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")]
}
3.1.2 Attach Method

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>varServer</td>
<td>VARIANT</td>
<td>[in, optional]</td>
<td>RTS Server or the host computer.</td>
</tr>
</tbody>
</table>

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. 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. 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. 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. 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. All other types will generate an error.
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

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDispatch*</td>
<td>Read-Only</td>
<td>Interface pointer to RTS Server</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSTR</td>
<td>Read-Only</td>
<td>TPS full file path.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>Read-Only</td>
<td>The number of faults occurred during the current execution</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDispatch*</td>
<td>Read-Only</td>
<td>Pointer to the IDispatch interface of the test object.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>Read-Only</td>
<td>The state of the RTS COM Adapter</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTSAX STATE_DETACHED</td>
<td>-1</td>
<td>Adapter object is disconnected from RTS Server</td>
</tr>
<tr>
<td>RTSAX STATE_UNLOADED</td>
<td>0</td>
<td>No TPS was loaded or TPS was unloaded</td>
</tr>
<tr>
<td>RTSAX STATE_READY</td>
<td>1</td>
<td>TPS loaded, server initialized and ready for execution</td>
</tr>
<tr>
<td>RTSAX STATE_RUNNING</td>
<td>2</td>
<td>Server is running TPS</td>
</tr>
<tr>
<td>RTSAX STATE_HALTIED</td>
<td>3</td>
<td>TPS execution is halted</td>
</tr>
<tr>
<td>RTSAX STATE_FINISH</td>
<td>4</td>
<td>TPS execution complete, FINISH or TERMINATE reached</td>
</tr>
</tbody>
</table>
3.1.9 Context Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>Read-Only</td>
<td>Context of execution</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTSAX_CTX_DETACHED</td>
<td>-1</td>
<td>Adapter object is disconnected from RTS Server</td>
</tr>
<tr>
<td>RTSAX_CTX_UNLOADED</td>
<td>0</td>
<td>The RTS Server is in TPS unloaded context</td>
</tr>
<tr>
<td>RTSAX_CTX_INIT</td>
<td>1</td>
<td>The RTS Server is in TPS initialization context</td>
</tr>
<tr>
<td>RTSAX_CTX_TPS</td>
<td>2</td>
<td>The RTS Server is in TPS execution context</td>
</tr>
<tr>
<td>RTSAX_CTX_TERMINAL</td>
<td>3</td>
<td>The RTS Server is in TPS unloading or termination context</td>
</tr>
</tbody>
</table>
### 3.1.10 Device Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSTR</td>
<td>Read-Only</td>
<td>The current or last device accessed by RTS</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARIANT_BOOL</td>
<td>Read-Only</td>
<td>Indicates if the RTS Server is delaying</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARIANT_BOOL</td>
<td>Read-Only</td>
<td>Indicates if the manual intervention is enabled</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDispatch*</td>
<td>Read-Only</td>
<td>An interface pointer to the StmInfo object</td>
</tr>
</tbody>
</table>

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

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

```idl
[ object, 
  uuid(3F6B2902-F0DA-11D2-BBB0-00C0268914D3), 
  dual, 
  helpstring("IRtsControl Interface"), 
  pointer_default(unique) ]
}
```
```idl
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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>varHost</td>
<td>VARIANT</td>
<td>[in, optional]</td>
<td>Name or IP address of the host computer</td>
</tr>
</tbody>
</table>
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. **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. **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. **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. **All other types** will generate an error.
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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strTps</td>
<td>BSTR</td>
<td>[in]</td>
<td>Full path to the PAWS project file to open</td>
</tr>
</tbody>
</table>

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”.

![Sequence Diagram](image)

**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”.

![Unload Sequence Diagram](image)

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.
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.
3.2.7 Reset Method

The Reset method starts the ATE and TPS reset sequence. During the reset sequence the RTS executive runs the \texttt{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.
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>varRsnc</td>
<td>VARIANT</td>
<td>[in, optional]</td>
<td>The desired input resource</td>
</tr>
</tbody>
</table>

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

The possible types and values of the optional varRsnc VARIANT parameter are:
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. VT_ERROR – this case occurs from scripting languages when the optional parameter is omitted. The behavior is the same as for case 1.

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. 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. 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

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRTsSettings*</td>
<td>Read-Only</td>
<td>Interface pointer to the RTS settings object.</td>
</tr>
</tbody>
</table>

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

```c
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

```idl
data object,
data uuid(3F6B2911-F0DA-11D2-BBB0-00C0268914D3),
data dual,
data helpstring("IRtsAxMonitor Interface"),
data 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);
    [ propget, 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);
    [ propput, 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

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARIANT_BOOL</td>
<td>Read-Write</td>
<td>Controls if run-time configuration is enabled</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLE_COLOR</td>
<td>Read-Write</td>
<td>Controls the background color of the output window</td>
</tr>
</tbody>
</table>

3.3.4 DspForeColor Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLE_COLOR</td>
<td>Read-Write</td>
<td>Controls the text foreground color for the output window</td>
</tr>
</tbody>
</table>

3.3.5 DspErrorColor Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLE_COLOR</td>
<td>Read-Write</td>
<td>Controls the foreground color for error messages</td>
</tr>
</tbody>
</table>

3.3.6 DspWarnColor Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLE_COLOR</td>
<td>Read-Write</td>
<td>Controls the foreground color for warning messages</td>
</tr>
</tbody>
</table>

3.3.7 DspInfoColor Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLE_COLOR</td>
<td>Read-Write</td>
<td>Controls the foreground color for information messages</td>
</tr>
</tbody>
</table>

3.3.8 DspLines Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>Read-Write</td>
<td>Controls the maximum number of lines stored for the output window</td>
</tr>
</tbody>
</table>

3.3.9 DspFont Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFontDisp*</td>
<td>Read-Write</td>
<td>Controls the font used by the output window</td>
</tr>
</tbody>
</table>

3.3.10 Dispatch Identifiers

```c
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

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

3.4.2 UML Design

![RTSMonitor class design](image)

**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.

![Color Property Page](image1.png)

![Font Property Page](image2.png)

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

```plaintext
[ 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

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARIANT_BOOL</td>
<td>Read-Write</td>
<td>Controls if run-time configuration is enabled</td>
</tr>
</tbody>
</table>

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

```plaintext
[ uuid(3F6B2920-F0DA-11D2-BBB0-00C0268914D3),
  helpstring("RtsControl Class")
]
coclass RtsControl
{
  [default] interface IRtsAxControl;
  [default, source] dispinterface _IRtsBaseEvents;
};
```
Figure Error! Bookmark not defined. RTSControl class design
3.6.3 PropertyPages

**Figure 13 General Property Page**

**Figure 14 RTS Log Property Page**
Figure 15 “Data Logger” Property Page

Figure 16 Data Logger Advanced Property Page
Figure 17 IOSubsystem Property Page

Figure 18 DefaultFormat 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),
    ```
HRESULT DspForeColor([in] OLE_COLOR clr);  
[propput, id(DISPID_DSPFORECOLOR),  
helpstring("property Display ForeColor")]

HRESULT DspForeColor([out, retval] OLE_COLOR* p clr);  
[propput, id(DISPID_DSPERRORCOLOR),  
helpstring("property Display ErrorColor")]

HRESULT DspForeColor([out, retval] OLE_COLOR* pVal);  
[propput, id(DISPID_DSPERRORCOLOR),  
helpstring("property Display ErrorColor")]

HRESULT DspWarnColor([in] OLE_COLOR newVal);  
[propput, 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);  
[propput, id(DISPID_DSPWARNCOLOR),  
helpstring("property Display WarningColor")]

HRESULT DspInfoColor([out, retval] OLE_COLOR* pVal);  
[propput, id(DISPID_DSPINFOCOLOR),  
helpstring("property Display InfoColor")]

HRESULT DspInfoColor([in] OLE_COLOR newVal);  
[propput, id(DISPID_DSPINFOCOLOR),  
helpstring("property Display InfoColor")]

HRESULT DspLines([out, retval] long* pVal);  
[propputref, id(DISPID_DSPLINES), helpstring("property Display Lines")]

HRESULT DspLines([in] long newVal);  
[propputref, id(DISPID_DSPLINES), helpstring("property Display Lines")]

HRESULT DspFont([in] IFontDisp* pFont);  
[propputref, id(DISPID_DSPFONT), helpstring("property Display Font")]

HRESULT DspFont([in] IFontDisp* pFont);  
[propputref, id(DISPID_DSPFONT), helpstring("property Display Font")]

HRESULT DspFont([out, retval] IFontDisp** ppFont);

3.7.2 RtsControl Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRtsAxControl*</td>
<td>Read-Only</td>
<td>Interface to the inner RTS Control object</td>
</tr>
</tbody>
</table>

3.7.3 RtsMonitor Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRtsAxMonitor*</td>
<td>Read-Only</td>
<td>Interface to the inner RTS Monitor object</td>
</tr>
</tbody>
</table>

3.7.4 Configurable Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARIANT_BOOL</td>
<td>Read-Write</td>
<td>Controls if run-time configuration is enabled</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLE_COLOR</td>
<td>Read-Write</td>
<td>Controls the background color of the output window</td>
</tr>
</tbody>
</table>
3.7.6 DspForeColor Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLE_COLOR</td>
<td>Read-Write</td>
<td>Controls the text foreground color for the output window</td>
</tr>
</tbody>
</table>

3.7.7 DspErrorColor Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLE_COLOR</td>
<td>Read-Write</td>
<td>Controls the foreground color for error messages</td>
</tr>
</tbody>
</table>

3.7.8 DspWarnColor Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLE_COLOR</td>
<td>Read-Write</td>
<td>Controls the foreground color for warning messages</td>
</tr>
</tbody>
</table>

3.7.9 DspInfoColor Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLE_COLOR</td>
<td>Read-Write</td>
<td>Controls the foreground color for information messages</td>
</tr>
</tbody>
</table>

3.7.10 DspLines Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>Read-Write</td>
<td>Controls the maximum number of lines stored for the output window</td>
</tr>
</tbody>
</table>

3.7.11 DspFont Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFontDisp*</td>
<td>Read-Write</td>
<td>Controls the font used by the output window</td>
</tr>
</tbody>
</table>

3.7.12 Dispatch Identifiers

```c
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

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

- IRtMonitor
  - Attach()
  - Detach()
  - Server()
  - Tps()
  - FaultCounter()
  - Test()
  - State()
  - Context()
  - Device()
  - Delaying()
  - MIEnabled()
  - StrmInfo()

- IRtsControl
  - Build()
  - Load()
  - Unload()
  - Run()
  - Halt()
  - Reset()
  - ManualIntervention()
  - SkipDelay()
  - CaptureInput()
  - ReleaseInput()
  - Settings()

- IRtAxCombo
  - RtsControl()
  - RtsMonitor()
  - Configurable()
  - DspBackColor()
  - DspBackColor()
  - DspForeColor()
  - DspForeColor()
  - DspErrorColor()
  - DspErrorColor()
  - DspWarnColor()
  - DspWarnColor()
  - DspInfoColor()
  - DspInfoColor()
  - DspLines()
  - DspLines()
  - DspFont()
  - DspFont()

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

```idl
[ 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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strTps</td>
<td>BSTR</td>
<td>[in]</td>
<td>Full TPS path</td>
</tr>
</tbody>
</table>

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

### 3.9.4 OnRtsTestLimits Event

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pTest</td>
<td>IDispatch*</td>
<td>[in]</td>
<td>Pointer to the IDispatch interface of the test object.</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pTest</td>
<td>IDispatch*</td>
<td>[in]</td>
<td>Pointer to the IDispatch interface of the test object.</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
</table>

The OnRtsState event is fired when the RTS state changes.

### 3.9.7 OnRtsContext Event

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
</table>

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

### 3.9.8 OnRtsDevice Event

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strDevice</td>
<td>BSTR</td>
<td>[in]</td>
<td>The new active device</td>
</tr>
</tbody>
</table>

The OnRtsDevice event is fired when the current device changes.

### 3.9.9 OnRtsDelay Event

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dTime</td>
<td>double</td>
<td>[in]</td>
<td>Remaining delay amount in seconds</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEnable</td>
<td>VARIANT_BOOL</td>
<td>[in]</td>
<td>Desired action - enable or disable</td>
</tr>
</tbody>
</table>

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

3.9.11 OnRtsOutput Event

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strMsg</td>
<td>BSTR</td>
<td>[in]</td>
<td>Text to display</td>
</tr>
</tbody>
</table>

The OnRtsOutput event is fired by ATLAS OUTPUT, to ‘DISPLAY’ statements.

3.9.12 OnRtsDisplay Event

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strMsg</td>
<td>BSTR</td>
<td>[in]</td>
<td>Text to display</td>
</tr>
</tbody>
</table>

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

3.9.13 OnRtsInfo Event

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strMsg</td>
<td>BSTR</td>
<td>[in]</td>
<td>Information message to display</td>
</tr>
</tbody>
</table>

The OnRtsInfo event is fired by information messages.

3.9.14 OnRtsWarning Event

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strMsg</td>
<td>BSTR</td>
<td>[in]</td>
<td>Warning message to display</td>
</tr>
</tbody>
</table>

The OnRtsWarning event is fired by warning messages.

3.9.15 OnRtsError Event

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strMsg</td>
<td>BSTR</td>
<td>[in]</td>
<td>Error message to display</td>
</tr>
</tbody>
</table>

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 {

typedef enum RtsConfigOptions {
    OPT_CFG_SIMULATOR = 0x0000,
    OPT_CFG_PRODUCTION = 0x0001,
    OPT_CFG_DATALOGGER = 0x0002,
    OPT_CFG_RTID = 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_SWITCHSEVERMANAGER), 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

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>Read-Write</td>
<td>RTS Server configuration.</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT_CFG_SIMULATION</td>
<td>0x0000</td>
<td>Simulation setup, all drivers simulated</td>
</tr>
<tr>
<td>OPT_CFG_PRODUCTION</td>
<td>0x0001</td>
<td>Production module enabled, drivers will be invoked</td>
</tr>
<tr>
<td>OPT_CFG_DATALOGGER</td>
<td>0x0002</td>
<td>Data logger enabled</td>
</tr>
<tr>
<td>OPT_CFG_RTDDG</td>
<td>0x0004</td>
<td>Runtime Test Diagram Generator enabled</td>
</tr>
<tr>
<td>OPT_CFG_DYNCHANGER</td>
<td>0x0008</td>
<td>Dynamic Parameter Changer enabled</td>
</tr>
</tbody>
</table>

**Notes:**
- The default value is OPT_CFG_SIMULATION, production feature is disabled

4.1.7 ResetOptions Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>Read-Write</td>
<td>Controls the reset options for TPS load and unload operations</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT_RESET_AT_LOAD</td>
<td>0x0001</td>
<td>Reset invoked after TPS load.</td>
</tr>
<tr>
<td>OPT_RESET_AT_UNLOAD</td>
<td>0x0002</td>
<td>Reset invoked before TPS unload.</td>
</tr>
</tbody>
</table>

**Notes:**
- The default value is 0, reset is not invoked automatically.
4.1.8  HaltOptions Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>Read-Write</td>
<td>Controls the RTS Halt conditions.</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Value</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT_HALT_AT_WARNING</td>
<td>0x0001</td>
<td>Halt at warning conditions</td>
</tr>
<tr>
<td>OPT_HALT_AT_ERROR</td>
<td>0x0002</td>
<td>Halt at error conditions</td>
</tr>
<tr>
<td>OPT_HALT_AT_NOGO</td>
<td>0x0004</td>
<td>Halt at NOGO conditions - failed tests or comparisons</td>
</tr>
<tr>
<td>OPT_HALT_AT_TEST</td>
<td>0x0008</td>
<td>Halt at failed tests</td>
</tr>
</tbody>
</table>

Notes:
- The default value is 0, RTS will not halt automatically.

4.1.9  LogEnable Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARIANT_BOOL</td>
<td>Read-Write</td>
<td>Controls if RTS logging is enabled</td>
</tr>
</tbody>
</table>

Notes:
- The default value is VARIANT_FALSE, logging is disabled

4.1.10 LogAppend Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARIANT_BOOL</td>
<td>Read-Write</td>
<td>Controls the logging mode - append or overwrite</td>
</tr>
</tbody>
</table>

Notes:
- The *LogAppend* property is ignored if logging is disabled
- The default value is VARIANT_TRUE, logging occurs in append mode

4.1.11 LogOptions Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>Read-Write</td>
<td>Specifies the RTS log setting options.</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Value</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT_LOG_OUTPUT</td>
<td>0x0001</td>
<td>Log ATLAS output messages.</td>
</tr>
<tr>
<td>OPT_LOG_STATEMENT</td>
<td>0x0002</td>
<td>Log ATLAS statement information.</td>
</tr>
<tr>
<td>OPT_LOG_DISPLAY</td>
<td>0x0004</td>
<td>Log display messages, generated by instrument drivers.</td>
</tr>
<tr>
<td>OPT_LOG_INFO</td>
<td>0x0008</td>
<td>Log information messages.</td>
</tr>
<tr>
<td>OPT_LOG_WARNING</td>
<td>0x0010</td>
<td>Log warning messages.</td>
</tr>
<tr>
<td>OPT_LOG_ERROR</td>
<td>0x0020</td>
<td>Log error messages.</td>
</tr>
</tbody>
</table>
Notes:
- The LogOptions property is ignored if logging is disabled
- The default value is OPT_LOG_ERROR | OPT_LOG_WARNING | OPT_LOG_BUS | OPT_LOG_OUTPUT | OPT_LOG_STATEMENT.

4.1.12 LogFileName Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSTR</td>
<td>Read-Write</td>
<td>Controls the log file name</td>
</tr>
</tbody>
</table>

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

4.1.13 NamOptions Property

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT_NAM_NORMAL</td>
<td>NAM process started with normal visibility</td>
</tr>
<tr>
<td>OPT_NAM_MINIMIZE</td>
<td>NAM process minimized at startup</td>
</tr>
<tr>
<td>OPT_NAM_HIDE</td>
<td>NAM process hidden at startup</td>
</tr>
<tr>
<td>OPT_NAM_COM</td>
<td>Enable COM NAM support</td>
</tr>
</tbody>
</table>

Notes:
- The default value is OPT_NAM_NORMAL | OPT_NAM_COM

4.1.14 SoundEvents Property

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT_SOUND_ERROR</td>
<td>“ERROR.WAV” played for RTS error conditions</td>
</tr>
<tr>
<td>OPT_SOUND_WARNING</td>
<td>“WARNING.WAV” played for RTS warning conditions</td>
</tr>
<tr>
<td>OPT_SOUND_NOGO</td>
<td>“NOGO.WAV” played for RTS NOGO conditions</td>
</tr>
<tr>
<td>OPT_SOUND_FINISH</td>
<td>“FINISH.WAV” played the TPS execution is finished</td>
</tr>
<tr>
<td>OPT_SOUND_BRKHIT</td>
<td>“BRKHIT.WAV” played the RTS encounters a breakpoint</td>
</tr>
</tbody>
</table>
“RPTHIT.WAV” played the RTS encounters a repeat section
“MIREQ.WAV” played when Manual-Intervention is enabled
“INPUT.WAV” played when RTS waits for input

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

4.1.15 DlogRecProgID Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSTR</td>
<td>Read-Write</td>
<td>Controls the data logger recorder ProgID.</td>
</tr>
</tbody>
</table>

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

4.1.16 DlogGuiProgID Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSTR</td>
<td>Read-Write</td>
<td>Controls the data logger GUI ProgID.</td>
</tr>
</tbody>
</table>

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

4.1.17 DlogServer Property

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

4.1.18 DlogRecorder Property

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

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

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

---

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

2 It is possible to change various settings related to IOSubSystem of the RTS Server. TYX advises to programmatically retrieve the Property Pages associated with the object to do so.
4.1.21 GetIOResource Method

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sName</td>
<td>BSTR</td>
<td>[in]</td>
<td>The name of resource to be retrieved.</td>
</tr>
<tr>
<td>pVal</td>
<td>IDispatch**</td>
<td>[out, retval]</td>
<td>Pointer to the resources’s IDispatch interface pointer</td>
</tr>
</tbody>
</table>

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

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

4.1.23 SwitchServerManager Property

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_HALT OPTIONS   = 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_DLOGRECROGID = 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 advises 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 advises to programmatically retrieve the Property Pages associated with the object to do so.
4.2.1 IDL Description

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

4.2.2 UML Design

![UML Diagram](RtsSettings_class_design)
4.2.3 Property Pages

Figure 22 General Property Page

Figure 23 RTS Log 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

```idl
interface IAnsiDisplay : IDispatch
{
    [propput, id(DISPID_BACKCOLOR)] HRESULT BackColor([in] OLE_COLOR clr);
    [propput, id(DISPID_BACKCOLOR)] HRESULT BackColor([out, retval] OLE_COLOR* pclr);
    [propput, id(DISPID_BORDERVISIBLE)] HRESULT BorderVisible([in] VARIANT_BOOL newVal);
    [propput, 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);
    [propput, id(DISPID_FONT)] HRESULT Font([in] IFontDisp* pFont);
    [propput, id(DISPID_FONT)] HRESULT Font([out, retval] IFontDisp** ppFont);
    [propput, id(DISPID_FORECOLOR)] HRESULT ForeColor([in] OLE_COLOR clr);
    [propput, id(DISPID_FORECOLOR)] HRESULT ForeColor([in] OLE_COLOR clr);
    [propput, id(DISPID_FORECOLOR)] HRESULT ForeColor([out, retval] OLE_COLOR* pclr);
    [propput, id(DISPID_ERRORCOLOR), helpstring("property ErrorColor")]
```
4.3.2 BackColor Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLE_COLOR</td>
<td>Read-Write</td>
<td>Controls the background color of the display.</td>
</tr>
</tbody>
</table>

4.3.3 BorderVisible Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARIANT_BOOL</td>
<td>Read-Write</td>
<td>Controls the visibility of the border of the display.</td>
</tr>
</tbody>
</table>

4.3.4 Font Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFontDisp</td>
<td>Read-Write</td>
<td>Controls the font of the display.</td>
</tr>
</tbody>
</table>

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

4.3.5 ForeColor Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLE_COLOR</td>
<td>Read-Write</td>
<td>Controls the foreground color of the display.</td>
</tr>
</tbody>
</table>

4.3.6 ErrorColor Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLE_COLOR</td>
<td>Read-Write</td>
<td>Controls the color used for display of errors.</td>
</tr>
</tbody>
</table>
### 4.3.7 WarningColor Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLE_COLOR</td>
<td>Read-Write</td>
<td>Controls the color used for display of warnings.</td>
</tr>
</tbody>
</table>

### 4.3.8 InfoColor Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLE_COLOR</td>
<td>Read-Write</td>
<td>Controls the color used for display of information.</td>
</tr>
</tbody>
</table>

### 4.3.9 Lines Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>Read-Write</td>
<td>Controls drawing of Lines in the display.</td>
</tr>
</tbody>
</table>

### 4.3.10 AutoURL Property

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>Read-Write</td>
<td>Controls display of a URL</td>
</tr>
</tbody>
</table>

### 4.3.11 DisplayText Method

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Val</td>
<td>BSTR</td>
<td>[in]</td>
<td>Displays text.</td>
</tr>
</tbody>
</table>

### 4.3.12 DisplayError Method

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Val</td>
<td>BSTR</td>
<td>[in]</td>
<td>Displays error text.</td>
</tr>
</tbody>
</table>

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

### 4.3.13 DisplayWarning Method

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Val</td>
<td>BSTR</td>
<td>[in]</td>
<td>Displays warning text.</td>
</tr>
</tbody>
</table>

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

### 4.3.14 DisplayInfo Method

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Val</td>
<td>BSTR</td>
<td>[in]</td>
<td>Displays information text.</td>
</tr>
</tbody>
</table>

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

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