RTP Template B/rtpt

Purpose

To issue run-time problem messages, and to perform some run-time type checking which may issue such messages.

---

§**1. Reporting.** All RTPs are produced by calling the following routine, which takes one compulsory argument: `n`, the RTP number. When I7 is being used with the Inform user interface, it's important that these numbers correspond to the explanatory web pages stored within the interface application: those in turn are created by the `inrtps` utility. The interface knows to display these pages because it parses the printed output during play to look for the text layout produced by the following routine: so do not reformat RTPs without ensuring that corresponding changes have been made to the Inform user interface applications.

The arguments `par1`, `par2` and `par3` are optional parameters clarifying the message; `ln` is an optional parameter specifying a paragraph number in the original source text (though in fact I7 does not use this at present).

```
[ RunTimeProblem n par1 par2 par3 ln   i c;
    if (enable_rte == false) return;
    enable_rte = false;
    print "^*** Run-time problem P", n;
    if (ln) print " (at paragraph ", ln, " in the source text)";
    print ": ";
    switch(n) {
        RTP_BACKDROP:
            print "Tried to move ", (the) par1, " (a backdrop) to ", (the) par2,
                ", which is not a region.^";
        RTP_CANTCHANGE:
            print "Tried to change player to ", (the) par1,
                ", which is not a player-character.^";
        RTP_NOEXIT:
            print "Tried to change ", (the) par2, " exit of ", (the) par1,
                ", but it didn't seem to have such an exit to change.^";
        RTP_EXITDOOR:
            print "Tried to change ", (the) par2, " exit of ", (the) par1,
                ", but it led to a door, not a room.^";
        RTP_IMPREL:
            print "Tried to access an inappropriate relation for ", (the) par1,
                ", violating '", (string) par2-->RR_DESCRIPTION, "'.^";
        RTP_RULESTACK:
            print "Too many procedural rules acting all at once.^";
        RTP_TOOMANYRULEBOOKS:
            print "Too many rulebooks in simultaneous use.^";
        RTP_TOOMANYEVENTS:
            print "Too many timed events are going on at once.^";
        RTP_BADPROPERTY:
            print "Tried to access non-existent property for ", (the) par1, ".^";
        RTP_UNPROVIDED:
```

```
    print "Since ", (the) par1, " is not allowed the property ~",
            (string) par2, "~, it is against the rules to try to use it.^";
RTP_UNSET:
    print "Although ", (the) par1, " is allowed to have the property ~",
        (string) par2, "~, no value was ever given, so it can't now be used.^";
RTP_TOOMANYACTS:
    print "Too many activities are going on at once.^";
RTP_CANTABANDON:
    print "Tried to abandon an activity which wasn't going on.^";
RTP_CANTEND:
    print "Tried to end an activity which wasn't going on.^";
RTP_CANTMOVENOTHING:
    print "You can't move nothing.^";
RTP_CANTREMOVENOTHING:
    print "You can't remove nothing from play.^";
RTP_DIVZERO:
    print "You can't divide by zero.^";
RTP_BADVALUEPROPERTY:
    print "Tried to access property for a value which didn't fit: ",
        "if this were a number it would be ", par1, ".^";
RTP_NOTBACKDROP:
    print "Tried to move ", (the) par1, " (not a backdrop) to ", (the) par2,
        ", which is a region.^";
RTP_TABLE_NOCOL:
    print "Attempt to look up a non-existent column in the table '",
        (PrintTableName) par1, "'.^";
RTP_TABLE_NOCORR:
    print "Attempt to look up a non-existent correspondence in the table '",
        (PrintTableName) par1, "'.^";
RTP_TABLE_NOROW:
    print "Attempt to look up a non-existent row in the table '",
        (PrintTableName) par1, "'.^";
RTP_TABLE_NOENTRY:
    print "Attempt to look up a non-existent entry at column ", par2,
        ", row ", par3, " of the table '", (PrintTableName) par1, "'.^";
RTP_TABLE_NOTABLE:
    print "Attempt to blank out a row from a non-existent table (value ",
        par1, ").^";
RTP_TABLE_NOMOREBLANKS:
    print "Attempt to choose a blank row in a table with none left: table '",
        (PrintTableName) par1, "'.^";
RTP_TABLE_NOROWS:
    print "Attempt to choose a random row in an entirely blank table: table '",
        (PrintTableName) par1, "'.^";
RTP_TABLE_CANTRUNTHROUGH:
    print "Attempt to repeat through a table in a tricky column order: table '",
        (PrintTableName) par1, "'.^";
RTP_TABLE_CANTSORT:
    print "Attempt to sort a table whose ordering must remain fixed: table '",
        (PrintTableName) par1, "'.^";
RTP_TABLE_CANTSAVE:
    print "Attempt to save a table to a file whose data is unstable: table '",
        (PrintTableName) par1, "'.^";
```

```
RTP_TABLE_WONTFIT:
    print "File being read has too many rows or columns to fit into table: table '",
        (PrintTableName) par1, "'.^";
RTP_TABLE_BADFILE:
    print "File being read is not a previously saved table: table '",
        (PrintTableName) par1, "'.^";
RTP_NOTINAROOM:
    print "Attempt to test if the current location is '",
        (the) par1, "', which is not a room or region.^";
RTP_BADTOPIC:
    print "Attempt to see if a snippet of text matches something which
        is not a topic.^";
RTP_ROUTELESS:
    print "Attempt to find route or count steps through an implicit
        relation.^";
RTP_PROPOFNOTHING:
    print "Attempt to use a property of the 'nothing' non-object: property ",
        (PrintPropertyName) par2, "^";
RTP_DECIDEONWRONGKIND:
    print "Attempt to 'decide on V' where V is the wrong kind of object.^";
RTP_DECIDEONNOTHING:
    print "Attempt to 'decide on nothing'.^";
RTP_LOWLEVELERROR:
    print "Low level error.^";
RTP_DONTIGNORETURNSEQUENCE:
    print "Attempt to ignore the turn sequence rules.^";
RTP_SAYINVALIDSNIPPET:
    print "Attempt to say a snippet value which is currently invalid: words ",
        par1, " to ", par2, ".^";
RTP_SPLICEINVALIDSNIPPET:
    print "Attempt to splice a snippet value which is currently invalid: words ",
        par1, " to ", par2, ".^";
RTP_INCLUDEINVALIDSNIPPET:
    print "Attempt to match a snippet value which is currently invalid: words ",
        par1, " to ", par2, ".^";
RTP_LISTWRITERMEMORY:
    print "The list-writer has run out of memory.^";
RTP_CANTREMOVEPLAYER:
    print "Attempt to remove the player from play.^";
RTP_CANTREMOVEDOORS:
    print "Attempt to remove a door from play.^";
RTP_CANTCHANGEOFFSTAGE:
    print "Attempt to change the player to a person off-stage.^";
RTP_MSTACKMEMORY:
    print "The memory stack is exhausted.^";
RTP_TYPECHECK:
    print "Phrase applied to an incompatible kind of value.^";
RTP_FILEIOERROR:
    print "Error handling external file.^";
RTP_HEAPERROR:
    print "Memory allocation proved impossible.^";
RTP_LISTRANGEERROR:
    print "Attempt to use list item which does not exist.^";
```

```
        RTP_REGEXPSYNTAXERROR:
            print "Syntax error in regular expression.^";
        RTP_NOGLULXUNICODE:
            print "This interpreter does not support Unicode.^";
        RTP_BACKDROPONLY:
            print "Only backdrops can be moved to multiple places.^";
        RTP_NOTBACKDROP:
            print "Tried to move ", (the) par1, " (not a thing) to ", (the) par2,
                ", but only things can move around.^";
        RTP_SCENEHASNTSTARTED:
            print "The scene ", (PrintSceneName) par1,
                " hasn't started, so you can't ask when it did.^";
        RTP_SCENEHASNTENDED:
            print "The scene ", (PrintSceneName) par1,
                " hasn't ended, so you can't ask when it did.^";
        RTP_NEGATIVEROOT:
            print "You can't take the square root of a negative number.^";
        RTP_CANTITERATE:
            print "You can't implicitly repeat through the values of this kind: ",
                "a problem arising from a description which started out here - ~",
                (string) par1, "~.^";
        RTP_WRONGASSIGNEDKIND:
            print "Attempt to set a variable to the wrong kind of object: ",
                "you wrote '", (string) par2, "', which sets the value to ", (the) par1,
                " - but that doesn't have the kind '", (string) par3, "'.^";
        }
    print "^";
];
```

§**2. Low-Level Errors.** The following is a residue from the old I6 library, and most of its possible messages can't be seen in I7 use – for instance I7 has no timers or daemons, so error 4 is out of the question. But we retain the routine because the veneer code added by I6 requires it to be present.

```
Constant MAX_TIMERS = 0;
[ RunTimeError n p1 p2;
    #Ifdef DEBUG;
    print "** Library error ", n, " (", p1, ",", p2, ") **^** ";
    switch (n) {
    1:    print "preposition not found (this should not occur)";
    2:    print "Property value not routine or string: ~", (property) p2, "~ of ~", (name) p1,
                "~ (", p1, ")";
    3:    print "Entry in property list not routine or string: ~", (property) p2, "~ list of ~",
                (name) p1, "~ (", p1, ")";
    4:    print "Too many timers/daemons are active simultaneously.
                The limit is the library constant MAX_TIMERS (currently ",
                MAX_TIMERS, ") and should be increased";
    5:    print "Object ~", (name) p1, "~ has no ~time_left~ property";
    7:    print "The object ~", (name) p1, "~ can only be used as a player object if it has
                the ~number~ property";
    8:    print "Attempt to take random entry from an empty table array";
    9:    print p1, " is not a valid direction property number";
    10:   print "The player-object is outside the object tree";
    11:   print "The room ~", (name) p1, "~ has no ~description~ property";
```

```
    12:   print "Tried to set a non-existent pronoun using SetPronoun";
    13:   print "A 'topic' token can only be followed by a preposition";
    default: print "(unexplained)";
    }
    print " **^";
    #Ifnot;
    print "** Library error ", n, " (", p1, ",", p2, ") **^";
    #Endif; ! DEBUG
    RunTimeProblem(RTP_LOWLEVELERROR);
];
```

§**3. Argument Type Checking Failed.** This is called when run-time type checking for the argument of a phrase fails, so that no definition for the phrase can be applied.

```
[ ArgumentTypeFailed file line arg;
    RunTimeProblem(RTP_TYPECHECK, 0, 0, 0, line);
];
```

§**4. Return Type Checking Failed.** Similarly, though in a more restricted set of circumstances. NI can usually prove that the value returned by a phrase matches its supposed kind, but because of the deliberately weak type-checking within the kind of value "object" it will allow a broader kind of object to be returned when the requirement is for a narrower one. In such cases, it checks the result with the following routine. The value `V` is a valid "object", but that means it can be `nothing`, and we must check that it matches a given I7 kind `K` (where `nothing` is not valid).

```
[ CheckKindReturned V K;
    if (V ofclass K) return V;
    if (v == nothing) RunTimeProblem(RTP_DECIDEONNOTHING);
    else RunTimeProblem(RTP_DECIDEONWRONGKIND);
    return V;
];
```

§**5. Whether Provides.** This routine defines the phrase "if O provides P": there are three tests to pass, and if any of the three fail, we return `false`. (The `issue_rtp` flag, causing RTPs to be issued depending on which test fails, is never set when the routine is simply testing the condition.)

Firstly, P has to be a property known to I7. Secondly, there has to be permission either for this individual object to have it, or for its kind to have it, or its kind's kind, and so on. Thirdly, the object has to actually have the property in question at an I6 level – having permission to have it doesn't mean it actually does have.

```
[ WhetherProvides obj either_or p issue_rtp  off i textual a l;
    if (metaclass(obj) ~= Object) rfalse;
    if (p<0) p = ~p;
    if (either_or) {
        if (p < FBNA_PROP_NUMBER) off = attributed_property_offsets-->p;
        else off = valued_property_offsets-->p;
    } else off = valued_property_offsets-->p;
    if (off<0) {
        if (issue_rtp) RunTimeProblem(RTP_BADPROPERTY, obj);
        rfalse;
    }
```

```
    textual = property_metadata-->off; off++;
    if (ScanPropertyMetadata(obj, off)) jump PermissionFound;
    if (obj provides KD_Count) {
        l = obj.KD_Count;
        while (l > 0) {
            a = l*2;
            if (ScanPropertyMetadata(KindHierarchy-->a, off)) jump PermissionFound;
            l = KindHierarchy-->(a+1);
        }
    }
    if (issue_rtp) RunTimeProblem(RTP_UNPROVIDED, obj, textual);
    rfalse;
    .PermissionFound;
        if (either_or) rtrue;
        if (obj provides p) rtrue;
        if (issue_rtp) RunTimeProblem(RTP_UNSET, obj, textual);
        rfalse;
];
[ PrintPropertyName  p  off textual;
    if (p<0) p = ~p;
    off = valued_property_offsets-->p;
    textual = property_metadata-->off;
    print (string) textual;
];
```

§**6. Scan Property Metadata.** The `property_metadata` table is a series of zero-terminated lists of objects (or class objects, representing I7 kinds). Each list corresponds to a single property; the position in the table is called the "offset" for the property. The following searches from a given offset.

```
[ ScanPropertyMetadata obj off i;
    for (i=off: property_metadata-->i >= 0: i++)
        if (obj == property_metadata-->i) rtrue;
    rfalse;
];
```

§**7. Get Either-Or Property.** If `p` represents a property, then `~p` (its bitwise negation) represents its logical negation. The bitwise negation will change the sign bit; since all properties are ordinarily positive, we can detect bitwise negation by looking for negative property numbers. (This could fail on Glulx story files above 1GB in size, but that's about 1000 times the size of the record-sized file created thus far.)

FBNA stands for First Boolean Not to be an Attribute, in the I6 sense. Some either/or (i.e., boolean) properties are stored as I6 attributes, others as I6 properties whose values are always `true` or `false`.

Note that we allow either/or properties to be *read* for any object, regardless of permissions, returning `false` if the object does not have the property. This is so that a description such as "open things" can be applied against any object without run-time errors, even though it is only normally valid for doors and containers.

```
[ GetEitherOrProperty o p;
    if (o == nothing) rfalse;
    if (p<0) p = ~p;
    if (WhetherProvides(o, true, p, false)) {
        if (p<FBNA_PROP_NUMBER) { if (o has p) rtrue; rfalse; }
        if ((o provides p) && (o.p)) rtrue;
    }
    rfalse;
];
```

§**8. Set Either-Or Property.** An attempt to *write* an either/or property which is not provided will, however, always produce a run-time problem.

```
[ SetEitherOrProperty o p negate adj;
    if (p<0) { p = ~p; negate = ~negate; }
    if (adj) {
        (adj)(o);
    } else if (WhetherProvides(o, true, p, true)) {
        if (negate) {
            if (p<FBNA_PROP_NUMBER) give o ~p; else o.p = false;
        } else {
            if (p<FBNA_PROP_NUMBER) give o p; else o.p = true;
        }
    }
];
```

§**9. Value Property.** Some value properties belong to other values (those created in tables), and these are detected by being properties of the special `ValuePropertyHolder` pseudo-object – an I6 object which is not part of the world model, and not a valid I7 "object" value, but which is used in order that properties belonging to values are still I6 property numbers. `ValuePropertyHolder.P` is the table column address for this property; `obj` is then a value for the kind of value created by the table, so it is used as an index into the table column to get the address of the memory location storing the property value.

The `door_to` property, relevant only for doors, is called rather than read: this enables it to be an I6 routine returning the other side of the door from the one which the player is on.

```
[ GProperty K V pr obj;
    if (K == OBJECT_TY) obj = V; else obj = KOV_representatives-->K;
    if (obj == 0) { RunTimeProblem(RTP_PROPOFNOTHING, obj, pr); rfalse; }
    if (obj provides pr) {
        if (K == OBJECT_TY) {
            if (pr == door_to) return obj.pr();
            if (WhetherProvides(V, false, pr, true)) return obj.pr;
            rfalse;
        }
        if (obj ofclass K0_kind)
            WhetherProvides(V, false, pr, true); ! to force a run-time problem
        if ((V < 1) || (V > obj.value_range)) {
            RunTimeProblem(RTP_BADVALUEPROPERTY); return 0; }
        return (obj.pr)-->(V+COL_HSIZE);
    } else {
        if (obj ofclass K0_kind)
            WhetherProvides(V, false, pr, true); ! to force a run-time problem
    }
    rfalse;
];
```

§**10. Write Value Property.** This routine's name must consist of the read-value-property routine's name with the prefix `Write`, as that is how a reference to such a property is converted from an rvalue to an lvalue.

```
[ WriteGProperty K V pr val obj;
    if (K == OBJECT_TY) obj = V; else obj = KOV_representatives-->K;
    if (obj == 0) { RunTimeProblem(RTP_PROPOFNOTHING, obj, pr); rfalse; }
    if (K == OBJECT_TY) {
        if (WhetherProvides(V, false, pr, true)) obj.pr = val;
    } else {
        if ((V < 1) || (V > obj.value_range))
            return RunTimeProblem(RTP_BADVALUEPROPERTY);
        if (obj provides pr) { (obj.pr)-->(V+COL_HSIZE) = val; }
    }
];
```

§**11. Printing Property Names.** Inform doesn't print property names prettily; it more or less prints them only as decimal numbers.

```
[ PROPERTY_TY_Say v;
    print "property ", v;
];
```