
CTAT's Tool-Tutor Application Interface

Table of Contents

1. Introduction	1
2. Protocol	2
3. Format	2
3.1. XML Dormin	2
3.2. Selection-Action-Input	3
4. Start State	3
4.1. Demonstrating the Start State	3
4.2. Resetting to the Start State	4
5. Attempt-Response Messages	5
5.1. Problem-Solving Attempt	5
5.2. Hint Request	6
5.3. Done Attempt	6
6. Curriculum Messages	7
7. Tutor Implementation	7
7.1. Models of Tool-Tutor Communication	7
7.2. Standard Mechanisms	7
Glossary	8
A. Additional Materials	11

1. Introduction

The Tool-Tutor application interface refers to the communication between the tutoring system that evaluates student interactions, providing hints and feedback, and the tool (e.g., user interface) that the student interacts with during problem solving.

In CTAT, the tutor is represented by the Behavior Recorder program. The interface between the Behavior Recorder and tool can be described in terms of a protocol for communication, a format for message exchange, and the validity of message timing and content.

This guide is intended for the programmer who desires to create or modify the interface between a tool (e.g., CyclePad, Flash, etc.) and the CTAT tutoring application (the Behavior Recorder). The goal of creating this interface may be:

- to add tutoring to a program; or
- to log student interactions with a program (e.g., for bootstrapping¹ as part of cognitive task analysis, or for logging a control condition).

This task requires knowledge of TCP Socket programming.

¹Bootstrapping refers to the process by which problem-solving data are used to create an initial model for tutoring in an application. These data are generated by students who solve problems in an application (tool), and are saved in log files. See 'Bootstrapping Novice Data: Semi-Automated Tutor Authoring Using Student Log Files' (McLaren, et. al. 2004) (PDF [<http://www.pitt.edu/~bmclaren/ITS2004-BND.pdf>]).

2. Protocol

The protocol for tool-tutor communication is TCP (Transmission Control Protocol). All communication between the tool and tutor takes place through a TCP socket on both the tool (client) and the Behavior Recorder (server).

The open port on the Behavior Recorder is port 1502. Currently, the most common setup is for both the tool and the Behavior Recorder to run on the same host. In this configuration, the tool should address `localhost` on port 1502. Support for running the Behavior Recorder on an external networked host is currently in the development phase.



Note

For more information on connecting to the Behavior Recorder via TCP SocketProxy, see Additional Materials .

3. Format

The format of messages exchanged between the Behavior Recorder and a tool is the Dormin format. Three variations of the Dormin format are used by the Behavior Recorder: native Dormin, XML Dormin, and the DataShop DTD format.²

Some legacy applications (as well as CTAT Java applications) utilize the native Dormin format. Of the two Dormin formats, XML Dormin is the more human readable.

3.1. XML Dormin

An example message in XML Dormin is presented below.

```
<message>
  <verb>NotePropertySet</verb>
  <properties>
    <MessageType>InterfaceAction</MessageType>
    <Selection>
      <value>Instruction1_ta</value>
    </Selection>
    <Action>
      <value>UpdateTextArea</value>
    </Action>
    <Input>
      <value>TEXT</value>
    </Input>
  </properties>
</message>
```

As in all Dormin messages exchanged between the tool and tutor, the root element is `<message>`. The first child element, `<verb>`, is used by Lisp in tools such as CyclePad and in TDK-based tutors, but is not presently used by CTAT.

The `<properties>` element wraps the `<MessageType>` element, and a series of property-value pairs. A property-value pair is defined as a property element (e.g., `Selection`) and a child value element.

²The DataShop DTD format is used for logging to either the hard disk or DataShop server for analysis. See `TutorMessage.dtd` for the currently implemented version of the format; future versions of the logging DTD in CTAT will be based on the DTDs available at DataShop's DTD page. [<http://learnlab.web.cmu.edu/dtd/>]



Note

New property-value pairs can be included in any message as long as they are children of the `<properties>` element; descendant (i.e, non-children) property-value pairs will be discarded by the Behavior Recorder.

For a list of all message types, see the Glossary of Message Types.

3.2. Selection-Action-Input

Selection-Action-Input is a common notion implicit in the Dormin format. Sometimes referred to as SAI, the Selection-Action-Input triple is a way of representing a student action in three dimensions: 'Selection' refers to the instance name of an interactive widget in the interface; 'Action' refers to the action that the student took upon the widget; and 'Input' is the value that the student entered. Where an input doesn't apply (such as a Done button), the value of -1 is used.

While the notion of SAI remains constant across the various message types, it is employed in slightly different ways. For example, requesting a hint corresponds with a selection of both "hint" and the instance name of the currently selected object.

4. Start State

In CTAT, problems are defined in Behavior Response Data files, or BRDs. A single BRD file represents one problem; it contains the structure of that problem, including hint and error messages. The BRD file is managed by the Behavior Recorder program.

Each problem BRD contains a definition of the initial problem state--the interface values that the student will first see when working on a problem. This initial problem state is called the *start state*.

A tool that communicates with the Behavior Recorder can enable a problem author to demonstrate the start state for a problem. The tool can also reset the interface to this start state, or to another state defined in the BRD. All of these functions are accomplished by exchanging Dormin messages with the Behavior Recorder.

4.1. Demonstrating the Start State

A problem author will commonly define a start state by demonstrating values in a tool's user interface. As the problem author inputs an initial problem state, each action should correlate with an `InterfaceAction` message that is sent to the Behavior Recorder at the time of demonstration. Each of these `InterfaceAction` messages will be recorded by the Behavior Recorder and stored in the BRD file.

A typical `InterfaceAction` message is given below:

```
<message>
  <verb>NotePropertySet</verb>
  <properties>
    <MessageType>InterfaceAction</MessageType>
    <Selection>
      <value>Instruction1_ta</value>
    </Selection>
    <Action>
      <value>UpdateTextArea</value>
    </Action>
    <Input>
      <value>TEXT</value>
    </Input>
  </properties>
</message>
```

```
</properties>  
</message>
```

The `InterfaceAction` text is enclosed in a `MessageType` element. The three vectors that describe the `InterfaceAction` message are `Selection`, `Action`, and `Input`. `Selection` describes a unique name of the interface element that was selected. `Action` describes the action that was applied to the interface element. `Input` represents the input value that was entered by the user.

After entering problem-specific information, the tutor author will then complete the start state by selecting the Behavior Recorder menu command `Graph > Create Start State`. In response, the Behavior Recorder sends a message of type `StartStateCreated` back to the tool:

```
<message>  
  <verb>SendNoteProperty</verb>  
  <properties>  
    <MessageType>StartStateCreated</MessageType>  
    <StartStateName>Name-of-start-state</StartStateName>  
  </properties>  
</message>
```

Most tools, upon receiving a `StartStateCreated` message, should lock all interface widgets that accept start values (e.g., a text box that contains problem description text). The exception is the tool that allows for the student to modify start state values.

4.2. Resetting to the Start State

A tutor is often reset to its start state. This occurs, for example, when loading a BRD file or selecting the start state node in the Behavior Recorder.

Resetting to the start state triggers a sequence of message events to take place:

1. The Behavior Recorder sends a message of type `StartProblem` to the tool; in response, the tool clears all values in the interface.
2. The tool sends a sequence of `InterfaceDescription` messages that describe the interface to Jess.
3. The Behavior Recorder sends `InterfaceAction` messages that it has stored to represent the start state; in response, the tool parses the `InterfaceAction` messages and updates its interface accordingly.
4. The Behavior Recorder sends the `StartStateEnd` message to signify that it has sent all start state messages.
5. The tool locks the interface widgets that are part of the start state and

4.2.1. Clearing the Interface (`StartProblem`)

The `StartProblem` message signifies that the start state has been requested. In response, the tool interface should:

1. Clear all values from interface widgets so that the widgets resemble their initial compiled state.



Note

There is no need to record the values that existed before clearing--the Behavior Recorder will supply the start state values through `InterfaceAction` messages later in the reset phase.

4.2.2. Describing the Interface to Jess (Cognitive Tutors only)

The tool interface should be described to aid the Jess rule engine in generating an adequate representation of working memory. If the interface is not described, potential tutor authors (rule developers) will need to author the working memory representation by hand.

Describing the interface is typically done by sending a single message of type `InterfaceDescription`. For the details of this message, see `InterfaceDescription` in the glossary.

4.2.3. Interpreting `InterfaceAction` messages

The Behavior Recorder will send a series of `InterfaceAction` messages representing the steps that led up to the finished start state. Each message should be processed and the tool interface should be updated accordingly.

The final message of type `StartStateEnd` signifies that all messages describing the start state have been sent.

5. Attempt-Response Messages

During problem solving, the tool should inform the Behavior Recorder of all attempts made by the student during problem solving. This is accomplished with the `InterfaceAction` message.

A few special types of attempt-response pairs are:

- Problem-solving attempt - Correct/Incorrect response
- Hint request - hint message response
- Done attempt - response

5.1. Problem-Solving Attempt

All problem-solving attempts are communicated with the `InterfaceAction` message. In this basic action, the message properties are selection, action, and input. Selection is the name of the widget. Action is the action performed on the widget. Input is the value(s) entered in the widget.

In response, the Behavior Recorder will send one of the following message types:

- `CorrectAction`
- `InCorrectAction`

A `CorrectAction` message may be accompanied by a message of type `SuccessMessage`.

An InCorrectAction message may be accompanied by a message of type BuggyMessage.

For details on these messages, consult the Glossary of Message Types.

5.2. Hint Request

When a student requests a hint, a message of type InterfaceAction should be sent. The selection vector should contain values of hint and the instance name of the currently selected object.

In response, the Behavior Recorder will send a message of type ShowHintsMessage. A sample message is shown below:

```
<message>
  <verb>SendNoteProperty</verb>
  <properties>
    <MessageType>ShowHintsMessage</MessageType>
    <HintsMessage>
      <value>This is the first hint.</value>
      <value>Please enter '2' in the highlighted field.</value>
    </HintsMessage>
    <Selection>
      <value>firstNumGiven</value>
    </Selection>
    <Action>
      <value>UpdateTextArea</value>
    </Action>
    <Input>
      <value>2</value>
    </Input>
  </properties>
</message>
```



Note

The ShowHintsMessage contains the values of *all hints*. If more than one hint is returned, the tool should store all hints and present a method to the student for accessing the other hints.

5.3. Done Attempt

The Done attempt, commonly accomplished by pressing a Done button, is treated like any other problem-solving attempt; in response to the InterfaceAction message, the Behavior Recorder will respond with either a CorrectAction or InCorrectAction message.

An example message for a done attempt is shown below:

```
<message>
  <verb>NotePropertySet</verb>
  <properties>
    <MessageType>InterfaceAction</MessageType>
    <Selection><value>done</value></Selection>
    <Action><value>ButtonPressed</value></Action>
    <Input><value>-1</value></Input>
  </properties>
</message>
```

6. Curriculum Messages

To be supplied.

7. Tutor Implementation

7.1. Models of Tool-Tutor Communication

Tool-tutor communication can be implemented using one of two communication models: asynchronous or synchronous communication. In asynchronous communication, the tool is always ready to accept messages from the Behavior Recorder. In synchronous communication, the tool waits for

7.2. Standard Mechanisms

For a tool to successfully support tutoring, it should also include:

- a *hint* facility that allows the student to request and view a hint, and cycle through hints if more than one exists;
- a *done* facility that allows the student to signal when they are done with the current problem;
- a *flag* facility for signifying to the student whether their input is correct or incorrect; and
- a *lock* facility for locking correct student input.

These facilities are not required for the tool to operate with CTAT's Behavior Recorder, but they have proven to be successful in intelligent tutor implementations over the years, and are therefore recommended.

For examples of how CTAT tutors have implemented these facilities, see the Fraction Addition Example-Tracing Tutor.

7.2.1. Hint

Commonly implemented as a button, the Hint facility allows the student to request helpful information regarding the current step of the problem.

Hint messages can either be shown within the tool, or within the Behavior Recorder's Hint window.

7.2.2. Done

Commonly implemented as a button, the Done facility allows the student to signal when they have finished the current problem.

7.2.3. Flag

Flagging refers to a modification made to the tool interface that reflects the evaluation of the student's input. CTAT tutors built with either the Java or Flash platform implement a flag as a visual change to the evaluated widget; in CTAT, green signifies correct input, red signifies incorrect input, and yellow signifies input that is currently being evaluated by the production system. The color is used to either outline the current widget, or to modify the font.

7.2.4. Lock

Locking is the process of freezing the current state of a widget, and preventing further modification. In CTAT tutors, locking correct input is an optional setting.

Glossary

Glossary of Message Types

AdvanceProblem	Empty message sent by the tool to advance to the next problem. In CTAT Java interfaces, this message is invoked by the teacher to advance problems for the student.
BuggyMessage	Sent by the Behavior Recorder if an error message exists on the current edge. Contains two elements: <pre><BuggyMsg>Text of error feedback.</BuggyMsg> <Selection> <value>firstTextField</value> </Selection></pre>
CorrectAction	Sent by the Behavior Recorder to signify that the student attempt was correction. Contains three elements:
GroupDescription	To be supplied.
GroupDescriptionEnd	Sent by the tool to signal that GroupDescription messages have been sent.
GroupDescriptionStart	Sent by the tool to signal that GroupDescription messages are to follow.
HighlightMsg	Sent by the Behavior Recorder to guide the student to work on the correct widget. Contains three elements: <pre><HighlightMsgText>Message text.</HighlightMsgText> <Selection> <value></value> </Selection> <Action> <value></value> </Action></pre>
InCorrectAction	Sent by the Behavior Recorder to indicate that the student attempt was incorrect. See CorrectAction.
InterfaceAction	Sent by either the tool or the Behavior Recorder to describe a solution step. During tutoring, InterfaceAction is commonly a student problem-solving attempt. This message type is also used to describe author actions that modify a widget prior to creating a start state in the behavior graph. During problem demonstration, an InterfaceAction message should be sent for each action. The final InterfaceAction message for a

problem should have a selection of done with value -1.

```
<message>
  <verb>NotePropertySet</verb>
  <properties>
    <MessageType>InterfaceAction</MessageType>
    <Selection>
      <value>dorminTextField1</value>
    </Selection>
    <Action>
      <value>UpdateTextField</value>
    </Action>
    <Input>
      <value>3</value>
    </Input>
  </properties>
</message>
```

InterfaceDescription

Sent by the tool to describe the interface to Jess. The vector `jess-Deftemplates` is used by Jess to create working memory; the vector `jessInstances` is used to populate working memory with values.

The following is a sample `InterfaceDescription` message that describes a single `DorminTextField` widget:

```
<WidgetType>DorminTextField</WidgetType>
<DorminName>dorminTextField1</DorminName>
<UpdateEachCycle>>false</UpdateEachCycle>
<jessDeftemplates>
  <entry>(deftemplate textField (slot name) (slot value))
</jessDefTemplates>
<jessInstances>
  <entry>(assert (textField (name dorminTextField1)))</en
</jessInstances>
```

The property `UpdateEachCycle` specifies whether or not to allow Jess to create solution steps.

IsTutorVisible

Sent by the Behavior Recorder to indicate whether the interface should be visible.

LoadBRDFileSuccess

Sent by the Behavior Recorder upon successful loading of BRD. Contains a single element:

```
<BRDFilePath>C:\Path\to\file.brd</BRDFilePath>
```

LoadBRDPathFile

Sent by the tool to load the student current working problem BRD file.

```
<BrdPathFileName>C:\path\to\file.brd</BrdPathFileName>
```

LoadProblem

Sent by the tool to load the problem BRD file in the Production System mode.

```
<ProblemName>Name-of-problem.brd</ProblemName>
```

LoadProblemWithHTTP	Sent by the tool to load the problem BRD file from the given URL. <ProblemName>http://www.webserver.org/path/to/brdfile.brd<
Logout	Empty message sent by the tool to log the student out of the system.
NumberOfTables	Sent by the tool to indicate the number of tables in the interface.
ResetAction	Sent by the Behavior Recorder to Java interfaces to set the specified widget with the reset value. Contains a single element: <Selection> <value>table1_C6R4</value> </Selection>
SendWidgetLock	Sent by the Behavior Recorder to specify whether widgets that receive correct input should be locked after a student attempt. It's value (true or false) is determined by the lockWidget property specified in problem BRD. This message is sent once when the tool connects to the tutor. Contains a single Boolean element: <WidgetLockFlag> <value>true</value> </WidgetLockFlag>
ShowHintsMessage	Sent by the Behavior Recorder to display hints. This message contains all hints for the current step. <HintsMessage> <value>This is the first hint.</value> <value>Please enter '2' in the highlighted field.</value> </HintsMessage> <Selection> <value>firstNumGiven</value> </Selection> <Action> <value>UpdateTextArea</value> </Action> <Input> <value>2</value> </Input>
ShowLoginWindow	Sent by the Behavior Recorder to prompt the tool to display a login window.
StartDialogue	Empty message sent by the tool to to initiate the dialogue system.
StartNewProblem	Empty message sent by the Behavior Recorder to indicate that the problem author created a new problem.
StartProblem	Sent by the Behavior Recorder when author clicks on the start state node of the graph in the Behavior Recorder, or when a graph is loaded. This is the first message in startNodeMessages container element (if loading from BRD). All fields in the interface should be cleared to ensure that the tool has a blank UI. Contains a single element:

	<code><ProblemName>Name of problem</ProblemName></code>
StartStateCreated	Sent by the Behavior Recorder when the author creates a start state. Contains a single element: <code><StartStateName>Name of start state</StartStateName></code>
StartStateEnd	Empty message sent by the Behavior Recorder that signifies all start state messages have been sent. This is the last message in startNodeMessages container element (if loading from BRD).
SuccessMessage	Sent by the Behavior Recorder to display a success message. Contains a single element: <code><SuccessMsg>Text of success message.</SuccessMsg></code>
Quit	Empty message sent by the tool to quit the application.
QuitWithoutSave	Empty message sent by the tool to quit the application without saving the student visited states.

A. Additional Materials

Additional materials can be accessed at <http://ctat.pact.cs.cmu.edu/source/>

`SocketProxyTest.java` provides a simple implementation of a program that connects to and exchanges messages with the Behavior Recorder's socket proxy.