CORSIM Run-Time Extension (RTE) Developer's Guide

Version 6.0

Prepared by:

ITT Industries, Inc., Systems Division
ATMS R&D and Systems Engineering Program Team
P O Box 15012
Colorado Springs, CO 80935-5012

Prepared for:

FHWA Office of Operations Research, Development and Technology
Federal Highway Administration
Turner-Fairbank Highway Research Center
6300 Georgetown Pike
McLean, Virginia 22101-2296

December 2006

NOTICE

This document is disseminated under the sponsorship of the Department of Transportation in the interest of information exchange. The United States Government assumes no liability for its contents or use thereof.

This document does not constitute a standard, specification, or regulation.

The Federal Government and McTrans are not responsible for implementation decisions (e.g., construction designs, traffic signal timings) made based on the results of analyses performed using the computer programs described herein.

Foreword

This Developer's Guide describes how to build and configure run-time extensions to the CORSIM simulation within the **T**raffic **S**oftware **I**ntegrated **S**ystem (TSIS). The McTrans Center is maintaining TSIS.

For detailed information on how to use the individual components of TSIS, please refer to the User's Guides for those individual components.

Abstract

The FHWA's **T**raffic **S**oftware **I**ntegrated **S**ystem (TSIS) is an integrated development environment that enables users to conduct traffic operations analysis. Built using component architecture, TSIS is a toolbox that contains tools that allow the user to define and manage traffic analysis projects, define traffic networks and create inputs for traffic simulation analysis, execute traffic simulation models, and interpret the results of those models.

Although TSIS comes pre-configured with a set of tools, the component architecture is open and users can add their own tools to the TSIS environment. Additionally, a user can provide extensions to the CORSIM simulation (part of the TSIS package) using the run-time extension (RTE) interface described in this document. Run-time extensions can be built to replace existing logic in CORSIM or to supplement its logic.

Notice

The TSIS 6.0 run-time extension interface logic has changed substantially from the TSIS 5.0 implementation as a result of TSIS architecture changes. Furthermore, the CORSIM data structures have significantly changed as a result of the implementation of dynamic memory within CORSIM. These changes greatly improve the efficiency of the simulation and ease the burden of creating run-time extensions. However, if you have built an RTE that works under TSIS 4.x or TSIS 5.0, it will not operate in TSIS 6.0. In this document, we provide instructions for migrating your RTE to the new TSIS 6.0 architecture. We apologize for any inconveniences these changes may cause, but in return we offer a more stable, easier to program RTE architecture.

Version 5.1.1 Release Notes

Version 5.1.1 of the TSIS run-time extension is an intermediate release that includes the CORSIM Actuated Control application programming interface (API). Section 3.5 of this document describes the functions provided by the API. Use of an API rather than the shared-memory mechanism reduces the coupling between a user's run-time extension and the underlying CORSIM data structures. With the API, CORSIM developers can modify those data structures to maintain and enhance CORSIM without causing users to re-write their run-time extensions. Not all of CORSIM's exported data structures have been replaced by APIs. As CORSIM continues to evolve, exported data structures will be replaced with API functions.

Version 6.0 Release Notes

Version 6.0 of the TSIS run-time extension is a full release that contains all of the version 5.1.1 changes. Version 6.0 extends the capability of the run-time extension interface by enabling developers to specify where in the CORSIM processing time line that an RTE function will be called. In previous versions of the interface, a single function in an RTE is called during the simulation time step, just after the vehicles have been moved at the point in the execution that CORSIM determines the state of the signals for the time step. That execution point could not be changed. With the new interface, the developer can specify multiple functions that can be called at different points within the time step and at other points in the CORSIM execution time line. Finally,

run-time extensions that were built to operate with TSIS 5.1 and TSIS 5.1.1 will continue to work with the new RTE interface in TSIS 6.0.

Contents

1 Introduction 1-1

1.1 Overview 1-1
1.2 The Run-Time Extension Interface 1-1
1.3 Required Tools 1-2

2 Building a Run-Time Extension 2-1

2.1 Creating and Compiling an RTE 2-1
2.1.1 Function Format 2-2
2.1.2 Accessing CORSIM Data Structures 2-2
2.1.3 Accessing Data through CORSIM's API Functions 2-4
2.1.4 Calling CORWin Interface Functions 2-6
2.2 Configuring an RTE in TShell 2-7
2.3 Sample RTE 2-13
2.3.1 RTE Code Files 2-13
2.3.2 CORSIM Data Structure Summary 2-14
2.3.3 Functional Description of the Example 2-17
2.4 Executing the RTE Example 2-17
2.4.1 CORSIM Input File Editing 2-17

3 Interface Reference Guide 3-1

3.1 Primary RTE Interface 3-1
3.1.1 Initialization Function 3-1
3.1.2 Main Execution Function 3-1
3.1.3 Exit Function 3-2
3.2 CORWin Interface 3-2
3.2.1 MsgBox 3-3
3.2.2 OutputString 3-3
3.2.3 RequestKeyInput 3-3
3.2.4 SendMsg 3-4
3.2.5 SetHWND 3-4
3.3 CORSIM Shared Memory 3-4
3.3.1 Scalar Variables 3-5
3.3.2 Statically Allocated Arrays 3-5
3.3.3 Dynamically Allocated Arrays 3-5
3.4 CORSIM Exported Functions 3-5
3.4.1 abortcorsim 3-6
3.4.2 assign_path 3-6
3.4.3 change_path 3-6
3.4.4 closelane 3-7
3.4.5 get_generated_vehicle_ids 3-7
3.4.6 gettxdversion 3-7
3.4.7 getvehicledata 3-8
3.4.8 openlane 3-8
3.4.9 put_path 3-9
3.4.10 put_vehicle 3-9
3.4.11 putvmsspeed 3-9
3.4.12 resize 3-10
3.4.13 WRITE_OUTPUTFILE 3-10
3.4.14 WRITE_SCREEN 3-11
3.5 Actuated Control API Exported Functions 3-12
3.5.1 GetACID 3-12
3.5.2 GetSyncReferenceTime 3-12
3.5.3 IsActuated 3-12

3.5.4 IsNewPlanPending....3-12
3.5.5 InTransition....3-13
3.5.6 GetLocalCycleTimer....3-13
3.5.7 GetCycleLength....3-13
3.5.8 GetNewCycleLength....3-14
3.5.9 SetNewCycleLength....3-14
3.5.10 GetOffset....3-14
3.5.11 GetNewOffset....3-14
3.5.12 SetNewOffset....3-15
3.5.13 GetSplits....3-15
3.5.14 GetMinSplits....3-15
3.5.15 GetNewSplits....3-16
3.5.16 SetNewSplits....3-16
3.5.17 GetTransitionMethod....3-16
3.5.18 SetTransitionMethod....3-17
3.5.19 GetTODOP....3-17
3.5.20 SetTODOP....3-18
3.5.21 GetActivePhaseInfo....3-18
3.5.22 GetTerminatedPhaseInfo....3-19
3.5.23 GetTerminationCode....3-19
3.5.24 GetDwellGreenPoint....3-20
3.5.25 GetSignal....3-20
3.5.26 SetSignal....3-21

4 Migrating Existing RTEs to TSIS 6.0.... 4-1

Glossary of Terms.... 4-1

5 Index.... 5-1

1 Introduction

1.1 Overview

The Traffic Software Integrated System (TSIS) is an integrated development environment that enables users to conduct traffic operations analysis. Although TSIS has been available since the early 1990s, it was not until 1995 that it became a Windows-based product. With the introduction of TSIS 5.0, the environment has become more integrated and supports an open component architecture that allows you to add and configure your own (or third-party) tools. TSIS 6.0 continues to use that open architecture.

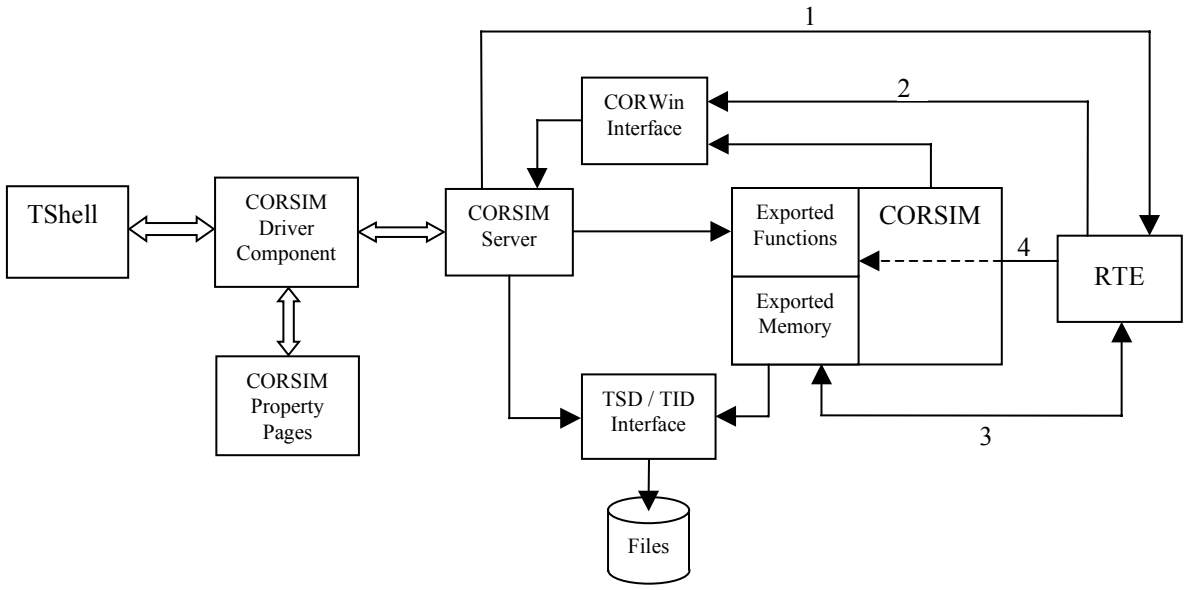
TSIS provides a mechanism by which an external application can interface directly with the CORSIM simulation tool. This type of application has become known as a CORSIM run-time extension (RTE). The original run-time extensions were tailored for signal timing studies. However, the concept has been expanded to support freeway monitoring, incident detection and ramp metering run-time extension packages.

The RTE capability enables CORSIM to operate with actual hardware in the loop. Some examples of these hardware-in-the-loop experiments include using 170, NEMA, and 2070 controllers to control the signal states at certain intersections in the simulation. In these experiments, the controller is interfaced to the simulation using a Controller Interface Device (CID) that connects to the serial interface of the computer that hosts the simulation. Another experiment involved using an advanced camera sensor and image processing software to estimate the queue state of simulated vehicles as they approach an intersection. The queue states for the approaches to the intersection are used by an adaptive control algorithm to make intelligent decisions about what the optimal signal state should be. Run-time extensions can also be used to test adaptive signal control algorithms for effectiveness before they are implemented in the field. With the rapid evolution of technology, these types of experiments are necessary, not only to assess the benefits of advanced ITS applications, but to verify their operational capability before field deployment.

This document describes how to construct a run-time extension and how to interface it with CORSIM and the TSIS graphical user interface, known as TShell. It also describes in detail each of the interfaces that TSIS provides to support RTE operation. Finally, the document presents a sample RTE.

1.2 The Run-Time Extension Interface

The following figure illustrates the operation of CORSIM within the TSIS 6.0 architecture including the interfaces that support the operation of run-time extensions. In the figure, the block arrows represent Component Object Model (COM) interfaces and the thin arrows represent standard dynamic-link library (DLL) interfaces. In this model, the CORSIM Driver Component is the TSIS tool that runs the CORSIM simulation and interfaces to the TShell user interface. This component enables the user to control CORSIM execution and output processing, and manages all user-supplied RTEs. Although the entire CORSIM architecture is shown for completeness, this document will concentrate on describing the RTE interfaces marked 1, 2, 3, and 4 in the diagram.



For each simulation time step, the CORSIM Server calls a series of functions within CORSIM to drive the simulation event loop. When an RTE is present and enabled, the CORSIM Server also calls the exported functions of the RTE based on messages it receives from CORSIM at different points in the CORSIM execution time line (refer to Section 2.1). The server also calls the RTE's initialization function during CORSIM initialization and the RTE's exit function at the end of the simulation. This interface is illustrated in the figure by the path marked "1". Although not illustrated in the figure, the architecture supports multiple RTEs.

TSIS also provides an interface (CORWin) that enables an RTE to send messages to the CORSIM Server and to send text messages to be displayed by the CORSIM Driver. This interface is illustrated in the figure by the path marked "2".

Additionally, the RTE can directly access many of the data structures in CORSIM because they have been exported as shared memory. The shared memory not only allows the RTE to extract information from the simulation but also enables the RTE to control different aspects of the simulation. The shared memory interface is illustrated in the figure by the path marked "3".

As this architecture evolves, the CORSIM shared memory interface is being replaced by APIs. Use of APIs rather than the shared-memory mechanism reduces the coupling between a user's run-time extension and the underlying CORSIM data structures. With APIs, CORSIM developers can modify those data structures to maintain and enhance CORSIM without causing users to re-write their run-time extensions.

Finally, the RTE may call directly call exported CORSIM functions via the control API (e.g., to abort the simulation) and the actuated control API. These interfaces are illustrated in the figure by the path marked "4".

Section 3 provides a detailed description of each of these four interfaces.

1.3 Required Tools

This section describes the tools needed for successful compilation of a CORSIM run-time extension. Building a CORSIM RTE requires a suitable compiler such as Microsoft Visual C++ or FORTRAN and the TSIS package, which includes all of the components shown in the previous figure (except for the RTE). Please refer to the TSIS User's Guide for information on how to obtain a copy of the TSIS package. Compilers supplied by vendors other than Microsoft can be used but the developer will have to refer to the compiler-specific documentation for information regarding linking with Microsoft-compiled libraries. All examples provided in this document were compiled with Microsoft compilers.

McTrans provides limited support for RTE development. This document, the CORSIM Data Dictionary, a debug version of the CORSIM Driver, and a sample RTE are available to all users. To take full advantage of the RTE interface, an RTE developer requires knowledge about CORSIM's data structures, which is provided by the CORSIM Data Dictionary document. The debug version of the CORSIM Driver, available from the website, enables a developer to use a debugger with the RTE. Finally, the complete source code for the RTE sample described in this document is available to all users.

2 Building a Run-Time Extension

As stated previously, a run-time extension (RTE) is a mechanism for an external application to interface with CORSIM. An RTE is compiled as a separate dynamic-link library (DLL) and must provide (export) certain functions for CORSIM (via the CORSIM Server) to call. By linking to the CORSIM libraries, an RTE has access to the data structures and functions exported by CORSIM. Installing TSIS 6.0 with the "custom" or "complete" option enables you to install the libraries required for developing an RTE.

2.1 Creating and Compiling an RTE

The RTE DLL that you develop must provide and export at least one function to enable the RTE to communicate with CORSIM. However, your RTE can export multiple functions that can be called at different points in the CORSIM execution time line. You may name the exported functions with any names you choose, but you must configure the RTE with those names as described in Section 2.2. The following table identifies and describes the points (Call Points) in the CORSIM execution time line at which CORSIM can call the exported RTE functions.

CORSIM Call Point	Description
Initialize	Called at the start of the simulation, prior to CORSIM initialization but after CORSIM has processed (read) its input (TRF) file.
PostVehicleEmit	Called each time step just after vehicles have been emitted into the network.
PreNetsimVehicle	Called each time step just prior to the point at which vehicles are moved in the NETSIM (surface-street) sub-model.
PreNetsimSignal	Called each time step just prior to the point at which signals are updated in the NETSIM (surface-street) sub-model.
PostNetsimTimestep	Called each time step at the end of NETSIM processing for the time step and prior to FRESIM processing.
PreFresimVehicle	Called each time step just prior to the point at which vehicles are moved in the FRESIM (freeway) sub-model.
PreFresimSignal	Called each time step just prior to the point at which signals (e.g., ramp meters) are updated in the FRESIM (freeway) sub-model.

CORSIM Call Point	Description
PostFresimTimestep	Called each time step at the end of FRESIM processing for the time step.
Shutdown	Called just prior to CORSIM termination.
BeginSimulation	Called after CORSIM initialization (network fill) at the start of simulation portion of the execution.
TimeStepComplete	Called each time step at the end of processing for the time step.
TimeIntervalComplete	Called each time interval at the end of processing for the time interval.
TimePeriodComplete	Called each time period at the end of processing for the time period.
SimulationComplete	Called at the end of the simulation portion of the execution and prior to shutdown.
TimePeriodValidated	Called at the end of input processing and validation for a time period, prior to initialization and prior to the start of simulation for each time period.

In addition to the call points specified in the preceding table, you may specify an exit function, which CORSIM will call once at the end of simulation.

2.1.1 Function Format

All examples provided in this document are C++ code, developed using the Microsoft Visual C++ development environment. The following code example illustrates the basic form for an RTE's main execution function named JMAIN.

```
#define DLL_EXPORT extern "C" __declspec( dllexport )

DLL_EXPORT void __stdcall JMAIN()
{
      // Code that is executed every time step should
      // be added here.
}
```

The first line enables functions for the DLL to be exported through the __declspec(dllexport) extension provided by Microsoft. The DLL_EXPORT alias is defined for convenience. The function, void JMAIN(), is declared next. Because this function is exported, the RTE library will contain a definition for the prototype function JMAIN. However, the name for the function will be decorated in a certain way depending on the type of compiler and compiler options used to create the DLL. The __stdcall calling convention is used to ensure that the function will be exported consistent with the naming decorations for the Win32 API functions, because this is what CORSIM expects. The initialization and exit functions are defined similarly.

2.1.2 Accessing CORSIM Data Structures

To access the data structures in CORSIM, you will need to link the RTE DLL with the corsim.lib library that is provided with TSIS 6.0. This library contains all of the data structures used by CORSIM to conduct the simulation. The following code example illustrates how the detector information in the CORSIM array, DTMOD, can be accessed.

```
// The following definition simplifies the importing of
// CORSIM data.
#define DLL_IMPORT extern "C" __declspec( dllimport )

// Use the following construct to obtain access to the
// DTMOD array.
#define IMXDET 7000
DLL_IMPORT struct{ int DTMOD[IMXDET]; } SIN314;
#define dtmod SIN314.DTMOD

int id = 0;
int detinfo = 0;
int type = 0;
int speed = 0;

// Get information from the array for the detector with
// an ID = 10.
id = 10;
detinfo = dtmod[id];

// The elements of the dtmod array are bit packed.
// Bits 1 through 3 contain the type of detector.
// To extract the type, "AND" detinfo with the mask
// 0x07 = 0111.
type = detinfo & 0x07;

// Bits 4 through 10 contain the vehicle's speed.  To
// extract the speed, first "AND" detinfo with the mask
// 0x03F8 = 1111111000.
speed = detinfo & 0x03F8;

// Next, right shift the value to remove the lowest 3
// bits to get the speed.
speed = speed >> 3;
```

In CORSIM, the DTMOD array is contained in the SIN314 common block (FORTRAN). In the example, SIN314 is first imported as a C structure with DTMOD as an element. Then an alias, “dtmod”, is defined as the DTMOD element of the SIN314 struct. Since arrays in C start with index 0, while FORTRAN arrays start with index 1, an index of 10 accesses the eleventh element in the FORTRAN array DTMOD. The elements in this array are bit packed, where certain bits contain the type of detector, and other bits contain the speed of the vehicle that activated the detector. The remaining lines of code illustrate how the type and speed can be extracted from the bit-packed array element.

DTMOD is a statically allocated integer array. However, the newest version of CORSIM also contains many dynamically allocated arrays. The following code example illustrates how dynamically allocated arrays are imported and accessed. The function, GetLinkCorsimId, accepts the upstream and downstream node numbers for a link and returns the internal array index that CORSIM uses to access other information about the link. The CORSIM arrays, NETSIM_LINKS_mp_DWNOD and NETSIM_LINKS_mp_UPNOD, contain the downstream and upstream node numbers for each link in the simulation. The function searches these arrays to find a link whose nodes match the specified node numbers. Notice that these arrays are bounded by the parameter, ttlnk, and it is important not to access memory beyond this bound. Also notice that since the original arrays in CORSIM are FORTRAN arrays, that the array index into nmap must be offset by –1.

```
// The following import statements are provided in Netsim.h.
DLL_IMPORT int* NETSIM_LINKS_mp_DWNOD;
DLL_IMPORT int* NETSIM_LINKS_mp_UPNOD;

DLL_IMPORT struct{ int NMAP[IMXNOD]; } SIN075;
#define nmap SIN075.NMAP

DLL_IMPORT struct{ int TTLNK; } SIN116;
#define ttlnk SIN116.TTLNK

int CNetwork::GetLinkCorsimId( int upnode, int dnnode )
{
      // Find the CORSIM link ID for the link (upnode,dnnode).
      int id = 0;
      int dnode = 0;
      int unode = 0;

      // Search through all the links in CORSIM.
      for( int i=0; i<ttlnk; i++ )
      {
            // For the ith link, get the downstream node
            // and upstream node as represented in CORSIM.
            dnode = NETSIM_LINKS_mp_DWNOD[i];
            unode = NETSIM_LINKS_mp_UPNOD[i];

            // For non-source nodes (<7000), use the nmap
            // array to map the node number in CORSIM back
            // to the user defined node number in the CORSIM
            // input file.  Use an offset of -1 (i.e. dnode-1),
            // because C arrays start at 0 and FORTRAN
            // arrays start at 1.
            if( dnode<7000) dnode = nmap[dnode-1];
            if( unode<7000) unode = nmap[unode-1];
            if( (dnode==dnnode) && (unode==upnode) )
            {
                  id = i;
            }
      }
      return id;
}
```

2.1.3 Accessing Data through CORSIM's API Functions

Some of the data in CORSIM, specifically data involving actuated control, must be accessed through the API functions described in section 3.5. These API functions have been developed to provide a safe and easy means of extracting and setting CORSIM data. This approach is much more convenient than directly accessing the internal data structures in CORSIM, reducing the need for the RTE developer to understand internal CORSIM data structures. It also helps enforce important programming constructs such as data hiding and allows the internal data structures in CORSIM to be decoupled from the RTE. The following code example shows how to set the signal state for a particular link.

```
// Declare and import the SetSignal function.  The following
// definition simplifies the importing of CORSIM functions.
#define DLL_IMPORT extern "C" __declspec( dllimport )

//   nUpnode - upstream node ID of the link
//   nDnnode - downstream node ID of the link
//   nLeft   - signal code for the left turn movement
//   nThru   - signal code for the thru movement
//   nRight  - signal code for the right turn movement
//   nDiag   - signal code for the diagonal turn movement
//
//   Signal Codes: 0 - Red
//                 1 - Yellow
//                 2 - Green
//                 3 - Permitted Green

DLL_IMPORT int __stdcall SetSignal( int nUpnode, int nDnnode,
                                    int nLeft, int nThru,
                                    int nRight, int nDiag );

void CLink::SetSignalState()
{
    int nUpnode = 15;
    int nDnnode = 14;
    int nLeft  = 2;
    int nThru  = 2;
    int nRight = 2;
    int nDiag  = 2;
    int errorCode = 0;

    errorCode = SetSignal( nUpnode, nDnnode,
                           nLeft, nThru, nRight, nDiag );

    return errorCode;
}
```

The purpose of this code is to set all the movements on the particular link with upstream node identification of 15 and downstream node identification of 14 to green. The code for green is 2 so this is the code that is passed in to the SetSignal API function. The following code example extracts the signal state for this same link.

```
// Declare and import the GetSignal function.  The following
// definition simplifies the importing of CORSIM functions.
#define DLL_IMPORT extern "C" __declspec( dllimport )

//   nUpnode - upstream node ID of the link
//   nDnnode - downstream node ID of the link
//   nLeft   - signal code for the left turn movement
//   nThru   - signal code for the thru movement
//   nRight  - signal code for the right turn movement
//   nDiag   - signal code for the diagonal turn movement
//
//   Signal Codes: 0 - Red
//                 1 - Yellow
//                 2 - Green
//                 3 - Permitted Green
```

```
DLL_IMPORT int __stdcall GetSignal( int nUpnode, int nDnnode,
                                    int* nLeft, int* nThru,
                                    int* nRight, int* nDiag );

void CLink::GetSignalState()
{
    int nUpnode = 15;
    int nDnnode = 14;
    int nLeft;
    int nThru;
    int nRight;
    int nDiag;
    int errorCode = 0;

    errorCode = GetSignal( nUpnode, nDnnode,
                           &nLeft, &nThru, &nRight, &nDiag );

    return errorCode;
}
```

Notice that the addresses of the signal code variables are passed into the function. These variables will contain the current signal states for each movement after the call to GetSignal.

2.1.4 Calling CORWin Interface Functions

TSIS provides a set of functions by which an RTE can send Windows messages to the CORSIM Driver software and text messages to the CORSIM Driver interface. To access these functions, you will need to link the RTE DLL with the CORWin.lib library that is provided with TSIS 6.0. This library contains several exported utility functions that are described in Section 3.2. The following code example illustrates how to send text messages to the CORSIM Driver. The driver will display the text messages in its window within TShell. The OutputString function is described in detail in Section 3.2.

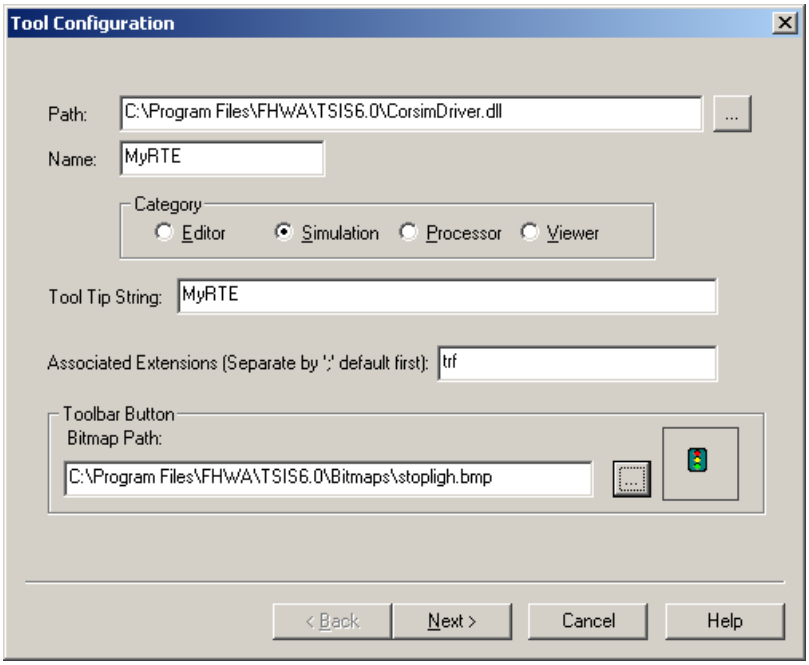
```
// Defined (in corwin.h) for convenience.
#define CORWINAPI __declspec( dllimport )

// Declare the CORWin OutputString function (provided by corwin.h).
//   str is a pointer to the character string
//   size is the length of the string excluding null terminator
//   msgCode: 0 indicates that argument 4 specifies an RGB color
//            1 CORSIM Driver-defined information message color
//            2 CORSIM Driver-defined warning message color
//            3 CORSIM Driver-defined error message color
//   color specifies the color of the string text when argument
//         3 has a value of zero.
CORWINAPI void __stdcall OutputString( char* str,
                              unsigned int size,
                              int msgCode,
                              unsigned long color );

// This is an example of an RTE initialization routine
// that sends text messages to the CORSIM Driver.  CORSIM
// calls the RTE initialization function once at the beginning
// of the simulation.
extern "C" DLL_EXPORT void __stdcall INIT()
{
      // Output a string to the CORSIM Driver that indicates
      // the start of initialization.
      char outbuffer[132];
      sprintf( outbuffer, "Starting Initialization\n" );
      OutputString( outbuffer, 132, 1, 0 );

      // Initialization code for the RTE should be
      // placed here.

      // Output a string to the CORSIM Driver that indicates
      // the end of initialization.
      sprintf( outbuffer, "Initialization Complete\n" );
      OutputString( outbuffer, 132, 1, 0 );
}
```

2.2 Configuring an RTE in TShell

This section describes the steps required to define a TSIS tool that uses a CORSIM RTE. The following description assumes that you have successfully installed TSIS 6.0 on your computer and that you have either obtained or successfully built an RTE DLL.

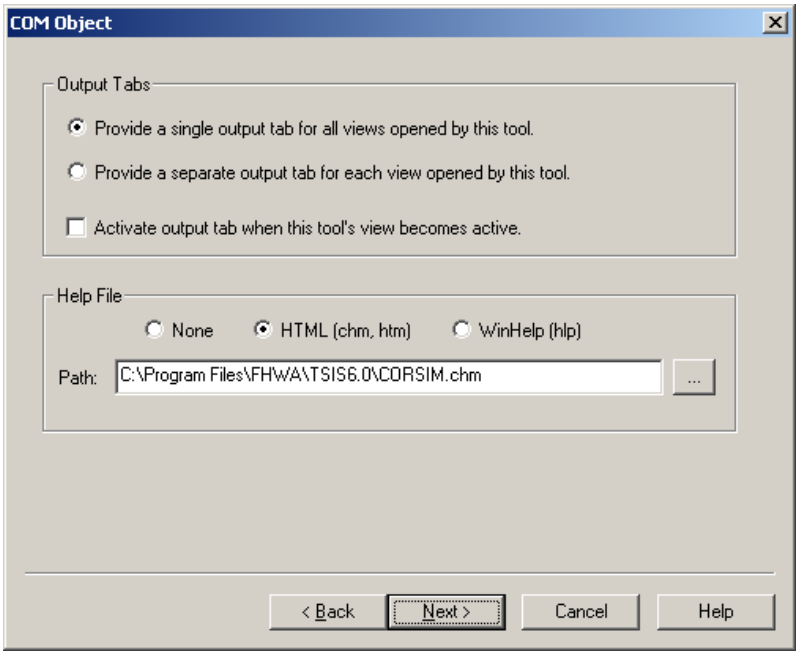
TSIS 6.0 comes automatically pre-configured with different tools, including CORSIM. However, you will need to create a new tool that executes your RTE. Please refer to the TShell User's Guide for detailed instructions on how to create TSIS tools. To begin the tool creation process, click on the **Tools** menu item and choose the **Tool Configuration** option. TShell will display the Tool Configuration dialog box that enables you to manage its set of tools. Click the **Add** button in the dialog to activate the TShell tool configuration wizard. The tool configuration wizard will step you through the process of adding a tool that uses your RTE. The following instructions illustrate how to add a CORSIM RTE named MyRTE.dll.

The following figure illustrates the first page of the tool configuration wizard.

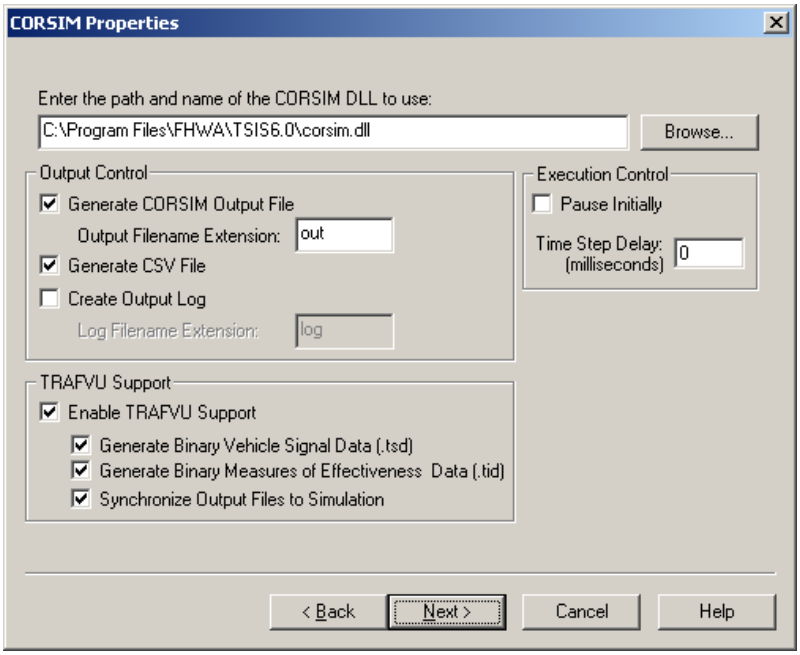


Click the browse button next to the **Path** edit box to locate the CorsimDriver or CORDebug DLL inside the TSIS directory. You must specify the CORDebug DLL if you wish to debug your RTE (e.g., to add breakpoints inside your RTE and perform other common tasks inside the debugging environment). See Section 1.3 for obtaining a copy of the CORDebug DLL, which is not part of the standard TSIS installation package. Next type in an appropriate name for your RTE tool and select the **Simulation** radio button. Under the **Associated Extensions** edit box enter the extension "trf" for CORSIM input file. Finally, use the browse button next to the **Bitmap Path** edit box to choose a bitmap that TShell will use for the button it places on its tools toolbar for your RTE tool.

When finished entering the information on this page, click the **Next** button to display the next page in the configuration wizard. The COM Object page is illustrated in the following figure.



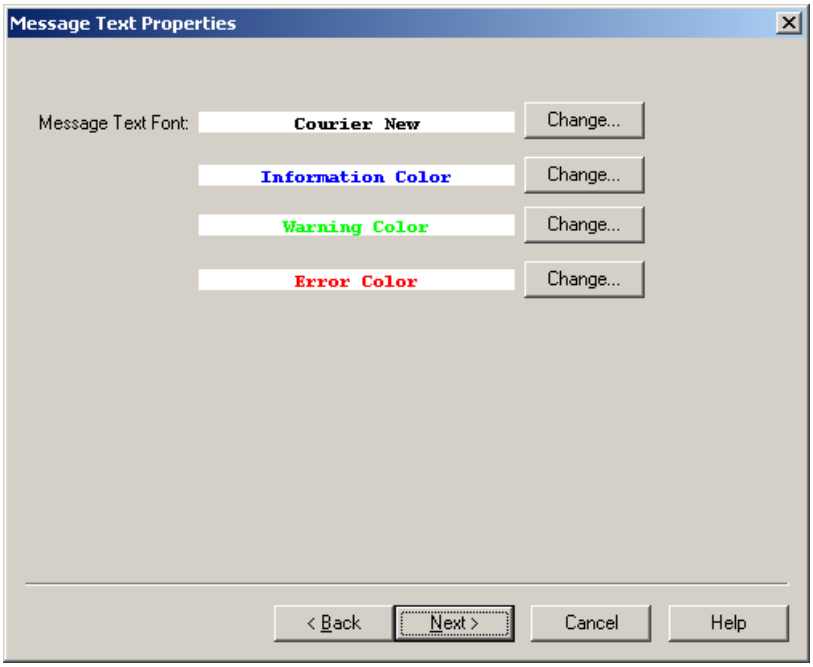
Please refer to the TShell User's Guide for information regarding the settings on this page. When finished entering the information on this page, click the **Next** button to display the next page in the configuration wizard. The CORSIM Properties page is illustrated in the following figure.



The first edit box at the top of the page should automatically contain the path and filename for the CORSIM DLL. Unless you are using a specialized version of CORSIM, the default setting for the CORSIM DLL will be sufficient. For most RTEs, the other options should be set as shown in the figure. However, this page has several useful parameters such as **Time Step Delay** and **Synchronize Output Files to Simulation**. These options are important for users who use the CORSIM simulation to perform hardware-in-the-loop experiments. If you are running a hardware-in-the-loop simulation, you should set the time delay to 1000 ms, so that CORSIM runs in real time. Checking the **Synchronize Output Files to Simulation** box ensures that output data are written to the file every time step (as opposed to having the operating system determine when to write data).

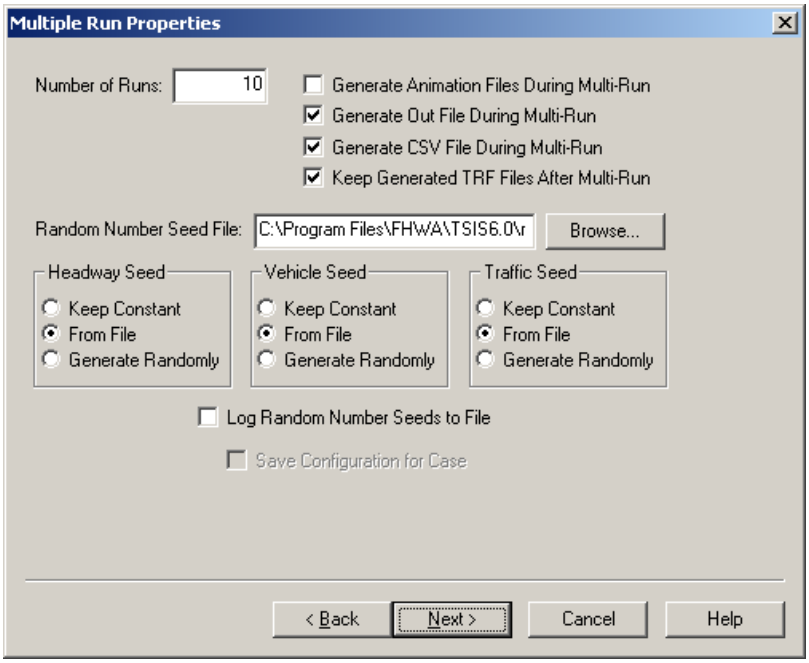
In version 5.0 of TSIS, this dialog contained a check box for enabling the RTE. With this version of TSIS, the check box was removed as the RTE is enabled simply by adding the RTE as described in the remainder of this section.

When finished entering the information on this page, click the **Next** button to display the next page in the configuration wizard. The Message Text Properties page is illustrated in the following figure.



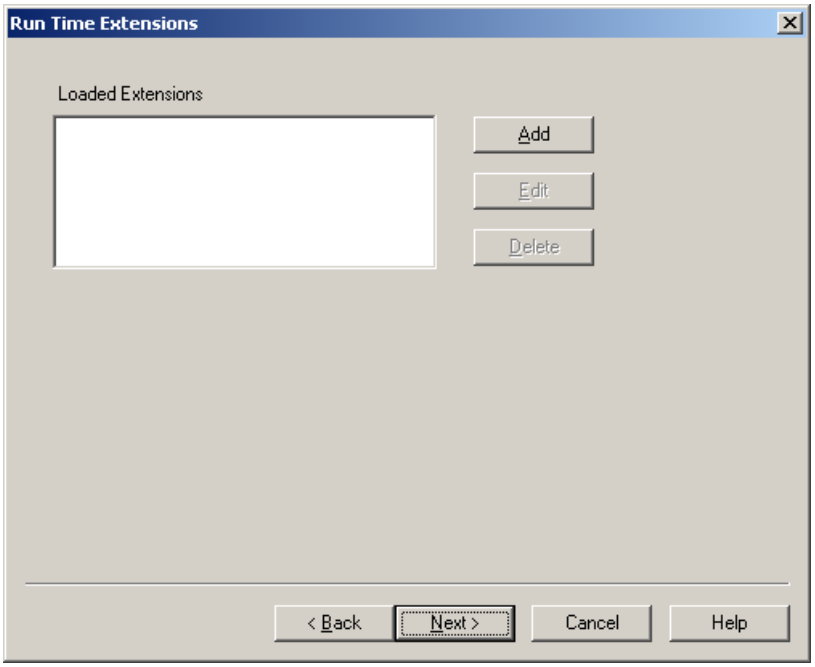
Use the **Change** buttons to modify the text color and font for the different types of messages that CORSIM writes to its document window as it executes.

When finished entering the information on this page, click the **Next** button to display the next page in the configuration wizard. The Multiple Run Properties page is illustrated in the following figure.

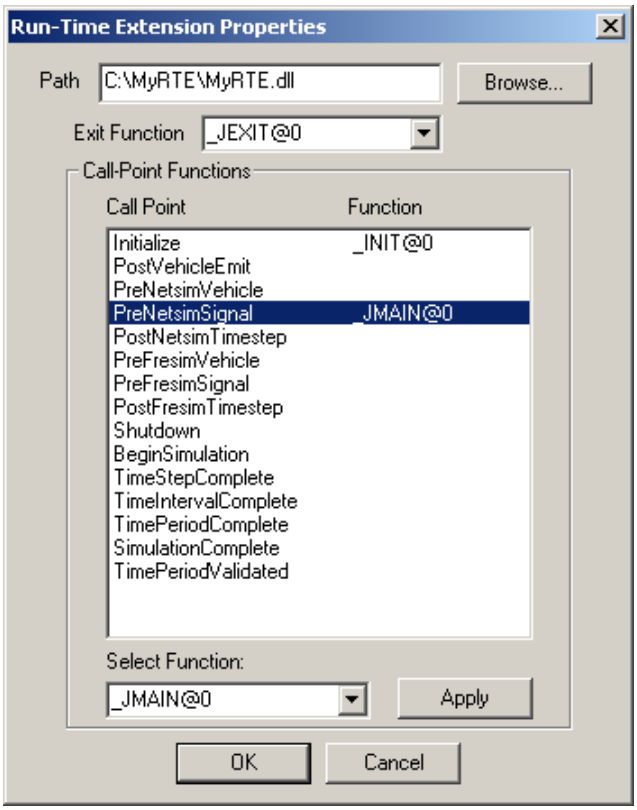


The Multiple Run Properties page enables you to control multiple runs of the simulation in order to obtain valid measures of effectiveness produced by the simulation. Please refer to the CORSIM User's Guide for details regarding the parameters on this page.

When finished entering the information on this page, click the **Next** button to display the next page in the configuration wizard. The Run Time Extensions page is illustrated in the following figure.

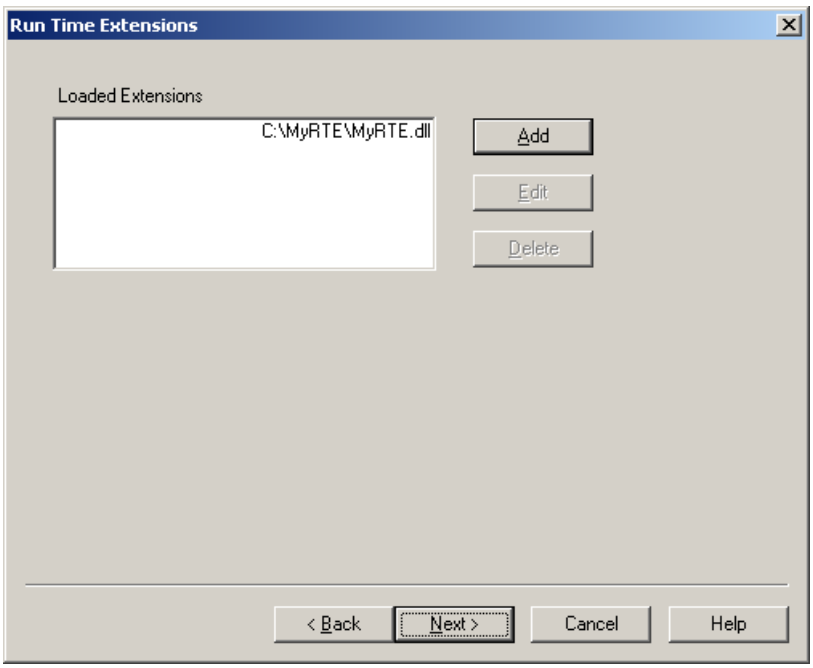


This property page enables you to add RTEs to the CORSIM simulation tool you are creating. To add your RTE, press the **Add** button on this page. When you press the **Add** button, the wizard displays the dialog illustrated in the following figure.

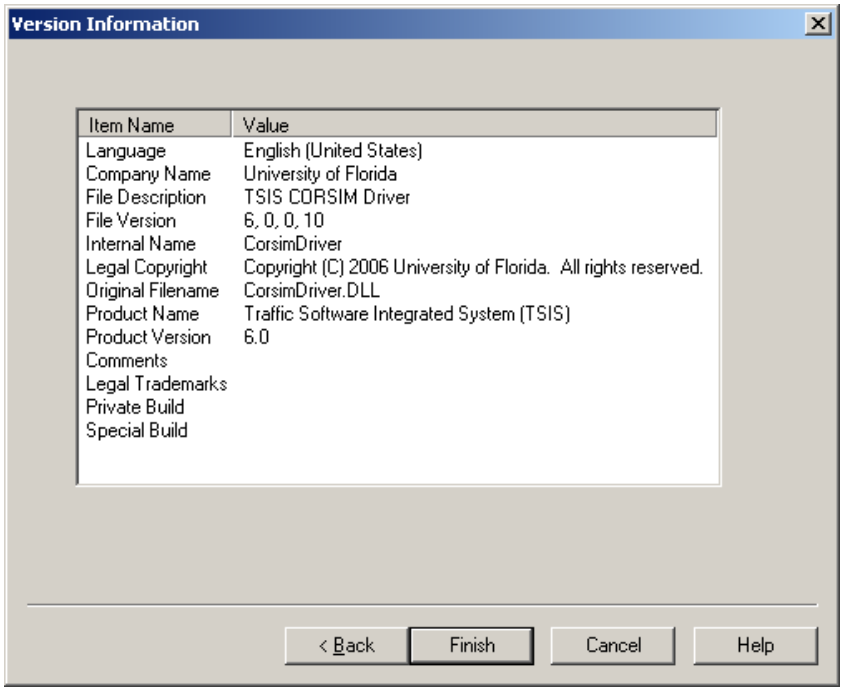


Use the Browse button to specify the name of your RTE DLL. In the edit boxes provided in this dialog, specify the names of the initialization, main execution, and exit functions that you implemented in your RTE and as described in Section 2.1. If you do not enter a name in one of these edit boxes, CORSIM will not call the function. Once you have entered all of the required information, press the **OK** button to add the RTE and return you to the Run Time Extensions page.

If the wizard successfully added the RTE, it will appear in the Run Time Extensions page as illustrated in the following figure. All successfully added RTEs are automatically enabled and CORSIM will call them in the order they appear in the dialog.



When you are finished adding RTEs and reviewing the information on this page, click the **Next** button to display the next page in the configuration wizard. The Version Information page is illustrated in the following figure.



The Version Information page presents version information for the primary tool DLL (the tool DLL in the **Path** edit box on the first page of the wizard). You are not required to enter any data on this page. To complete the tool configuration process, click the **Finish** button on this page. The newly created tool will now appear in the list on the Tool Configuration dialog. Press the **OK** button on the dialog to complete the process (the new tool is not added until this dialog is dismissed with the **OK** button). The tool is now ready to use.

2.3 Sample RTE

The sample RTE was developed by ITT Industries, Inc., Systems Division as a product of research into adaptive signal control algorithms. For this type of application, the signal states for certain intersections are controlled by the RTE rather than by CORSIM's internal signal control logic. The adaptive signal control algorithm reads the detector data from CORSIM's exported data structures and writes the signal states for intersections it is controlling. Via the CORSIM input file, the user specifies which intersections are being controlled by the RTE (see Section 2.4.1) and CORSIM will not compute their signal state. Instead, it will use the signal state information for those intersections as set by the RTE.

To execute and test the RTE, ITT developed a CORSIM input file, SampleRTE.trf. This file specifies the control for the intersections at nodes 13 and 14 will be provided by the RTE. At those intersections, the RTE will imitate the fixed timing plan specified in the file for intersections 13 and 14. Other more elaborate signal control algorithms could also be tested using the same basic setup described in this example. However, this example provides a starting point for the user and will control the signals in the SampleRTE.trf network as described above. The user may want to modify this RTE to perform more sophisticated types of signal control, but consistency between the CORSIM libraries, the RTE, and the tool configured in TSIS must be maintained.

2.3.1 RTE Code Files

The following table identifies and briefly describes the project files and code files used to create and compile the sample RTE. The RTE was created and compiled with Microsoft Visual C++. The name of the sample RTE project is Interfac.

File Name	Description
Interfac.dll	Interfac RTE DLL
Interfac.dsp	Interfac project settings file
Interfac.dsw	Interfac project workspace file
BinarySequence.cpp BinarySequence.h	Integer ID collection class files.
Detector.cpp Detector.h	Detector information container class files.
Integer.cpp Integer.h	Used in storing data within the BinarySequence.
Lane.cpp Lane.h	Lane information container class files.
Link.cpp Link.h	Link information and management class files.
Netsim.h	Header file that contains the export definitions to make the CORSIM variables available to the C/C++ Run-time extension.
Network.cpp Network.h	Network class files. Maintains the node and link class object collections.
Node.cpp Node.h	Node information container class files.
SignalState.cpp SignalState.h	Signal code information collection class files.
Upcntrl.cpp Upcntrl.h	C interfacing code files. These files contain the declarations and definitions of the interface functions: UPINIT, UPCNTRL, UPEXIT.

2.3.2 CORSIM Data Structure Summary

Before we can discuss the specifics of this example, we must describe some of the data structures in CORSIM and how these data structures can be accessed. The following table describes some of the data structures in CORSIM needed for testing adaptive control algorithms. The CORSIM Data Dictionary, available on the TSIS website, provides complete descriptions of all the data structures in CORSIM. The table describes the variables that are identified in the first column and identifies the mechanisms used to access them.

Variable name	Description	Data Access
NETSIM_LINKS_mp_IMXLNK	Maximum allowable number of links in a network	Imported from the CORSIM.dll as an integer
IMXNOD	Maximum allowable number of nodes in a network	Defined using a #define statement. Currently it is 8999

Variable name	Description	Data Access
IMXDET	Maximum allowable number of detectors in a network	Defined using a #define statement. Currently it is 7000
YINIT	Flag that indicates if the simulation reached equilibrium	Imported from the CORSIM.dll through the common GLR091
NETSIM_LINKS_mp_DWNOD	Downstream node number of link IL, where IL is the CORSIM link ID	Imported as a dynamically allocated integer array from the CORSIM.dll
NETSIM_LINKS_mp_UPNOD	Upstream node number of link IL, where IL is the CORSIM link ID	Imported as a dynamically allocated integer array from the CORSIM.dll
NMAP[IN]	User-specified node number that corresponds to the internally-assigned NETSIM subnetwork node number, IN	Imported from the CORSIM.dll through the common SIN075
CLOCK	Current time since start of the simulation in seconds	Imported from the CORSIM.dll through the common SIN104
TTLNK	Total number of links in subnetwork	Imported from the CORSIM.dll through the common SIN116
DTLNK[DT]	Link Number surveillance detector DT is on	Imported from the CORSIM.dll through the common SIN700
DTLEN(DT)	Length of Detector DT in tenths of a foot	Imported from the CORSIM.dll through the common SIN313
DTMOD(DT)	Bit-packed detector array Bits 1-3: detector type (0=Presence, 1=Passage) Bits 4-10: speed of vehicle when passing the passage detector Bits 11-23: vehicle count since beginning of simulation	Imported from the CORSIM.dll through the common SIN314
DTNLNK(DT)	Detector identification number of the next detector on the same link as detector DT	Imported from the CORSIM.dll through the common SIN308
DTPOS(DT)	Distance between the detector's downstream edge and the downstream stop-bar, in tenths-of-a-foot	Imported from the CORSIM.dll through the common SIN312
NETSIM_LINKS_mp_DTFLNK	Detector identification number of the first detector on the referenced link, IL	Imported as a dynamically allocated integer array from the CORSIM.dll

Variable name	Description	Data Access
DETON(DT)	Bit-packed detector "on" array: Bit 1: (0,1) if detector was (on,off) for first 0.1 second Bit 2: (0,1) if detector was (on,off) for second 0.1 second Bit 3: (0,1) if detector was (on,off) for third 0.1 second Bit 4: (0,1) if detector was (on,off) for forth 0.1 second Bit 5: (0,1) if detector was (on,off) for fifth 0.1 second Bit 6: (0,1) if detector was (on,off) for sixth 0.1 second Bit 7: (0,1) if detector was (on,off) for seventh 0.1 second Bit 8: (0,1) if detector was (on,off) for eighth 0.1 second Bit 9: (0,1) if detector was (on,off) for ninth 0.1 second Bit 10: (0,1) if detector was (on,off) for tenth 0.1 second	Imported from the CORSIM.dll through the common SIN070
TOTDET	Number of surveillance detectors specified for this subnetwork. The data for each detector is contained in arrays DTCTR1, DTCTR2 and DTCTR3.	Imported from the CORSIM.dll through the common SIN109
linfname	Input filename for the simulation	Imported from the CORSIM.dll through the common LIOFILES
loutfname	Output filename for the simulation	Imported from the CORSIM.dll through the common LIOFILES
linflen	length of the input filename	Imported from the CORSIM.dll through the common LIOFILES
loutflen	length of the output filename	Imported from the CORSIM.dll through the common LIOFILES

The information for surveillance detectors is stored in the variables DETON or DTMOD. These data structures must be imported by the RTE if they are to be accessed. For applications developed using Microsoft Visual C++, this can be accomplished using the dllimport statements as shown in the Netsim.h file. If you are not developing your RTE using Microsoft Visual C++, then a different mechanism will be needed to import the above data structures into the RTE. In this event, you will need to consult the documentation provided with the development tool you are using.

2.3.3 Functional Description of the Example

When a CORSIM tool is configured with an RTE, it will call the RTE exported functions as described in Section 2.1. At the start of the simulation, CORSIM will call the initialization function. This function in the interfac.dll, named INIT, reads the CORSIM input file and creates such things as the link objects, node objects, and detector objects used by the RTE. It also reads the pre-timed control specified on cards 35 and 36 for nodes 13 and 14. Using this information, it creates a list of signal state objects. The INIT function is called only once at the beginning of the simulation.

CORSIM calls the main execution function, named JMAIN, every time step throughout the simulation. This function uses the objects that were created by the INIT function to control the signal states at intersections 13 and 14 of the SampleRTE network. Specifically, JMAIN calls the SetSignalState() method of the CLink class. This method gets the next signal state from the list of signal state objects and then calls the exported API function SetSignal to set the signal state for each of the approach links to the node.

At the end of the simulation, the CORSIM Server calls the RTE exit function, JEXIT. This function deletes all the objects that were created by the RTE.

There are several useful functions provided in this example. The function in the CNetwork class named GetLinkCorsimId(int upnode, int dnnode) illustrates how to find the actual CORSIM link ID for a particular link with a specified upstream node, upnode and downstream node, dnnode. The function, GetDetectorCorsimId(CDetector* pDetector), shows how to find the CORSIM ID for a specified detector.

Previous versions of this example RTE could only handle signal state transitions from North/South all green to East/West all green. Recent updates to the example allow for different transitions. For example, if the CORSIM input file contained signal phasing for North/South left turners and North/South all green the RTE will now accommodate those transitions. In fact, the example RTE now handles transitions between any of the following record type 36 control codes: 1, 2, 3, 4, 7, 8, 9.

2.4 Executing the RTE Example

To execute this example RTE, configure the Interfac RTE according to the instructions provided in Section 2.2, where the names of the RTE-provided functions are INIT, JMAIN, and JEXIT. The user is cautioned that this RTE has not been tested with any CORSIM input files other than the one provided in the example. The following subsection describes how to configure the CORSIM input file to run the example.

2.4.1 CORSIM Input File Editing

To get CORSIM to use the signal states set by the RTE, you must indicate which intersections will be controlled by the RTE by setting a value in the type 36 or type 43 records. Regardless of whether the node is to be under external control or CORSIM control, it must be specified as either pre-timed or actuated. If a node to be controlled by the RTE was originally specified as having pre-timed control, you should enter a 2 in column 77 of card type 36. This value flags CORSIM that the external algorithm will control the node. Similarly if the node was originally specified as having actuated or semi-actuated control then column 77 of card type 43 should be set to 2. The following text, taken from the SampleRTE.trf file, illustrates the 36 cards with column 77 set to the appropriate code specifying external control for nodes 13 and 14.

```
 7 111                                                                  36
 8 111                                                                  36
13 1122 0022 2211 2200                                                2 36
14 1122 0022 2211 2200                                                2 36
15 11                                                                  36
```

Note: pre-timed nodes that are assigned to be under external control will be handled in the same manner as actuated controls by CORSIM. This is because the simulation no longer has control or knowledge of the timing scheme.

3 Interface Reference Guide

3.1 Primary RTE Interface

This section provides descriptions of the RTE functions to be provided by the RTE developer and called by CORSIM. Because these functions implement the code provided by the RTE developer, the descriptions cover only the basic purpose of the functions and when they are called by CORSIM. Because the RTE developer provides the name of the function, the following descriptions are identified by the primary purpose of the function rather than by actual function name.

3.1.1 Initialization Function

CORSIM calls this function once at the start of the simulation. Specifically, CORSIM calls the initialization function after it has processed the input data, just prior to the start of the first time step in the run-to-equilibrium phase of the simulation. The RTE should use this function to perform all required initialization tasks.

Returns

This function returns no value.

Parameters

This function has no parameters.

C Declaration Example

```
#define DLL_EXPORT __declspec( dllexport )
DLL_EXPORT void __stdcall INIT()
```

C++ Declaration Example

```
#define DLL_EXPORT extern "C" __declspec( dllexport )
DLL_EXPORT void __stdcall INIT()
```

FORTRAN Declaration Example

```
INTEGER*4 FUNCTION INIT[DLLEXPORT, STDCALL]()
```

3.1.2 Main Execution Function

CORSIM calls this function every time step during the simulation. The point at which CORSIM calls the main execution function during a simulation time step is fixed inside CORSIM and cannot be changed. In loose

terms, the function is called after the vehicles have been moved at the point in the execution that CORSIM normally determines the state of the signals for the time step.

Returns

This function returns no value.

Parameters

This function has no parameters.

C Declaration Example

```
#define DLL_EXPORT __declspec( dllexport )
DLL_EXPORT void __stdcall MAIN_RTE()
```

C++ Declaration Example

```
#define DLL_EXPORT extern "C" __declspec( dllexport )
DLL_EXPORT void __stdcall MAIN_RTE()
```

FORTRAN Declaration Example

```
INTEGER*4 FUNCTION MAIN_RTE[DLLEXPORT, STDCALL]()
```

3.1.3 Exit Function

The CORSIM Server calls the exit function once at the end of simulation. The call occurs after CORSIM has completed all of its processing and just prior to the server unloading the DLLs (RTE and CORSIM). The RTE should use this function to perform all clean up tasks.

Returns

This function returns no value.

Parameters

This function has no parameters.

C Declaration Example

```
#define DLL_EXPORT __declspec( dllexport )
DLL_EXPORT void __stdcall EXIT()
```

C++ Declaration Example

```
#define DLL_EXPORT extern "C" __declspec( dllexport )
DLL_EXPORT void __stdcall EXIT()
```

FORTRAN Declaration Example

```
INTEGER*4 FUNCTION EXIT[DLLEXPORT, STDCALL]()
```

3.2 CORWin Interface

This section provides detailed descriptions of the functions available in the CORWin interface. The CORWin interface enables CORSIM to communicate directly with the CORSIM Server and indirectly with the CORSIM Driver and TShell in a Windows operating system environment. This interface enables an RTE to send Windows messages to the server and text messages to the CORSIM Driver. Functions are listed in alphabetical order.

3.2.1 MsgBox

This function displays a Windows pop-up message box dialog with an "OK" button, a specified title, and a specified message.

Returns

This function returns no value.

Parameters

1. char* message - pointer to the message to be displayed in the message box.
2. unsigned int msize - the length (number of characters) of the message excluding the null terminator.
3. char* title - pointer to the title to be used by the message box.
4. unsigned int tsize - the length (number of characters) of the title excluding the null terminator.

Example

```
char szTitle[] = "RTE Error";
char szMessage[] = "RTE initialization failed.";
MsgBox( szMessage, strlen(szMessage), szTitle, strlen(szTitle) );
```

3.2.2 OutputString

This function displays the specified string in the CORSIM Driver window within TShell.

Returns

This function returns no value.

Parameters

1. char* string - pointer to the string to be displayed.
2. unsigned int size - the length (number of characters) of the string excluding the null terminator.
3. int msgCode - a code that indicates the type (color) of the message:

   SIM_COLOR_RGB - Use the RGB value specified in the fourth argument to define the color.

   SIM_COLOR_INFO - Use the information message color defined in the CORSIM Driver.

   SIM_COLOR_WARNING - Use the warning message color defined in the CORSIM Driver.

   SIM_COLOR_ERROR - Use the error message color defined in the CORSIM Driver.
4. unsigned long color - a user-specified color in RGB format.

Example

```
OutputString( "Starting Initialization…", 26, SIM__COLOR_INFO, 0L );

OutputString( "Detector Counts", 26, SIM__COLOR_RGB, 0x00000080 );
```

3.2.3 RequestKeyInput

This function is used by CORSIM to display an input dialog that prompts the user for a string. This function is not useful for an RTE, but is included for completeness.

Returns

This function returns no value.

Parameters

1. char* prompt - pointer to the prompt string to be displayed in the message box.
2. unsigned int msize - the length (number of characters) of the prompt string excluding the null terminator.

Example

```
char szPrompt[] = "Enter the name of the file:";
RequestKeyInput( szPrompt, strlen(szPrompt) );
```

3.2.4 SendMsg

This function is used by CORSIM to send control and status messages to the CORSIM Driver. This function is not useful for an RTE, but is included for completeness.

Returns

This function returns no value.

Parameters

1. int msgID - simulation message code.
2. int msgCode - simulation status message code

Example

```
SendMsg( SIM_STATUS, SIM_LINKCOMPLETE );
```

3.2.5 SetHWND

This function is used by the CORSIM Server to establish a communications channel between CORSIM and the server. It is not useful for an RTE, but is included for completeness. WARNING: Calling this function will disable the CORSIM Driver tool.

Returns

0 = handle was not set

1 = handle was set

Parameters

1. HWND hWnd - window (Windows) handle

Example

```
SetHWND( hWnd );
```

3.3 CORSIM Shared Memory

CORSIM provides access to most of its input data and other data generated during a simulation run. The CORSIM Data Dictionary identifies all data that CORSIM exports. The data items CORSIM exports fall into one of three general categories: scalar (non-array) variables, statically allocated arrays, and dynamically allocated arrays. Methods for importing the different types of items are described in the following subsections. To simplify the import of data, most programmers use the following constructs:

```
// The following definition simplifies the importing of
// CORSIM data into a C++ source file.
#define DLL_IMPORT extern "C" __declspec( dllimport )

// The following definition simplifies the importing of
// CORSIM data into a C source file.
#define DLL_IMPORT extern __declspec( dllimport )
```

3.3.1 Scalar Variables

CORSIM exports scalar variables both directly and using FORTRAN common blocks. To import a scalar variable that is directly exported, use the following construct:

```
DLL_IMPORT int IMXGLK;
```

To import a scalar variable that is exported via a FORTRAN common block, use the following construct:

```
DLL_IMPORT struct{ int TTLNK; } SIN116;
#define ttlnk SIN116.TTLNK
```

The define statement is not required, but is used to simplify the access of the imported variable. Note that a common block may include multiple variables. Use the "." notation to access individual members of the common block.

3.3.2 Statically Allocated Arrays

CORSIM exports statically allocated arrays using FORTRAN common blocks. The size (dimension) of a statically allocated array is specified in the code and is fixed to that value during run time. To import an item of this type, use the following construct:

```
#define IMXDET 7000
DLL_IMPORT struct{ int DTMOD[IMXDET]; } SIN314;
#define dtmod SIN314.DTMOD
```

The final define statement is not required, but is used to simplify the access of the imported array.

3.3.3 Dynamically Allocated Arrays

CORSIM exports dynamically allocated arrays directly. The size (dimension) of a dynamically allocated array is defined during run time and can change during the simulation. CORSIM exports scalar variables that provide the current size of dynamically allocated arrays. It is important for the RTE developer to be aware of the size of dynamically allocated arrays and to not access elements outside the array. To import an item of this type, use the following construct:

```
DLL_IMPORT int* NETSIM_LINKS_mp_DWNOD;
#define dwnod NETSIM_LINKS_mp_DWNOD
```

The define statement is not required, but is used to simplify the access of the imported array.

3.4 CORSIM Exported Functions

The CORSIM DLL exports several functions that provide execution control, path-based vehicle control and data access. Not all of the functions are of use to an RTE. The execution control functions are provided so that the CORSIM Server can execute the simulation. The RTE should not call these functions, with the exception of the abortcorsim function. Calling any other execution control function will cause unpredictable results. The following list identifies the execution control functions:

- abortcorsim
- runtoequilibrium
- setinputname
- setioflags
- setkbstr
- setoutputname
- shutdown
- simulate
- startup

Because the execution control functions (with the exception of abortcorsim) should not be called by an RTE, they are not documented here. All other functions exported by CORSIM are described in alphabetical order. C language examples are provided for calling the functions.

3.4.1 abortcorsim

This function terminates a CORSIM simulation run and can be used by an RTE to safely terminate the active simulation.

Returns

This function returns no value.

Parameters

This function has no parameters.

Example

```
abortcorsim();
```

3.4.2 assign_path

This function assigns a new path to a newly generated vehicle.

Returns

0 = success

1 = failure - could not find specified vehicle

2 = failure - GetCode could not determine the turn code (NETSIM) or find destination link (FRESIM)

Parameters

1. int IVG - global ID of vehicle to assign.
2. int PATHID - new path ID.

Example

```
assign_path( 322, 5 );
```

3.4.3 change_path

This function re-routes a vehicle by giving it a new path ID.

Returns

0 = success

1 = failure - could not find specified vehicle

Parameters

1. int IVG - global ID of vehicle to re-route.

2. int PATHID - new path ID.

Example

```
change_path( 322, 5 );
```

3.4.4 closelane

This function closes the specified lane on the specified link. A lane on a link that has only one full lane cannot be closed.

Returns

0 = success

1 = failure - could not close the specified lane

Parameters

1. int UserUpNode - upstream node of the link.
2. int UserDwNode - downstream node of the link.
3. int Lane - number of lane to be closed.

Example

```
closelane( 15, 16, 2 );
```

3.4.5 get_generated_vehicle_ids

This function returns the first and last global IDs for the vehicles that were generated in the current time step.

Returns

0 = success

1 = failure

Parameters

1. int* ID1 - global ID of the first vehicle generated in the current time step.
2. int* ID2 - global ID of the last vehicle generated in the current time step.

Example

```
int* ID1;
int* ID2;
get_generated_vehicle_ids( ID1, ID2 );
```

3.4.6 gettxdversion

This function returns the current version ID for the animation data files.

Returns

This function returns the current version ID for the animation data files.

Parameters

This function has no parameters.

Example

```
int versionID = gettxdversion();
```

3.4.7 getvehicledata

This function gets the current data for the specified vehicle. The data are set in shared memory variables.

Returns

This function returns no value. However, the function places vehicle data in the following shared memory variables:

VEHICLE_DATA_MODULE_mp_ACCELERATION - vehicle's current acceleration in feet/second/second

VEHICLE_DATA_MODULE_mp_DESTINATIONFORVEH - ID of the vehicle's current destination node

VEHICLE_DATA_MODULE_mp_DOWNSTREAMNODEOFCURRENTLINK - downstream node ID of the link on which the vehicle is currently traveling

VEHICLE_DATA_MODULE_mp_DRIVERTYPE - driver type associated with the vehicle (1-10)

VEHICLE_DATA_MODULE_mp_ENTERNETWORKTIME - time at which the vehicle entered the network (seconds from simulation start)

VEHICLE_DATA_MODULE_mp_LANEID - ID of the lane in which the vehicle is currently traveling

VEHICLE_DATA_MODULE_mp_ORIGINFORVEH - ID of the vehicle's origin node

VEHICLE_DATA_MODULE_mp_SPEED - vehicle's current speed in feet/second

VEHICLE_DATA_MODULE_mp_UPSTREAMNODEOFCURRENTLINK - upstream node ID of the link on which the vehicle is currently traveling

VEHICLE_DATA_MODULE_mp_VEHICLEFLEET - vehicle's fleet type

VEHICLE_DATA_MODULE_mp_VEHICLEPATHID - ID of the current path on which the vehicle is traveling

VEHICLE_DATA_MODULE_mp_VEHICLETYPE - vehicle type code

Parameters

1. int VehicleID - global ID for the vehicle for which information is requested.

Example

```
getvehicledata( 345 );
```

3.4.8 openlane

This function opens the specified lane on the specified link that was previously closed by a call to closelane.

Returns

0 = success

1 = failure - could not open the specified lane

Parameters

1. int UserUpNode - upstream node of the link.
2. int UserDwNode - downstream node of the link.
3. int Lane - number of lane to be opened.

Example

```
openlane( 15, 16, 2 );
```

3.4.9 put_path

This function inserts a path into CORSIM's array of paths. The array of nodes that define the path is a shared memory variable named PATH_MOD_mp_PATH. To use this function, load the PATH_MOD_mp_PATH array with the desired nodes and then call the function.

Returns

0 = success

1 = failure

Parameters

1. int pathID - ID for the new path.
2. int numNodes - number of nodes in the new path.

Example

```
PATH_MOD_mp_PATH[0] = 8001;
PATH_MOD_mp_PATH[1] = 1;
PATH_MOD_mp_PATH[2] = 5;
PATH_MOD_mp_PATH[3] = 11;
PATH_MOD_mp_PATH[4] = 8005;
put_path( 15, 5 );
```

3.4.10 put_vehicle

This function inserts a vehicle into CORSIM on the specified path.

Returns

0 = failure

> 0 = success - global vehicle ID

Parameters

1. int time – time in seconds when vehicle will enter the network..
2. int enode - entry node.
3. int pathID - ID of path onto which the vehicle will be inserted.
4. int driver - driver code.
5. int fleet - fleet code.
6. int vtype - vehicle type code.
7. int VMScomp - VMS compliance factor, 0=not compliant, 1 through 10 indicates degree of compliance to VMS systems.
8. int probe - 0 = not probe vehicle; 1= probe vehicle.

Example

```
put_vehicle( 100, 8002, 15, 4, 1, 4, 5, 0 );
```

3.4.11 putvmsspeed

This function sets the variable message sign (VMS) speed to the specified value for the specified link.

Returns

0 = success

1 = failure

Parameters

1. int UpNode - upstream node ID of the link.
2. int DnNode - downstream node ID of the link.
3. int VMSspeed - VMS speed in miles/hour.

Example

```
putvmsspeed( 12, 13, 35 );
```

3.4.12 resize

This function enables the caller to dynamically increase the allowable number of paths and nodes per path.

Returns

0 = success

1 = failure

Parameters

1. int paths - new number of allowable paths.
2. int nodes - new number of nodes per path.

Example

```
resize( 20, 35 );
```

3.4.13 WRITE_OUTPUTFILE

This function enables an RTE to write text to the CORSIM output file. Because the implementation of the function uses a fixed 2047 character FORTRAN string, extra steps must be taken to ensure the string is properly formatted when calling this function from C or C++. The examples illustrate how to properly use this function.

Returns

0 = success

1 = failure

Parameters

C or C++

1. char* string - pointer to a 2047 character string to write to the file. The unused part of the string must be filled with blanks and the string does not need to be null terminated.
2. int size - length of the string; must be set to 2047.

FORTRAN

1. CHARACTER*2047 STRING - string dimensioned to 2047 characters.

Example

C or C++

```
// Declare and fill the string with blanks.
char string[2047];
memset( string, 32, sizeof(string) );
```

```
// Copy the desired text into the string.  Because strcpy adds
// a terminating null, replace the null with a blank.
strcpy( string, "Place this string in the output file." );
string[strlen(string)] = ' ';

// Call the function to write the string to the output file.
WRITE_OUTPUTFILE( string, sizeof(string) );
```

FORTRAN

```
C     Set the string and call the function.
      CHARACTER*2047 STRING
      STRING = 'Place this string in the output file.'
      WRITE_OUTPUTFILE( STRING )
```

3.4.14 WRITE_SCREEN

This function displays the specified string in the CORSIM Driver window within TShell. Because the implementation of the function uses a fixed 2047 character FORTRAN string, extra steps must be taken to ensure the string is properly formatted when calling this function from C or C++. The examples illustrate how to properly use this function. NOTE: if you linked your RTE with the CORWin library, you should use the OutputString function in the CORWin library rather than this function.

Returns

0 = success

1 = failure

Parameters

C or C++

1. char* string - pointer to a 2047 character string to display. The unused part of the string must be filled with blanks and the string does not need to be null terminated.
2. int size - length of the string; must be set to 2047.

FORTRAN

1. CHARACTER*2047 STRING - string dimensioned to 2047 characters.

Example

C or C++

```
// Declare and fill the string with blanks.
char string[2047];
memset( string, 32, sizeof(string) );

// Copy the desired text into the string.  Because strcpy adds
// a terminating null, replace the null with a blank.
strcpy( string, "Display this string in the CORSIM Driver." );
string[strlen(string)] = ' ';

// Call the function to display the string.
WRITE_SCREEN( string, sizeof(string) );
```

FORTRAN

```
C     Set the string and call the function.
      CHARACTER*2047 STRING
      STRING = 'Display this string in the CORSIM Driver.'
      WRITE_SCREEN( STRING )
```

3.5 Actuated Control API Exported Functions

The CORSIM DLL exports several functions that provide access to the actuated control model in the CORSIM simulation. This section describes those functions. Each function description includes a C/C++ example of the function usage.

3.5.1 GetACID

This function returns the actuated controller ID for the specified node. If the node is not under actuated control, the function returns 0.

Returns

> 0 - actuated controller ID

= 0 - specified node is not under actuated control

Parameters

1. int nNodeID - ID of the node at which the controller is located.

Example

```
int ID = GetACID( 1 );
```

3.5.2 GetSyncReferenceTime

This function returns the system sync reference time in seconds since midnight.

Returns

> 0 - sync reference time in seconds since midnight

= -1 - no sync reference time was specified

Parameters

This function has no parameters.

Example

```
int time = GetSyncReferenceTime();
```

3.5.3 IsActuated

This function indicates if the specified node is under actuated control.

Returns

= 0 - specified node is not under actuated control

= 1 - specified node is under actuated control

Parameters

1. int nNodeID - ID of the node at which the controller is located.

Example

```
int i = IsActuated( 1 );
```

3.5.4 IsNewPlanPending

This function indicates if new parameters are pending update by the controller at the specified node.

Returns

= 0 - no updates pending

= 1 - updates are pending

= -1 - specified node is not under actuated control

Parameters

1. int nNodeID - ID of the node at which the controller is located.

Example

```
int i = IsNewPlanPending( 1 );
```

3.5.5 InTransition

This function indicates if the controller at the specified node is currently in transition.

Returns

= 0 - not in transition

= 1 - in transition

= -1 - specified node is not under actuated control

Parameters

1. int nNodeID - ID of the node at which the controller is located.

Example

```
int i = InTransition( 1 );
```

3.5.6 GetLocalCycleTimer

This function returns the value of the controller's local cycle timer in seconds past its yield point.

Returns

> = 0 - local timer value

= -1 - specified node is not under actuated control

Parameters

1. int nNodeID - ID of the node at which the controller is located.

Example

```
int value = GetLocalCycleTimer( 1 );
```

3.5.7 GetCycleLength

This function returns the cycle length currently in use by the controller at the specified node.

Returns

> = 0 - cycle length currently in use by the controller

= -1 - specified node is not under actuated control

Parameters

1. int nNodeID - ID of the node at which the controller is located.

Example

```
int CycleLength = GetCycleLength( 1 );
```

3.5.8 GetNewCycleLength

This function returns the cycle length that will be implemented at the next plan update by the controller at the specified node.

Returns

> = 0 - cycle length that will be implemented at the next plan update by the controller

= -1 - specified node is not under actuated control

Parameters

1. int nNodeID - ID of the node at which the controller is located.

Example

```
int NewCycleLength = GetNewCycleLength( 1 );
```

3.5.9 SetNewCycleLength

This function sets the cycle length that will be implemented at the next plan update by the controller at the specified node.

Returns

= 0 - success

= -1 - specified node is not under actuated control

= -2 - a plan transition is already in progress

= -3 - specified cycle length is invalid

Parameters

1. int nNodeID - ID of the node at which the controller is located.
2. int nCycleLength - the cycle length to be used.

Example

```
int i = SetNewCycleLength( 1, 100 );
```

3.5.10 GetOffset

This function returns the offset currently in use by the controller at the specified node.

Returns

= 0 - offset currently in use by the controller

= -1 - specified node is not under actuated control

Parameters

1. int nNodeID - ID of the node at which the controller is located.

Example

```
int offset = GetOffset( 1 );
```

3.5.11 GetNewOffset

This function returns the offset that will be implemented at the next plan update by the controller at the specified node.

Returns

>= 0 - offset that will be implemented at the next plan update by the controller

= -1 - specified node is not under actuated control

Parameters

1. int nNodeID - ID of the node at which the controller is located.

Example

```
int newoffset = GetNewOffset( 1 );
```

3.5.12 SetNewOffset

This function sets the offset that will be implemented at the next plan update by the controller at the specified node.

Returns

= 0 - success

= -1 - specified node is not under actuated control

= -2 - a plan transition is already in progress

= -3 - specified cycle length is invalid

Parameters

1. int nNodeID - ID of the node at which the controller is located.
2. int nOffset - the offset to be used.

Example

```
int i = SetNewOffset( 1, 5 );
```

3.5.13 GetSplits

This function returns the phase splits currently in use by the controller at the specified node.

Returns

= 0 - success

= -1 - specified node is not under actuated control

Parameters

1. int nNodeID - ID of the node at which the controller is located.
2. int nSplits[8] - address of an array to receive the split values.

Example

```
int splits[8];
int i = GetSplits( 1, splits );
```

3.5.14 GetMinSplits

This function returns the minimum allowable phase splits based on the current min. green, pedestrian, and clearance times for the controller at the specified node.

Returns

= 0 - success

= -1 - specified node is not under actuated control

Parameters

1. int nNodeID - ID of the node at which the controller is located.
2. int nMinSplits[8] - address of an array to receive the minimum split values.

Example

```
int minsplits[8];
int i = GetMinSplits( 1, minsplits );
```

3.5.15 GetNewSplits

This function returns the phase splits that will be implemented at the next plan update by the controller at the specified node.

Returns

= 0 - success

= -1 - specified node is not under actuated control

Parameters

1. int nNodeID - ID of the node at which the controller is located.
2. int nSplits[8] - address of an array to receive the split values.

Example

```
int newsplits[8];
int i = GetNewSplits( 1, newsplits );
```

3.5.16 SetNewSplits

This function sets the phase splits that will be implemented at the next plan update by the controller at the specified node.

Returns

= 0 - success

= -1 - specified node is not under actuated control

= -2 - at least one specified split is invalid

Parameters

1. int nNodeID - ID of the node at which the controller is located.
2. int nSplits[8] - address of an array that contains the new split values.

Example

```
int newsplits[8] = { 15, 30, 15, 20, 15, 30, 15, 20 };
int i = SetNewSplits( 1, newsplits );
```

3.5.17 GetTransitionMethod

This function returns the plan transition method and parameters for the controller at the specified node.

Returns

= 0 - short way transition method

= 1 - dwell transition method

= 2 - add transition method

= 3 - subtract transition method

= -1 - specified node is not under actuated control

Parameters

1. int nNodeID - ID of the node at which the controller is located.
2. int* nMaxPctAdd - address of integer to receive the maximum percent add value.
3. int* nMaxPctSub - address of integer to receive the maximum percent subtract value.

Example

```
int* nMaxPctAdd;
int* nMaxPctSub;
int method = GetTransitionMethod( 1, nMaxPctAdd, nMaxPctSub );
```

3.5.18 SetTransitionMethod

This function returns the plan transition method and parameters for the controller at the specified node.

Returns

= 0 - success

= -1 - specified node is not under actuated control

= -2 - invalid method specified

= -3 - maximum percent add out of range [1, 100]

= -4 - maximum percent subtract out of range [1, 100]

Parameters

1. int nNodeID - ID of the node at which the controller is located.
2. int nMethod - plan transition method:

   0 = short way transition method

   1 = dwell transition method

   2 = add transition method

   3 = subtract transition method
3. int nMaxPctAdd - maximum percent add value.
4. int nMaxPctSub - maximum percent subtract value.

Example

```
int i = SetTransitionMethod( 1, 0, 20, 20 );
```

3.5.19 GetTODOP

This function returns the state of the time-of-day (TOD) operation for the controller at the specified node. When the TOD operation is enabled and a new plan is specified for a subsequent time period, that plan will be implemented by the controller. When TOD operation is disabled, the controller will not load plans that are specified in time periods after the first time period. The TOD operation does not affect plans that are specified via the API.

Returns

= 0 - time-of-day operation disabled

= 1 - time-of-day operation enabled

= -1 - specified node is not under actuated control

Parameters

1. int nNodeID - ID of the node at which the controller is located.

Example

```
int i = GetTODOP( 1 );
```

3.5.20 SetTODOP

This function sets the state of the time-of-day operation for the controller at the specified node. When the TOD operation is enabled and a new plan is specified for a subsequent time period, that plan will be implemented by the controller. When TOD operation is disabled, the controller will not load plans that are specified in time periods after the first time period. The TOD operation does not affect plans that are specified via the API.

Returns

= 0 - success

= -1 - specified node is not under actuated control

Parameters

1. int nNodeID - ID of the node at which the controller is located.
2. int nTOD - flag indicating the state of the time-of-day operation: 0 = disabled, 1 = enabled.

Example

```
int i = SetTODOP( 1, 1 );
```

3.5.21 GetActivePhaseInfo

This function returns information about the active phase for the specified ring and node.

Returns

= 0 - no active phase

> 0 - number of the active phase [1, 8]

= -1 - specified node is not under actuated control

Parameters

1. int nNodeID - ID of the node at which the controller is located.
2. int nRing - controller ring.
3. int* nElapsedTim - address of integer to receive the elapsed time for active phase
4. int* nCause - address of integer to receive the reason the phase became active:

    0 = not active

    1 = vehicle call

    2 = max. termination

    3 = min. recall

    4 = max. recall

    5 = pedestrian call or recall

    6 = dual entry

Example

```
int* nElapsedTim;
int* nCause;
int ActivePhase = GetActivePhaseInfo( 1, 1, nElapsedTim, nCause );
```

3.5.22 GetTerminatedPhaseInfo

This function returns information about the last phase to be terminated in the specified ring for the specified node.

Returns

= 0 - no active phase

> 0 - number of the terminated phase [1, 8]

= -1 - specified node is not under actuated control

Parameters

1. int nNodeID - ID of the node at which the controller is located.
2. int* nRing - controller ring.
3. int* nCause - address of integer to receive the reason the phase was terminated:

   0 = not terminated (i.e., it is active)

   1 = forced off

   2 = gapped out

   3 = maxed out

Example

```
int* nCause;
int TerminatedPhase = GetTerminatedPhaseInfo( 1, 1, nCause );
```

3.5.23 GetTerminationCode

This function returns the termination code for the specified node and phase.

Returns

= 0 - not terminated (i.e., it is active)

= 1 - forced off

= 2 - gapped out

= 3 - maxed out

= -1 - specified node is not under actuated control

Parameters

1. int nNodeID - ID of the node at which the controller is located.
2. int nPhase - phase number.

Example

```
int TerminationCode = GetTerminationCode( 1, 8 );
```

3.5.24 GetDwellGreenPoint

This function returns the dwell green point, in seconds, for the specified node and phase. The dwell green point is referenced to the start of the simulation (during simulation initialization, the value is referenced to the start of initialization).

Returns

= 0 - active phase, dwell point not reached

> 0 - dwell green point in seconds

= -1 - specified node is not under actuated control

Parameters

1. int nNodeID - ID of the node at which the controller is located.
2. int nPhase - phase number.

Example

```
int DwellGreenPoint = GetDwellGreenPoint( 1, 8 );
```

3.5.25 GetSignal

This function returns the signal values for the specified link (can be used with pre-timed control and sign control as well).

Returns

= 0 success

= -1 no link found

Parameters

1. int nUpNodeID - Upstream node for the link.
2. int nDwnNodeID - Downstream node for the link.
3. int* nLeft - address of integer to receive the left turn movement code.
4. int* nThru - address of integer to receive the thru turn movement code.
5. int* nRight - address of integer to receive the right turn movement code.
6. int* nDiag - address of integer to receive the diagonal turn movement code.

Signal Code Values

0 = red

1 = amber

2 = green (includes protected green)

3 = permitted green

Example

```
int* nLeft;
int* nThru;
int* nRight;
int* nDiag;
int i = GetSignal( 101, 102, nLeft, nThru, nRight, nDiag );
```

3.5.26 SetSignal

This function sets the signal values for the specified link (can be used with pre-timed control and sign control as well).

Returns

= 0 success

= -1 no link found

= -2 invalid signal code

Parameters

1. int nUpNodeID - Upstream node for the link.
2. int nDwnNodeID - Downstream node for the link.
3. int nLeft - Left turn movement code.
4. int nThru - Thru turn movement code.
5. int nRight - Right turn movement code.
6. int nDiag - Diagonal turn movement code.

Signal Code Values

0 = red

1 = amber

2 = green (includes protected green)

3 = permitted green

Example

```
int i = SetSignal( 101, 102, 0, 2, 2, 0 );
```

4 Migrating Existing RTEs to TSIS 6.0

Recent changes to CORSIM and TSIS make it much simpler to develop and compile RTEs. This section discusses how to migrate RTEs that are compatible with older versions of TSIS and CORSIM to the newest version of TSIS. These changes eliminate the need for linking to the KSC.obj file. Consequently, the FORTRAN libraries and FORTRAN compiler referred to in previous versions of the documentation are no longer needed for successful compilation of an RTE. To migrate an RTE to TSIS 6.0, the three RTE interface functions called by the CORSIM Server must be exported directly from the C or C++ code. The CORSIM Server will expect these functions, which are exported to the RTE's library, to be decorated in a specific manner. A sample of code that accomplishes this, when using the Microsoft Visual C++ compiler, is provided in Section 2.1.

In addition to removing the KSC.obj file from your project, exporting the three functions called by CORSIM inside the C++ code, and compiling your existing RTE with the new libraries provided, there are three major changes that impact existing RTE's:

1. CORSIM has converted a number of its statically allocated arrays into dynamically allocated arrays,
2. The TSIS package provides a set of exported functions that can be used by the RTE,
3. The RTE is now specified as part of the CORSIM Driver tool (or its debug version, CORDebug).

Because many of the arrays in CORSIM are now dynamically allocated arrays, it is not necessary to import these data structures by linking the commons in CORSIM to C structs and then accessing certain elements of these structs. Instead, the arrays can be imported directly (see Sections 2.2 and 3.3 for examples). However, the names of the CORSIM arrays have changed and must be updated in your code. The user is cautioned about not exceeding the bounds for these dynamically allocated arrays and accessing memory inappropriately. CORSIM provides current element counts for all dynamically allocated arrays, which are described in the CORSIM Data Dictionary.

Section 3.2 describes a set of functions that TSIS provides that can be called from an RTE. These functions enable an RTE to display text in the CORSIM Driver window within TShell and to send Windows messages to the CORSIM Server. Because these functions are exported via the CORWin library, the RTE can directly access them. Previously they had to be defined in a separate FORTRAN file (i.e., KSC.for).

In the latest version of TSIS, a separate driver tool executes the CORSIM simulation as opposed to TShell running CORSIM. There are two versions of the driver, one to execute CORSIM normally (release mode), and one to run CORSIM in debug mode. If you wish to run the debugger with your RTE, you will need to specify CORDebug.dll rather than CorsimDriver.dll when defining the RTE tool in TSIS. When configuring an RTE that is to be debugged, this library should be referenced in the **Path** edit box of the Tool Configuration property page. See Section 1.3 for obtaining a copy of CORDebug.dll, which is not part of the standard TSIS installation package.

Glossary of Terms

ATMS

Advanced Traffic Management Systems

CID

Controller Interface Device. This is a hardware interface used to connect a signal controller to a computer running a traffic simulation.

Component

An independent software application that can be easily integrated into other software applications or into a container program.

Component Architecture

A software architecture in which a framework, called a container, supports the use and interaction of independent software components (tools).

Container

A computer program composed of a framework that supports the use and interaction of independent software components.

COM

Component Object Model

CORDebug

This is the name of the debug version of the CORSIM Driver DLL. It is used for debugging RTEs.

CORSIM

CORridor SIMulation. A microscopic traffic simulation tool supported by the TSIS environment.

CORSIM Data Dictionary

This document contains a description of the CORSIM elements that are shared for access by code that resides outside of the CORSIM DLL.

CORSIM Driver

This is the name of the DLL that is installed as a tool in the TSIS package and is used to run the CORSIM simulation.

CORSIM Server

The CORSIM Server is an interface between the CORSIM Driver tool and the CORSIM DLL. The Server enables TSIS to run multiple simulations at one time.

CORWin

The CORWin interface provides a Windows interface between the CORSIM Server and the CORSIM DLL. It also serves as an interface between an RTE and the CORSIM Server, which enables the RTE to display messages in the CORSIM Driver tool.

DLL

Dynamic-Link Library. In general, a DLL is a file that contains one or more functions that are compiled, linked, and stored separately from the processes that use them.

DOT

Department of Transportation

Dynamic-Link Library

In general, a DLL is a file that contains one or more functions that are compiled, linked, and stored separately from the processes that use them.

Dynamic Memory

Memory that is allocated while a process is running as opposed to statically allocated at the time a module is compiled and linked.

Exported Functions

A function that resides in a DLL, but that made accessible (exported) to other processes.

FHWA

Federal Highway Administration. Sponsor for the development of the TSIS suite of traffic analysis tools.

Graphical User Interface

An interface between a user and a software tool, consisting of graphical elements and controls, e.g., windows, dialogs, buttons.

GUI

Graphical User Interface

NETSIM

NETwork SIMulation. The part of the CORSIM simulation that models surface-street operations.

RTE

Run-Time Extension

Run-Time Extension

A method by which a TSIS user can extend (or replace) functionality in the CORSIM simulation without having to modify, compile, and link the CORSIM code. It is typically used to modify/replace the signal control logic in CORSIM.

Shared Memory

A mechanism by which CORSIM exports its memory for use by other processes such as run-time extensions.

Time Step

The smallest unit of time at which CORSIM moves vehicles (updates vehicle positions). This is also the frequency at which CORSIM calls a run-time extension's main execution function.

Tool

A program or component that is installed into the TSIS environment for use in conducting traffic operations analysis. A tool can be an application (EXE), Dynamic-Link Library (DLL), COM object or ActiveX Control (OCX), or a batch program (BAT).

TRAFED

TRAFED is a graphical user interface-based editor that allows you to easily create and edit traffic networks and simulation input for the CORSIM model.

TRAFVU

TRAFVU (TRAF Visualization Utility) is a user-friendly graphics post-processor that displays traffic networks, animates simulated traffic flow operations, animates and displays simulation output measures of effectiveness, and displays user-specified input parameters for simulated network objects.

TRF

A file that contains the input data used to define a CORSIM network and to drive the CORSIM simulation for a single simulation case.

TShell

The graphical user interface for the TSIS integrated development environment. It provides a Project view that enables you to manage your TSIS projects. It is also the container for the pre-configured tools and any tools that you add to the suite.

TSIS

Traffic Software Integrated System. TSIS is the integrated development environment that hosts the CORSIM simulation and its support tools.

TSIS Website

This website, http://mctrans.ce.ufl.edu/featured/tsis, contains the latest information about new tools, product updates, known problems, example projects, and usage tips.

5 Index

A

actuated control API functions
  GetACID 3-13
  GetActivePhaseInfo 3-21
  GetCycleLength 3-15
  GetDwellGreenPoint 3-22, 3-23
  GetLocalCycleTimer 3-15
  GetNewCycleLength 3-16
  GetNewOffset 3-17
  GetOffset 3-16, 3-17
  GetSignal 2-7, 2-8, 3-23
  GetSyncReferenceTime 3-14
  GetTerminationCode 3-22
  GetTODOP 3-20
  GetTransitionMethod 3-19
  InTransition 3-15
  IsActuated 3-14
  IsNewPlanPending 3-14
  SetNewCycleLength 3-16
  SetNewOffset 3-17
  SetNewSplits 3-18, 3-19
  SetSignal 2-6, 2-7, 3-23, 3-24
  SetTODOP 3-20, 3-21
  SetTransitionMethod 3-19, 3-20
actuated signal control 1-3, 2-6, 2-22, 3-13, 3-14, 3-15, 3-16, 3-17, 3-18, 3-19, 3-20, 3-21, 3-22
adaptive signal control 1-1, 2-18
API vii, 1-3, 2-6, 2-7, 3-13, 3-20
arrays
  dimension 3-6
  dynamically allocated 2-3, 2-18, 2-19, 3-5, 3-6, 4-1
  FORTRAN 2-3, 2-4
  index 2-4
  statically allocated 2-3, 3-5, 3-6, 4-1, 4-2

C

C++ 2-2, 2-18, 2-21, 3-1, 3-2, 3-5, 3-11, 3-12, 3-13, 4-1
CID (Controller Interface Device) 1-1, 4-1
COM (Component Object Model) 1-2, 2-11, 4-1, 4-3
common blocks 2-3, 3-5, 3-6
compilers 1-3, 2-2, 4-1
  FORTRAN 4-1
Component Object Model 1-2, 4-1
configuring an RTE v, vii, 1-1, 2-1, 2-9, 2-10, 2-17, 2-21, 4-2, 4-4
Controller Interface Device 1-1, 4-1
controllers 1-1, 3-13, 3-14, 3-15, 3-16, 3-17, 3-18, 3-19, 3-20, 3-21, 3-22, 3-23, 4-1
  170 1-1
  2070 1-1
  NEMA 1-1
CORDebug 2-11, 4-1, 4-2, 4-1
CORSIM i, v, vii, 1-1, 1-2, 1-3, 2-1, 2-2, 2-3, 2-5, 2-6, 2-7, 2-8, 2-9, 2-10, 2-11, 2-12, 2-13, 2-14, 2-15, 2-17, 2-18, 2-18, 2-19, 2-20, 2-21, 2-22, 3-1, 3-2, 3-3, 3-4, 3-5, 3-6, 3-7, 3-10, 3-11, 3-12, 3-13, 4-1, 4-2, 4-1, 4-2, 4-3, 4-4
  architecture 1-2
  data structures vii, 1-2, 1-3, 2-1, 2-2, 2-6, 2-18, 2-21, 4-1
  DLL 2-12, 3-6, 3-13, 4-2
  exported functions 1-3, 2-6, 2-7, 3-6
  input file 2-11, 2-17, 2-21, 2-22
  libraries 2-1, 2-17
  shared memory 1-2, 3-5, 3-9, 3-10, 4-3
  User's Guide 2-14
CORSIM Data Dictionary 2-18, 3-5, 4-1, 4-2
CORSIM Driver 1-2, 2-8, 2-9, 3-3, 3-4, 3-5, 3-12, 3-13, 4-1, 4-2
CORSIM exported functions 1-3, 2-6, 2-7, 3-6
  abortcorsim 3-6, 3-7
  assign_path 3-7
  change_path 3-7, 3-8
  closelane 3-8, 3-9
  get_generated_vehicle_ids 3-8
  GetSignal 2-7, 2-8, 3-23
  gettxdversion 3-8, 3-9
  getvehicledata 3-9
  openlane 3-9, 3-10
  put_path 3-10
  put_vehicle 3-10, 3-11
  putvmsspeed 3-11
  resize 3-11
  SetSignal 2-6, 2-7, 3-23, 3-24
CORSIM Properties 2-12
CORSIM Server 1-2, 2-1, 2-21, 3-2, 3-3, 3-5, 3-6, 4-1, 4-2
CORWin 1-2, 2-8, 2-9, 3-3, 3-12, 4-1, 4-2

CORWin interface functions 2-8, 3-3
  MsgBox 3-3
  OutputString 2-8, 2-9, 3-3, 3-4, 3-12
  RequestKeyInput 3-4
  SendMsg 3-4
  SetHWND 3-5

D

debugging 2-11, 4-1, 4-2, 4-1
  breakpoints 2-11
declspec keyword 2-2, 2-3, 2-6, 2-7, 2-9, 3-1, 3-2, 3-5
define statement 2-18, 3-6
detectors 2-3, 2-18, 2-19, 2-20, 2-21
DLL 1-2, 2-1, 2-2, 2-3, 2-5, 2-6, 2-7, 2-8, 2-9, 2-10, 2-11, 2-12, 2-15, 2-16, 2-17, 3-1, 3-2, 3-3, 3-5, 3-6, 4-1, 4-2, 4-3
DLL_EXPORT 2-2, 2-9, 3-1, 3-2, 3-3
DLL_IMPORT 2-3, 2-5, 2-6, 2-7, 3-5, 3-6
dllexport 2-2, 3-1, 3-2
dllimport 2-3, 2-6, 2-7, 2-9, 2-21, 3-5
dynamic memory vii, 4-2
dynamically allocated 2-3, 2-18, 2-19, 3-5, 3-6, 4-1

E

export 2-1, 2-17
exported functions 1-2, 4-1, 4-2
extern 2-2, 2-3, 2-6, 2-7, 2-9, 3-1, 3-2, 3-5

F

Federal Highway Administration i, 4-3
FHWA i, vii, 4-3
FORTRAN 1-3, 2-3, 2-4, 2-5, 3-1, 3-2, 3-3, 3-5, 3-6, 3-11, 3-12, 3-13, 4-1, 4-2
  common blocks 2-3, 3-5, 3-6
Function Format 2-2
function naming decorations 2-2, 4-1

I

import 2-5, 2-6, 2-7, 2-21, 3-5, 3-6, 4-1
intersections 1-1, 2-17, 2-22
ITT Industries i, 2-17

L

libraries 1-2, 1-3, 2-1, 2-2, 2-8, 3-12, 4-1, 4-2
links 1-2, 2-1, 2-2, 2-3, 2-5, 2-6, 2-7, 2-8, 2-18, 2-19, 2-21, 3-7, 3-8, 3-9, 3-10, 3-11, 3-23, 3-24, 4-2, 4-3

M

messages 1-2, 2-8, 2-9, 2-13, 3-3, 3-4, 3-11, 4-2
  control 3-4
  status 3-4
  text 1-2, 2-8, 2-9, 3-3
  Windows 2-8, 3-3, 4-1
migrating an RTE to TSIS 6.0 4-1
multiple runs 2-14

N

nodes 2-3, 2-5, 2-6, 2-7, 2-17, 2-18, 2-19, 2-21, 2-22, 3-8, 3-9, 3-10, 3-11, 3-13, 3-14, 3-15, 3-16, 3-17, 3-18, 3-19, 3-20, 3-21, 3-22, 3-23, 3-24

P

pre-timed signal control 2-21, 2-22, 3-23
prototype function 2-2

R

RTE interface 1-2, 3-1, 4-1
RTE interface functions 2-15, 2-21, 3-1, 4-1
  exit 1-2, 2-2, 2-21, 3-2
  initialization 3-1
  main execution 2-2, 3-2, 4-3
run-time extension interface vii, 1-2

S

sample RTE 1-1, 2-17, 2-22
SampleRTE 2-17, 2-22
shared memory 1-2, 3-5, 3-9, 3-10, 4-3
signal ii, 1-1, 2-6, 2-7, 2-8, 2-17, 2-21, 2-22, 3-23, 3-24, 4-1, 4-3
signal control
  actuated 1-3, 2-6, 2-22, 3-13, 3-14, 3-15, 3-16, 3-17, 3-18, 3-19, 3-20, 3-21, 3-22
  adaptive 1-1, 2-18
  CORSIM 2-22
  external 2-22
  pre-timed 2-21, 2-22, 3-23
  semi-actuated 2-22
signal phasing 2-21
signal states 1-1, 2-6, 2-7, 2-8, 2-21, 2-22
  transitions 2-21
simulation v, vii, 1-1, 1-2, 1-3, 2-3, 2-4, 2-9, 2-12, 2-14, 2-15, 2-18, 2-19, 2-20, 2-21, 2-22, 3-1, 3-2, 3-4, 3-5, 3-6, 3-7, 3-9, 3-13, 3-22, 4-2, 4-1, 4-2, 4-3, 4-4
simulation event loop 1-2
statically allocated 2-3, 3-5, 3-6, 4-1, 4-2

stdcall calling convention 2-2, 2-6, 2-7, 2-9, 3-1, 3-2, 3-3

T

time step 1-2, 2-2, 2-13, 3-1, 3-2, 3-8, 4-3
tool 1-1, 1-2, 2-10, 2-11, 2-15, 2-16, 2-17, 2-21, 3-5, 4-1, 4-2, 4-3
tool configuration 2-10, 2-16
TShell 1-1, 1-2, 2-8, 2-9, 2-10, 2-11, 2-12, 3-3, 3-12, 4-1, 4-2, 4-4
TSIS installation 2-11, 4-2
TSIS website 2-18

V

Visual C++ 1-3, 2-2, 2-17, 2-21, 4-1

W

Win32 API 2-2
Windows ii, 1-1, 2-8, 3-3, 3-5, 4-1, 4-2
wizard 2-10, 2-11, 2-12, 2-13, 2-14, 2-15, 2-16