x0004,
        OPT_LOG_INFO        = 0x0008,
        OPT_LOG_WARNING     = 0x0010,
        OPT_LOG_ERROR       = 0x0020,
        OPT_LOG_BUS         = 0x0040,
} RtsLogOptions;

typedef enum RtsNamOptions {
        OPT_NAM_NORMAL      = 0x0000,
        OPT_NAM_MINIMIZE    = 0x0001,
        OPT_NAM_HIDE        = 0x0002,
        OPT_NAM_COM         = 0x0004,
} RtsNamOptions;

typedef enum RtsSoundOptions {
        OPT_SOUND_ERROR     = 0x0001,
        OPT_SOUND_WARNING   = 0x0002,
        OPT_SOUND_NOGO      = 0x0004,
        OPT_SOUND_FINISH    = 0x0008,
        OPT_SOUND_BRKHIT    = 0x0010,
        OPT_SOUND_RPTHIT    = 0x0020,
        OPT_SOUND_MI        = 0x0040,
        OPT_SOUND_INPUT     = 0x0080,
} RtsSoundOptions;

[
        object,
        uuid(3F6B2961-F0DA-11D2-BBB0-00C0268914D3),
        dual,
        helpstring("IRtsSettings Interface"),
        pointer_default(unique)
]
interface IRtsSettings : IDispatch
{
        [id(DISPID_ATTACH), helpstring("method Attach")]
        HRESULT Attach([in] IUnknown* pUnk);
        [id(DISPID_DETACH), helpstring("method Detach")]
        HRESULT Detach();
        [propget, id(DISPID_SERVER), helpstring("property Server")]
        HRESULT Server([out, retval] IDispatch** pVal);
        [id(DISPID_CONFIGURE), helpstring("method Configure")]
        HRESULT Configure([in] IUnknown* pUnk);
        [propget, id(DISPID_CONFIG), helpstring("property Config")]
        HRESULT Config([out, retval] long* pVal);
        [propput, id(DISPID_CONFIG), helpstring("property Config")]
        HRESULT Config([in] long newVal);
        [propget, id(DISPID_RESETOPTIONS), helpstring("property ResetOptions")]
        HRESULT ResetOptions([out, retval] long* pVal);
        [propput, id(DISPID_RESETOPTIONS), helpstring("property ResetOptions")]
        HRESULT ResetOptions([in] long newVal);
        [propget, id(DISPID_HALTOPTIONS), helpstring("property HaltOptions")]
        HRESULT HaltOptions([out, retval] long* pVal);
        [propput, id(DISPID_HALTOPTIONS), helpstring("property HaltOptions")]
        HRESULT HaltOptions([in] long newVal);
        [propget, id(DISPID_LOGENABLE), helpstring("property Enable Logging")]
        HRESULT LogEnable([out, retval] VARIANT_BOOL* pVal);
        [propput, id(DISPID_LOGENABLE), helpstring("property Enable Logging")]
        HRESULT LogEnable([in] VARIANT_BOOL newVal);
        [propget, id(DISPID_LOGAPPEND), helpstring("property Append Log")]
        HRESULT LogAppend([out, retval] VARIANT_BOOL* pVal);
        [propput, id(DISPID_LOGAPPEND), helpstring("property Append Log")]
        HRESULT LogAppend([in] VARIANT_BOOL newVal);
        [propget, id(DISPID_LOGOPTIONS), helpstring("property LogOptions")]
        HRESULT LogOptions([out, retval] long* pVal);
        [propput, id(DISPID_LOGOPTIONS), helpstring("property LogOptions")]
        HRESULT LogOptions([in] long newVal);
        [propget, id(DISPID_LOGFILENAME), helpstring("property LogFileName")]
        HRESULT LogFileName([out, retval] BSTR* pVal);
        [propput, id(DISPID_LOGFILENAME), helpstring("property LogFileName")]
        HRESULT LogFileName([in] BSTR newVal);
        [propget, id(DISPID_NAMOPTIONS), helpstring("property NamOptions")]
        HRESULT NamOptions([out, retval] long* pVal);
        [propput, id(DISPID_NAMOPTIONS), helpstring("property NamOptions")]
        HRESULT NamOptions([in] long newVal);
        [propget, id(DISPID_SOUNDEVENTS), helpstring("property SoundEvents")]
        HRESULT SoundEvents([out, retval] long* pVal);
        [propput, id(DISPID_SOUNDEVENTS), helpstring("property SoundEvents")]
        HRESULT SoundEvents([in] long newVal);
        [propget, id(DISPID_DLOGRECPROGID), helpstring("property DlogRecProgID")]
        HRESULT DlogRecProgID([out, retval] BSTR* pVal);
        [propput, id(DISPID_DLOGRECPROGID), helpstring("property DlogRecProgID")]
        HRESULT DlogRecProgID([in] BSTR newVal);
        [propget, id(DISPID_DLOGGUIPROGID), helpstring("property DlogGuiProgID")]
        HRESULT DlogGuiProgID([out, retval] BSTR* pVal);
        [propput, id(DISPID_DLOGGUIPROGID), helpstring("property DlogGuiProgID")]
        HRESULT DlogGuiProgID([in] BSTR newVal);
        [propget, id(DISPID_DLOGSERVER), helpstring("property DlogServer")]
        HRESULT DlogServer([out, retval] IDispatch** pVal);
        [propget, id(DISPID_DLOGRECORDER), helpstring("property DlogRecorder")]
        HRESULT DlogRecorder([out, retval] IDispatch** pVal);
        [propget, id(DISPID_DLOGGUI), helpstring("property DlogGui")]
        HRESULT DlogGui([out, retval] IDispatch** pVal);
        [propget, id(DISPID_IOSUBSYSTEM), helpstring("property IOSubsystem")]
        HRESULT IOSubsystem([out, retval] IUnknown** pVal);
        [id(DISPID_GETIORESOURCE), helpstring("method GetIOResource")]
        HRESULT GetIOResource([in]BSTR sName, [out, retval]IDispatch** pVal);
        [propget, id(DISPID_DEFAULTFORMAT), helpstring("property DefaultFormat")]
        HRESULT DefaultFormat([out, retval] BSTR* pVal);
        [propput, id(DISPID_DEFAULTFORMAT), helpstring("property DefaultFormat")]
        HRESULT DefaultFormat([in] BSTR newVal);
        [propget, id(DISPID_SWITCHSERVERMANAGER),
                  helpstring("property SwitchServerManager")]
        HRESULT SwitchServerManager([out, retval] IUnknown** pVal);
};
```

4.1.2 Attach Method

TYX internal use only, subject to change. Do not use in client applications.

4.1.3 Detach Method

TYX internal use only, subject to change. Do not use in client applications.

4.1.4 Server Property

TYX internal use only, subject to change. Do not use in client applications.

4.1.5 Configuration Method

TYX internal use only, subject to change. Do not use in client applications.

4.1.6 Config Property

Type	Access	Description
Long	Read-Write	RTS Server configuration.

Use this property to query or change the configuration of the RTS Server. Changing the *Config* property is allowed only for RTS Servers in the "UNLOADED" state. An error will be generated when attempting to commit the configuration to a RTS Server with a loaded TPS.
This property is a bit-field. The long integer value is a combination of the following constants exposed by the RtsAx type library as the *RtsConfigOptions* enumeration:

Config property bit-field constants

Name	Value	Description
OPT_CFG_SIMULATION	0x0000	Simulation setup, all drivers simulated
OPT_CFG_PRODUCTION	0x0001	Production module enabled, drivers will be invoked
OPT_CFG_DATALOGGER	0x0002	Data logger enabled
OPT_CFG_RTDG	0x0004	Runtime Test Diagram Generator enabled
OPT_CFG_DYNCHANGER	0x0008	Dynamic Parameter Changer enabled

Notes:
o o The default value is `OPT_CFG_SIMULATION`, production feature is disabled

4.1.7 ResetOptions Property

Type	Access	Description
Long	Read-Write	Controls the reset options for TPS load and unload operations

The *ResetOptions* property provides read-write access to the reset options. The reset options control the RTS behavior with respect to resetting the ATE automatically during TPS load and unload operations.

This property is a bit-field. The long integer value is a combination of the following constants exposed by the RtsAx type library as the *RtsResetAtOptions* enumeration:

ResetOptions property bit-field constants

Name	Value	Description
OPT_RESET_AT_LOAD	0x0001	Reset invoked after TPS load.
OPT_RESET_AT_UNLOAD	0x0002	Reset invoked before TPS unload.

Notes:
o o The default value is `0`, reset is not invoked automatically.

4.1.8 HaltOptions Property

Type	Access	Description
Long	Read-Write	Controls the RTS Halt conditions.

The *HaltOptions* property provides read-write access to the halt options. The halt options control under what conditions the RTS is halted automatically, without operator intervention.

This property is a bit-field. The long integer value is a combination of the following constants exposed by the RtsAx type library as the *RtsHaltAtOptions* enumeration:

HaltOptions property bit-field constants

Value	Value	Description
OPT_HALT_AT_WARNING	0x0001	Halt at warning conditions
OPT_HALT_AT_ERROR	0x0002	Halt at error conditions
OPT_HALT_AT_NOGO	0x0004	Halt at **NOGO** conditions - failed tests or comparisons
OPT_HALT_AT_TEST	0x0008	Halt at failed tests

Notes:

o o The default value is 0, RTS will not halt automatically.

4.1.9 LogEnable Property

Type	Access	Description
VARIANT_BOOL	Read-Write	Controls if RTS logging is enabled

Notes:

o o The default value is VARIANT_FALSE, logging is disabled

4.1.10 LogAppend Property

Type	Access	Description
VARIANT_BOOL	Read-Write	Controls the logging mode - append or overwrite

Notes:

o o The *LogAppend* property is ignored if logging is disabled

o o The default value is VARIANT_TRUE, logging occurs in append mode

4.1.11 LogOptions Property

Type	Access	Description
Long	Read-Write	Specifies the RTS log setting options.

The *LogOptions* property provides read-write access to the RTS logging options. The RTS logging options control what information and events are logged by the RTS.

This property is a bit-field. The long integer value is a combination of the following constants exposed by the RtsAx type library as the *RtsLogOptions* enumeration:

LogOptions property bit-field constants

Value	Value	Description
OPT_LOG_OUTPUT	0x0001	Log ATLAS output messages.
OPT_LOG_STATEMENT	0x0002	Log ATLAS statement information.
OPT_LOG_DISPLAY	0x0004	Log display messages, generated by instrument drivers.
OPT_LOG_INFO	0x0008	Log information messages.
OPT_LOG_WARNING	0x0010	Log warning messages.
OPT_LOG_ERROR	0x0020	Log error messages.

OPT_LOG_BUS	0x0040	Log bus traffic.

Notes:
- o The *LogOptions* property is ignored if logging is disabled
- o The default value is `OPT_LOG_ERROR | OPT_LOG_WARNING | OPT_LOG_BUS | OPT_LOG_OUTPUT | OPT_LOG_STATEMENT`.

4.1.12 LogFileName Property

Type	Access	Description
BSTR	Read-Write	Controls the log file name

Notes:
- o The *LogFileName* property is ignored if logging is disabled
- o The startup default value is “Log”

4.1.13 NamOptions Property

Type	Access	Description
RtsNam	Read-Write	Controls the Non-ATLAS Module execution options

The *NamOptions* property provides read-write access to the Non-ATLAS Module execution options. The Non-ATLAS Module execution options control what information and events are logged by the RTS.

This property is a bit-field. The long integer value is a combination of the following constants exposed by the RtsAx type library as the *RtsNamOptions* enumeration:

NamOptions property bit-field constants

Value	Value	Description
OPT_NAM_NORMAL	0x0000	NAM process started with normal visibility
OPT_NAM_MINIMIZE	0x0001	NAM process minimized at startup
OPT_NAM_HIDE	0x0002	NAM process hidden at startup
OPT_NAM_COM	0x0004	Enable COM NAM support

Notes:
- o The default value is `OPT_NAM_NORMAL | OPT_NAM_COM`

4.1.14 SoundEvents Property

Type	Access	Description
Long	Read-Write	Controls the RTS sound options, the sound generation conditions

The *SoundEvents* property provides read-write access to the RTS sound event options. The RTS sound event options specify for what events the RTS will generate sounds. The sounds can be customized by changing or providing the desired WAV file in the <usr>\tyx\media directory. If the required WAV file is missing a default system sound will be played.
This property is a bit-field. The long integer value is a combination of the following constants exposed by the RtsAx type library as the *RtsSoundOptions* enumeration:

SoundEvents property bit-field constants

Value	Value	Description
OPT_SOUND_ERROR	0x0001	“ERROR.WAV” played for RTS error conditions
OPT_SOUND_WARNING	0x0002	“WARNING.WAV” played for RTS warning conditions
OPT_SOUND_NOGO	0x0004	“NOGO.WAV” played for RTS NOGO conditions
OPT_SOUND_FINISH	0x0008	“FINISH.WAV” played the TPS execution is finished
OPT_SOUND_BRKHIT	0x0010	“BRKHIT.WAV” played the RTS encounters a breakpoint

OPT_SOUND_RPTHIT	0x0020	“RPTHIT.WAV” played the RTS encounters a repeat section
OPT_SOUND_MI	0x0040	“MIREQ.WAV” played when Manual-Intervention is enabled
OPT_SOUND_INPUT	0x0080	“INPUT.WAV” played when RTS waits for input

Notes:
o o The startup default value is 0, no sound are generated

4.1.15 DlogRecProgID Property[1][1]

Type	Access	Description
BSTR	Read-Write	Controls the data logger recorder ProgID.

Notes:
o o This property is used only if the data logger is enabled
o o The default value is “DataLogger.XmlDlogRecorder”
o o TYX internal use only, subject to change. Do not use in client applications.

4.1.16 DlogGuiProgID Property[1]

Type	Access	Description
BSTR	Read-Write	Controls the data logger GUI ProgID.

Notes:
o o This property is used only if the data logger is enabled
o o The default value is “DataLogger.HtmlDlogGui”
o o TYX internal use only, subject to change. Do not use in client applications.

4.1.17 DlogServer Property[1]

TYX internal use only, subject to change. Do not use in client applications.

4.1.18 DlogRecorder Property

Type	Access	Description
IDispatch*	Read-Only	Provides access to the data logger recorder object

The *DlogRecorder* property provides access to the data logger recorder for configuration purposes. This property is null if the data logger is not enabled

4.1.19 DlogGui Property

Type	Access	Description
IDispatch*	Read-Only	Gets the Data Logger GUI.

The *DlogGui* property provides access to the data logger GUI for configuration purposes. This property is null if the data logger is not enabled

4.1.20 IOSubsystem Property[2][2]

TYX internal use only, subject to change. Do not use in client applications.

[1][1] It is possible to change various settings related to Logging of information encountered by the RTS Server. TYX advices to programmatically retrieve the Property Pages associated with the object to do so.
[2][2] It is possible to change various settings related to IOSubSystem of the RTS Server. TYX advices to programmatically retrieve the Property Pages associated with the object to do so.

4.1.21 GetIOResource Method

Parameters

Name	Type	Access	Description
sName	BSTR	[in]	The name of resource to be retrieved.
pVal	IDispatch**	[out, retval]	Pointer to the resources's IDispatch interface pointer

The GetIOResource method provides access to IOResources by name for configuration purposes. The "PRINTER" and "SOUND" resources are typically reconfigured programmatically by client applications but the method provides access to all resources. The I/O resource specified by the *sName* parameter must implement the *IDispatch* interface, else a COM error will be returned. All TYX provided I/O resources meet the above requirement.

4.1.22 DefaultFormat Property[3[3]]

TYX internal use only, subject to change. Do not use in client applications.

4.1.23 SwitchServerManager Property[4[4]]

TYX internal use only, subject to change. Do not use in client applications.

4.1.24 Dispatch Identifiers

```
const DISPID DISPID_ATTACH                    = 31;
const DISPID DISPID_DETACH                    = 32;
const DISPID DISPID_SERVER                    = 33;
const DISPID DISPID_CONFIGURE                 = 10;
const DISPID DISPID_CONFIG                    = 11;
const DISPID DISPID_RESETOPTIONS              = 12;
const DISPID DISPID_HALTOPTIONS               = 13;
const DISPID DISPID_LOGENABLE                 = 14;
const DISPID DISPID_LOGAPPEND                 = 15;
const DISPID DISPID_LOGOPTIONS                = 16;
const DISPID DISPID_LOGFILENAME               = 17;
const DISPID DISPID_NAMOPTIONS                = 18;
const DISPID DISPID_SOUNDEVENTS               = 19;
const DISPID DISPID_DLOGRECPROGID             = 20;
const DISPID DISPID_DLOGGUIPROGID             = 21;
const DISPID DISPID_DLOGSERVER                = 22;
const DISPID DISPID_DLOGRECORDER              = 23;
const DISPID DISPID_DLOGGUI                   = 24;
const DISPID DISPID_IOSUBSYSTEM               = 25;
const DISPID DISPID_GETIORESOURCE             = 26;
const DISPID DISPID_DEFAULTFORMAT             = 27;
const DISPID DISPID_SWITCHSERVERMANAGER       = 28;
```

4.2 RtsSettings CoClass

RtsSettings is a COM object used by the RTS COM adapters to implement the *Settings* property. The RTS options and settings exposed by the *IRtsSettings* interface. The component implements a variety of other standard COM interfaces such as *IPersistStreamInit*, *IPersistPropertyBag* and *ISpecifyPropertyPages*.

[3[3]] It is possible to change various settings related to DataFormat of the RTS Server. TYX advices to programmatically retrieve the Property Pages associated with the object to do so.

[4[4]] It is possible to change various settings related to Switch Server Manager of the RTS Server. TYX advices to programmatically retrieve the Property Pages associated with the object to do so.

4.2.1 IDL Description

```
[
	uuid(3F6B2960-F0DA-11D2-BBB0-00C0268914D3),
	helpstring("RtsSettings Class")
]
coclass RtsSettings
{
	[default] interface IRtsSettings;
};
```

4.2.2 UML Design

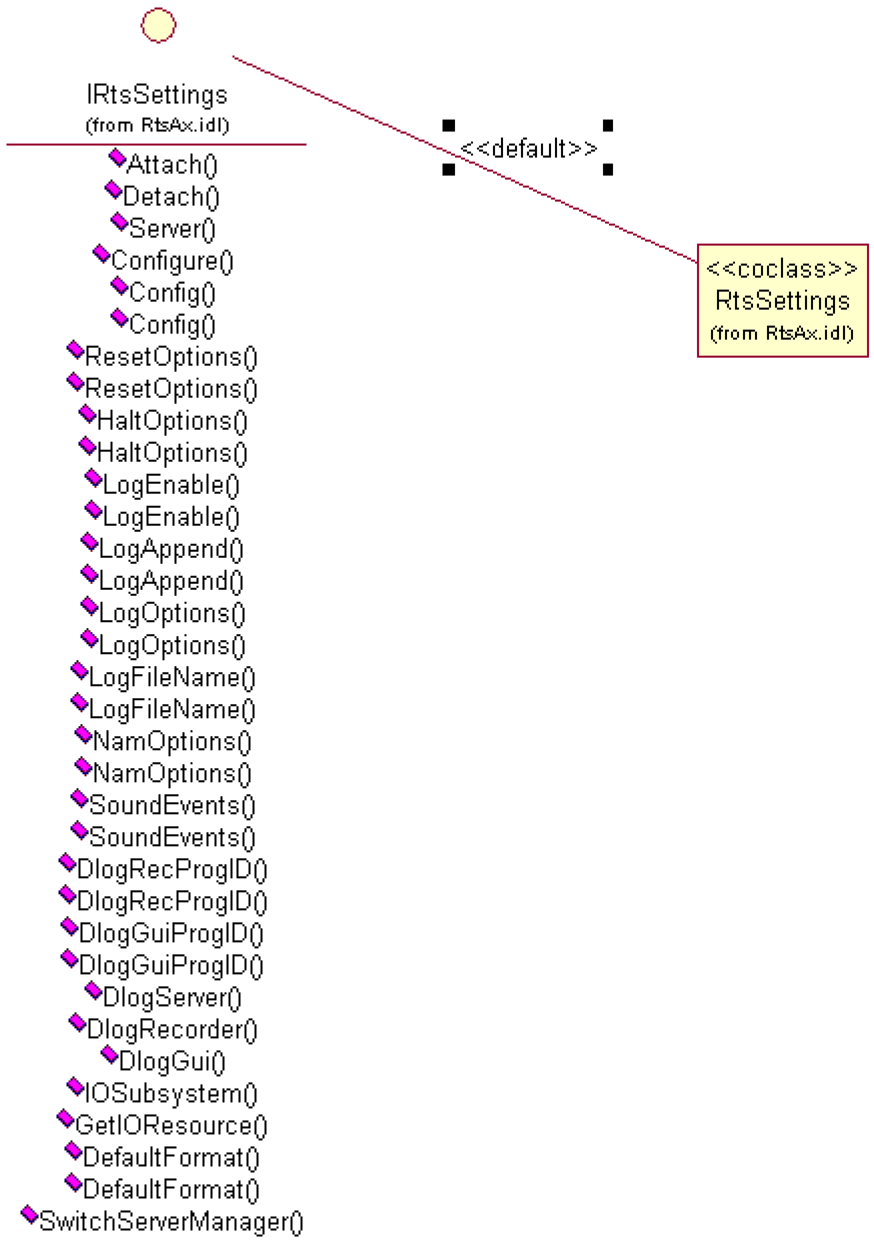
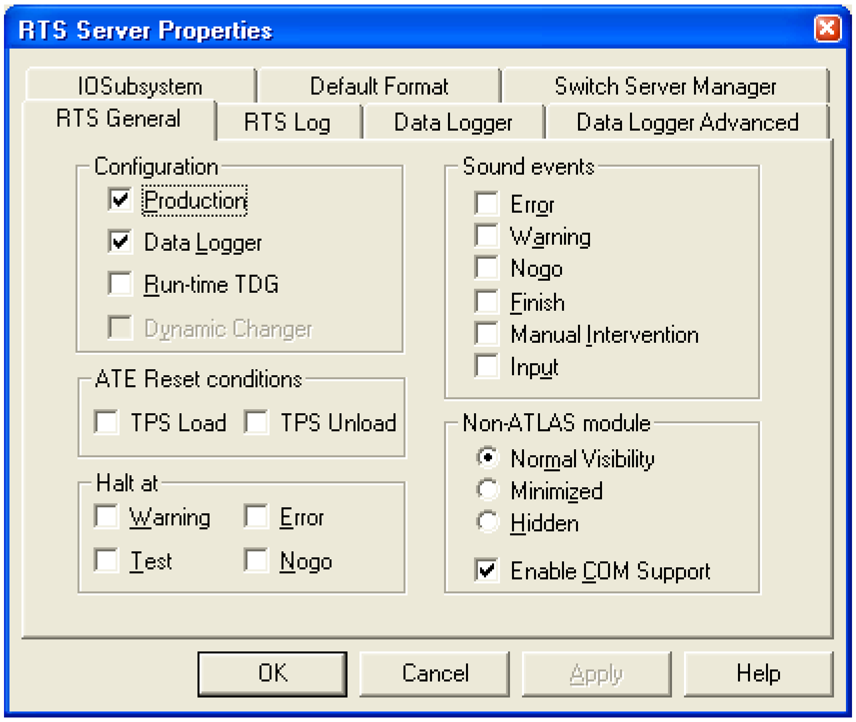


Figure Error! Bookmark not defined. **RtsSettings class design**

4.2.3 Property Pages



Configuration
- [✓] Production
- [✓] Data Logger
- [ ] Run-time TDG
- [ ] Dynamic Changer

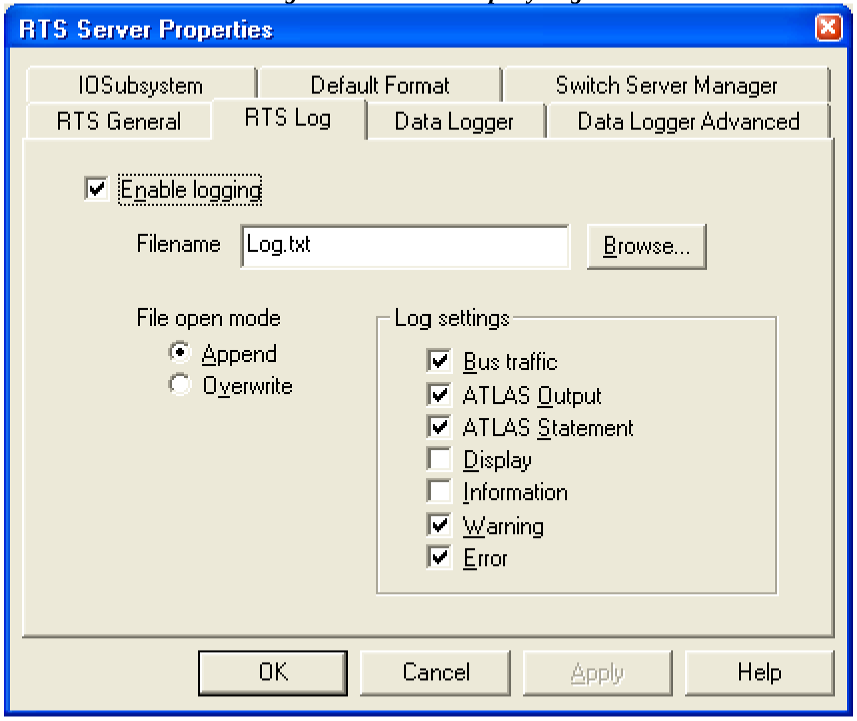
ATE Reset conditions
- [ ] TPS Load
- [ ] TPS Unload

Halt at
- [ ] Warning
- [ ] Error
- [ ] Test
- [ ] Nogo

Sound events
- [ ] Error
- [ ] Warning
- [ ] Nogo
- [ ] Finish
- [ ] Manual Intervention
- [ ] Input

Non-ATLAS module
- [✓ Normal Visibility]
- Minimized
- Hidden
- [✓] Enable COM Support

Figure 22 General Property Page



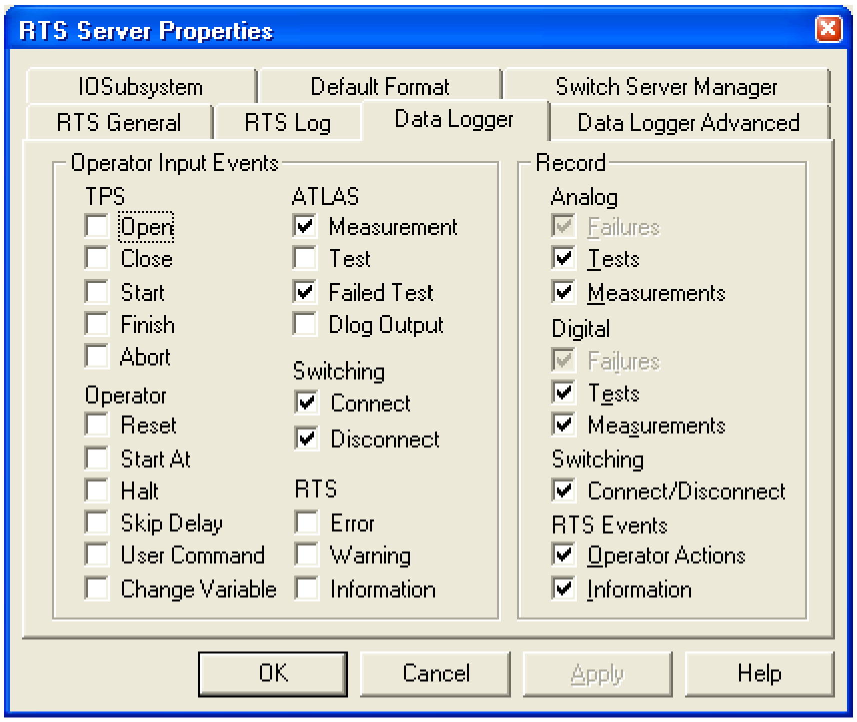
- [✓] Enable logging

Filename: Log.txt

File open mode
- [✓ Append]
- Overwrite

Log settings
- [✓] Bus traffic
- [✓] ATLAS Output
- [✓] ATLAS Statement
- [ ] Display
- [ ] Information
- [✓] Warning
- [✓] Error

Figure 23 RTS Log Property Page



Operator Input Events

TPS
- [ ] Open
- [ ] Close
- [ ] Start
- [ ] Finish
- [ ] Abort

Operator
- [ ] Reset
- [ ] Start At
- [ ] Halt
- [ ] Skip Delay
- [ ] User Command
- [ ] Change Variable

ATLAS
- [✓] Measurement
- [ ] Test
- [✓] Failed Test
- [ ] Dlog Output

Switching
- [✓] Connect
- [✓] Disconnect

RTS
- [ ] Error
- [ ] Warning
- [ ] Information

Record

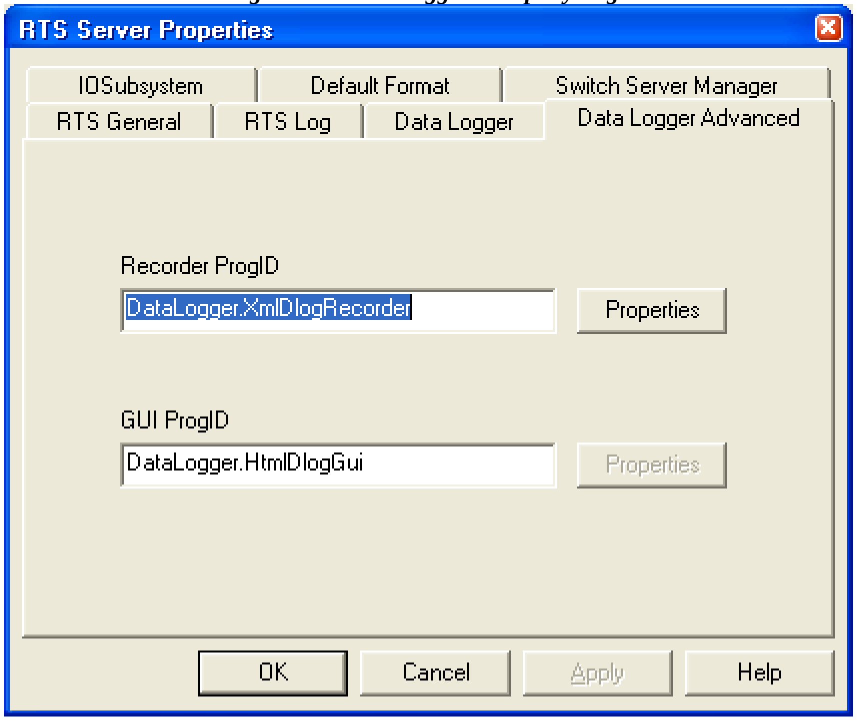
Analog
- [✓] Failures
- [✓] Tests
- [✓] Measurements

Digital
- [✓] Failures
- [✓] Tests
- [✓] Measurements

Switching
- [✓] Connect/Disconnect

RTS Events
- [✓] Operator Actions
- [✓] Information

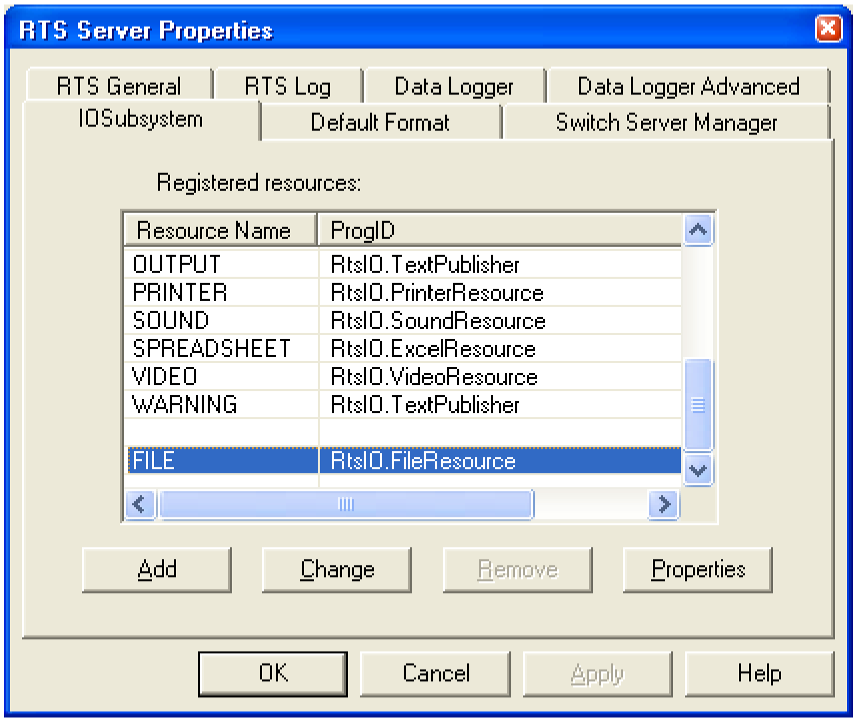
Figure 24 "Data Logger" Property Page



Recorder ProgID: DataLogger.XmlDlogRecorder

GUI ProgID: DataLogger.HtmlDlogGui

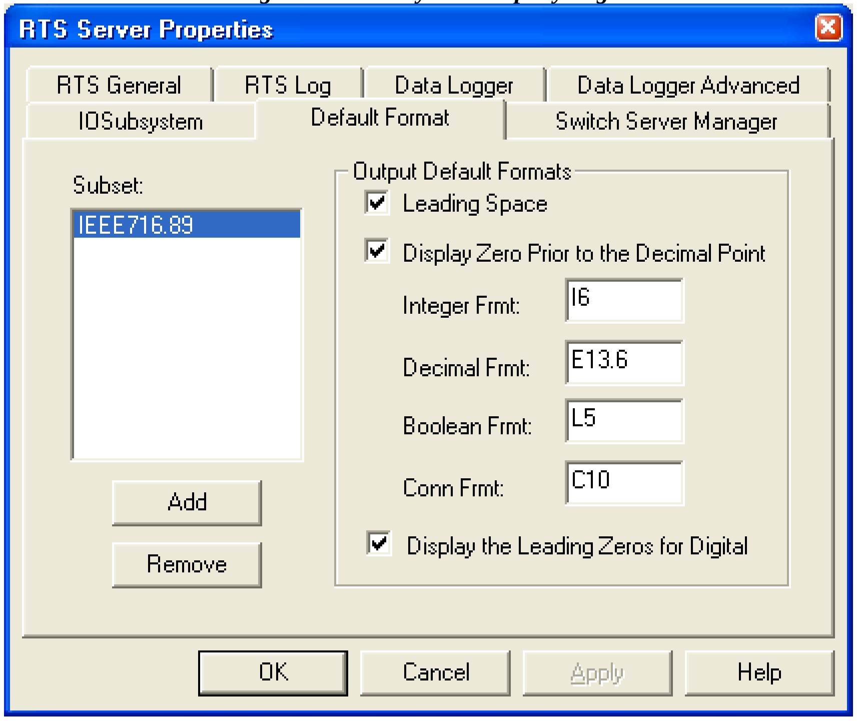
Figure 25 Data Logger Advanced Property Page


RTS Server Properties

Registered resources:

Resource Name	ProgID
OUTPUT	RtsIO.TextPublisher
PRINTER	RtsIO.PrinterResource
SOUND	RtsIO.SoundResource
SPREADSHEET	RtsIO.ExcelResource
VIDEO	RtsIO.VideoResource
WARNING	RtsIO.TextPublisher
FILE	RtsIO.FileResource

Figure 26 IOSubsystem Property Page


RTS Server Properties

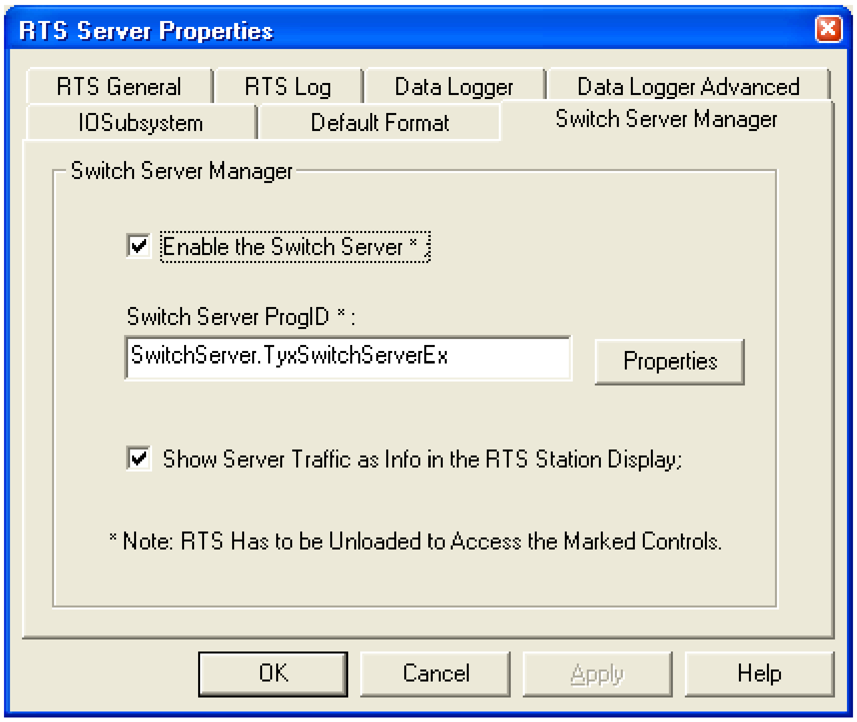
Subset:
IEEE716.89

Output Default Formats
[✓] Leading Space
[✓] Display Zero Prior to the Decimal Point

Integer Frmt:	I6
Decimal Frmt:	E13.6
Boolean Frmt:	L5
Conn Frmt:	C10

[✓] Display the Leading Zeros for Digital

Figure 27 DefaultFormat Property Page



Switch Server Manager

[✓] Enable the Switch Server *

Switch Server ProgID * : SwitchServer.TyxSwitchServerEx

[✓] Show Server Traffic as Info in the RTS Station Display;

* Note: RTS Has to be Unloaded to Access the Marked Controls.

Figure 28 SwitchServerManager Property Page

4.3 IAnsiDisplay

The interface provides enhanced text and look and feel features to enhance display of ATE programs.

4.3.1 IDL Description

```
[
	object,
	uuid(3F6B2951-F0DA-11D2-BBB0-00C0268914D3),
	dual,
	helpstring("IAnsiDisplay Interface"),
	pointer_default(unique)
]
interface IAnsiDisplay : IDispatch
{
	[propput, id(DISPID_BACKCOLOR)]
	HRESULT BackColor([in]OLE_COLOR clr);
	[propget, id(DISPID_BACKCOLOR)]
	HRESULT BackColor([out,retval]OLE_COLOR* pclr);
	[propput, id(DISPID_BORDERVISIBLE)]
	HRESULT BorderVisible([in]VARIANT_BOOL newVal);
	[propget, id(DISPID_BORDERVISIBLE)]
	HRESULT BorderVisible([out, retval]VARIANT_BOOL* pVal);
	[propputref, id(DISPID_FONT)]
	HRESULT Font([in]IFontDisp* pFont);
	[propput, id(DISPID_FONT)]
	HRESULT Font([in]IFontDisp* pFont);
	[propget, id(DISPID_FONT)]
	HRESULT Font([out, retval]IFontDisp** ppFont);
	[propput, id(DISPID_FORECOLOR)]
	HRESULT ForeColor([in]OLE_COLOR clr);
	[propget, id(DISPID_FORECOLOR)]
	HRESULT ForeColor([out,retval]OLE_COLOR* pclr);
	[propget, id(DISPID_ERRORCOLOR), helpstring("property ErrorColor")]
```

```
        HRESULT ErrorColor([out, retval] OLE_COLOR* pVal);
        [propput, id(DISPID_ERRORCOLOR), helpstring("property ErrorColor")]
        HRESULT ErrorColor([in] OLE_COLOR newVal);
        [propget, id(DISPID_WARNINGCOLOR), helpstring("property WarningColor")]
        HRESULT WarningColor([out, retval] OLE_COLOR* pVal);
        [propput, id(DISPID_WARNINGCOLOR), helpstring("property WarningColor")]
        HRESULT WarningColor([in] OLE_COLOR newVal);
        [propget, id(DISPID_INFOCOLOR), helpstring("property InfoColor")]
        HRESULT InfoColor([out, retval] OLE_COLOR* pVal);
        [propput, id(DISPID_INFOCOLOR), helpstring("property InfoColor")]
        HRESULT InfoColor([in] OLE_COLOR newVal);
        [propget, id(DISPID_LINES), helpstring("property Lines")]
        HRESULT Lines([out, retval] long* pVal);
        [propput, id(DISPID_LINES), helpstring("property Lines")]
        HRESULT Lines([in] long newVal);
        [propget, id(DISPID_AUTOURL), helpstring("property AutoURL")]
        HRESULT AutoURL([out, retval] VARIANT_BOOL* pVal);
        [propput, id(DISPID_AUTOURL), helpstring("property AutoURL")]
        HRESULT AutoURL([in] VARIANT_BOOL newVal);
        [id(1), helpstring("method DisplayText")]
        HRESULT DisplayText([in] BSTR val);
        [id(2), helpstring("method DisplayError")]
        HRESULT DisplayError([in] BSTR val);
        [id(3), helpstring("method DisplayWarning")]
        HRESULT DisplayWarning([in] BSTR val);
        [id(4), helpstring("method DisplayInfo")]
        HRESULT DisplayInfo([in] BSTR val);
        [id(5), helpstring("method Clear")]
        HRESULT Clear();
};
```

4.3.2 BackColor Property

Type	Access	Description
`OLE_COLOR`	`Read-Write`	Controls the background color of the display.

4.3.3 BorderVisible Property

Type	Access	Description
`VARIANT_BOOL`	`Read-Write`	Controls the visibility of the border of the display.

4.3.4 Font Property

Type	Access	Description
`IFontDisp`	`Read-Write`	Controls the font of the display.

The IFontDisplay interface is used to set/read the Font property of the display.

4.3.5 ForeColor Property

Type	Access	Description
`OLE_COLOR`	`Read-Write`	Controls the foreground color of the display.

4.3.6 ErrorColor Property

Type	Access	Description
`OLE_COLOR`	`Read-Write`	Controls the color used for display of errors.

4.3.7 WarningColor Property

Type	Access	Description
OLE_COLOR	Read-Write	Controls the color used for display of warnings.

4.3.8 InfoColor Property

Type	Access	Description
OLE_COLOR	Read-Write	Controls the color used for display of information.

4.3.9 Lines Property

Type	Access	Description
Long	Read-Write	Controls drawing of Lines in the display.

4.3.10 AutoURL Property

Type	Access	Description
Long	Read-Write	Controls display of a URL

4.3.11 DisplayText Method

Parameters

Name	Type	Access	Description
Val	BSTR	[in]	Displays text.

4.3.12 DisplayError Method

Parameters

Name	Type	Access	Description
Val	BSTR	[in]	Displays error text.

Displayed text uses the *ErrorColor* property for the text color.

4.3.13 DisplayWarning Method

Parameters

Name	Type	Access	Description
Val	BSTR	[in]	Displays warning text.

Displayed text uses the *WarningColor* property for the text color.

4.3.14 DisplayInfo Method

Parameters

Name	Type	Access	Description
Val	BSTR	[in]	Displays information text.

Displayed text uses the *InfoColor* property for the text color.

4.3.15 Clear Method

This method clears the display.

4.3.16 Dispatch Identifiers

```
#define DISPID_BACKCOLOR               (-501)
#define DISPID_BORDERVISIBLE           (-519)
#define DISPID_FONT                    (-512)
#define DISPID_FORECOLOR               (-513)
const DISPID DISPID_ERRORCOLOR          = 91;
const DISPID DISPID_WARNINGCOLOR        = 92;
const DISPID DISPID_INFOCOLOR           = 93;
const DISPID DISPID_LINES               = 94;
const DISPID DISPID_AUTOURL             = 95;
```

4.4 AnsiDisplay CoClass

ANSIDisplay is an *ActiveX* component that emulates the behavior of the RTS executives “*Station Display*”. Like the station display it also recognizes various escape sequences returned by the RTS, thus providing a richer text display.

4.4.1 IDL Description

```
[
	uuid(3F6B2950-F0DA-11D2-BBB0-00C0268914D3),
	helpstring("AnsiDisplay Class")
]
coclass AnsiDisplay
{
	[default] interface IAnsiDisplay;
};
```

4.4.2 UML Design

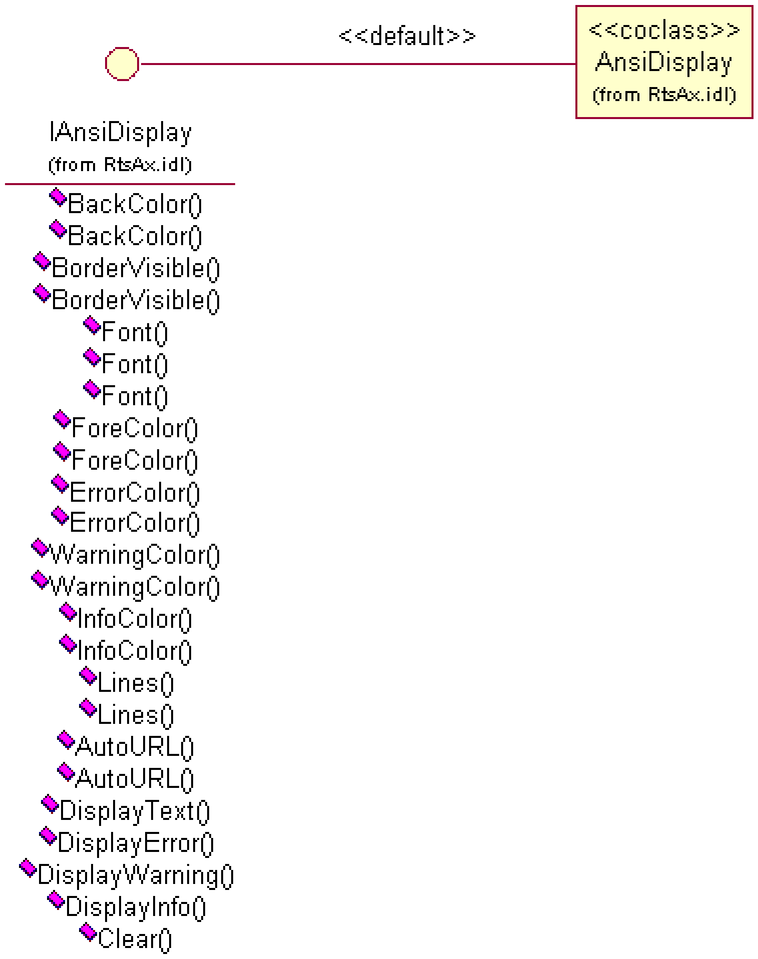


Figure Error! Bookmark not defined. **AddressInformation class design**

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