Customization Tips

This document provides basic information useful for programmers customizing the Calendar Express interface for iPlanet Calendar Server. It includes the following sections:

- Customization Overview
- XML and XSLT Introduction
- Three Levels of Customization
- Starting the Process: Request for a View
- Calendar Express XML Tags
- Debug Mode
- XML/XSL Links

Customization Overview

The Calendar Express interface for iPlanet Calendar Server is written using XML and XSLT to generate the HTML for each view. This architecture was selected, in part, in order to provide ease of customization. The XML files and the corresponding XSL files are what make up the user interface. The customer can customize the user interface by modifying the existing files, or writing their own XML and XSL files. The customer cannot change what XML tags the iPlanet Calendar Server XSLT processor understands, but they can change which of those tags the XML files contain, and they can change how the XSL files use the XML data. Thus, anyone customizing the user interface will need to understand the XML schema detailed in this document in the section titled “Calendar Express XML Tags,” on page 7.

The HTML that creates the Calendar Express user interface is a frameset with two or more frames in it. For example, the main views (overview, weekview, etc.) consist of a header frame, a toolbar frame and a data frame. While customizing, you can add frames to the frameset, or just modify those frames that already exist as part of the frameset. The way that the frameset and frames are established is explained more fully in this document in the section titled "Calendar Express XML Tags".

For customers who want to write their own client, not based on the Calendar Express XML/XSLT user interface, there is an alternate protocol, WCAP, with which calendar data may be requested in several output formats. For information about WCAP please read the *iPlanet Calendar Server Programmer's Reference*.

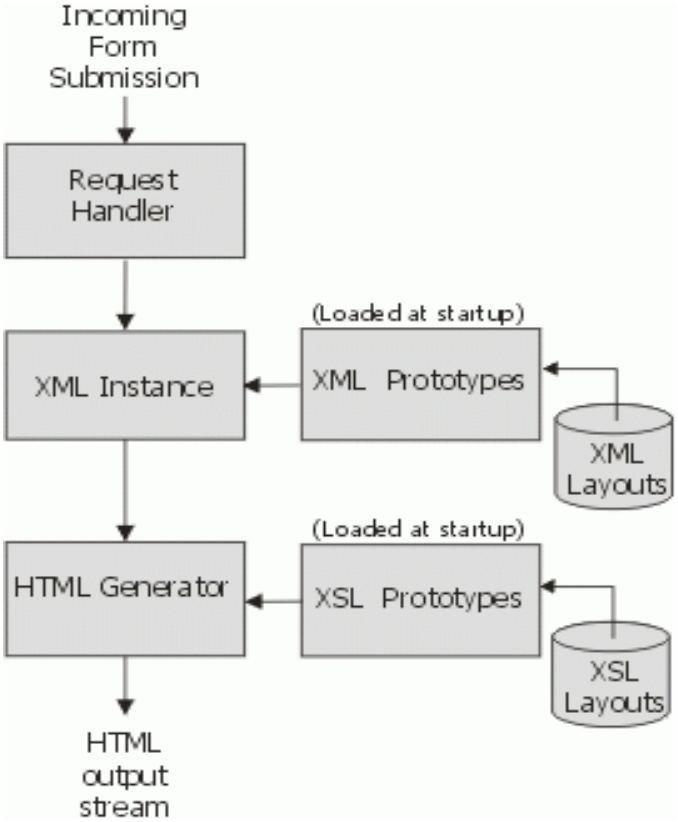
XML and XSLT Introduction

As a general description, XML is not a language, and in fact it doesn't really do anything. It provides a way of structuring information and sending it from one piece of software to another. The definition of XML does not in any way dictate or limit the type of information that it can describe. It simply provides rules for the way information can be categorized and related to other information. Each application using XML defines *tags* or *nodes* (in a tree structure) that are appropriate for its needs. There is no universal definition of tags across applications. Thus, applications that plan on sharing XML data must agree on a common tag set and the relationships between the XML nodes. For example, a banking application might define tags such as `<branch>`, `<customer>` and `<account>`, while an application that dealt with calendaring would want tags called `<calendar>`, `<date>`, and `<user>`. The application also defines what attributes make sense for a given tag and what are reasonable relationships between tags. In the banking example, it makes sense to have the `<customer>` nodes as children of the branch node and the `<account>` nodes as children of the `<customer>` nodes in order to keep track of what customers are at what branches, but for a different application it might make sense to have a `<branch>` node as a child of a `<customer>` node (when listing all the branches with which that customer has accounts).

In a standard XML/XSLT scenario, the XML file that you create is the one used by the XSLT processor. In the case of the iPlanet Calendar Server server, however, the XML files you create are really a prototype, or generalized description of the data needed. The server then preprocesses the XML prototype file for a particular view and creates a specific instance of the XML file depending on the current user, the current time and other appropriate state information.

Figure 1 illustrates this preprocessing of the XML prototype files.

Figure 1 iPlanet Calendar Server XML Processing



This preprocessing allows a very limited set of XML files to serve the needs many users, calendars and dates. For example a particular XML file might request user context information (user name, default calendar, preferences) by using the iPlanet Calendar Server XML tag `<usrctx/>`. The tag does not indicate the name of the user for which it wants information, nor the current calendar being viewed (a user may have multiple calendars). The server uses current state information (provided in the original HTTP command) to expand the simple `<usrctx/>` tag into the completed tag set `<usrctx>...</usrctx>` which now also contains child nodes such as `<user>...</user>` (providing basic user information), `<userprefs>`, `<calendarList>...</calendarList>` and so forth. For the first pass (frameset pass), the server uses the logged-in user in the `usrctx` expansion.

XSLT provides a way of parsing an XML file (or tree) and generating a different type of output. It transforms (hence the *T* in XSLT) the XML into something else, based on one or more XSL files associated with the XML file.

In the case of iPlanet Calendar Server, the transformed output is HTML, which is best displayed on either Netscape Navigator, or Internet Explorer, 4.x and later browsers with JavaScript 1.2. In iPlanet Calendar Server, the XSLT processor is part of the server.The XSLT transformation (HTML Generator in Figure 1) will be applied to the filled-in XML instance, and it will assume all the information is there when writing a header. In the earlier example, that includes the current user's name.

In summary, then, keep in mind that the XML file you write or modify will only contain the generic data required for that view. The server will create an internal copy of that XML file and modify it to contain complete data for that view. The XSL file will then be used to transform the current instance of the XML file into HTML.

NOTE It can be helpful to view the filled out, preprocessed XML file when working with XSL. Instructions for doing so are included in this document in the section titled "Starting the Process: Request for a View," on page 6.

An XSLT transformation requires three things:

- The input XML file that describes the data to be displayed.
- One or more corresponding XSL files that describe how the data is to be displayed.
- An XSLT processor that knows how to parse the XML file, store it in some internal structure, and operate on it to create the resulting transformed output file.

For links to more general information about XML,XSL and XSLT, see the section "XML/XSL Links".

The rest of this document assumes that the reader is familiar with the concepts behind XML and XSLT.

Three Levels of Customization

There are three levels of customization, progressing from the simplest to the most complex:

1. Change the look of an existing view without changing the data contained in the view.

   For example:

   - Adding a new or different banner somewhere on the page.
   - Changing the position or look of elements on the page.
   - Removing some of the information on the page.

   To achieve this level of customization, you will need to change one or more of the XSL files, which tell the server how to format the data from the XML file. You will not need to modify the view's XML file, since the XML file tells the server what data is needed for the view.

2. Change the content of an existing view.

   For example:

   - Adding a list of today's tasks and events to the top of the yearview calendar.
   - Changing the one week lookahead view on the overview page to a two week lookahead.

   For this level of customization, you will need to modify the view's XML file for the necessary additional data. The corresponding XSL file must also be modified to add formatting for the new data.

3. Create a new view to provide calendar data in some different way.

   For example:

   - Providing a quarterly view showing three months at a time, with the days containing a task or event highlighted in red.
   - Providing a comparison view with individual calendars lined up horizontally.

   This level of customization requires creation of both a new XML file that defines the data for the view, and a new XSL file that defines how to format the new view's data. You may take advantage of reusable templates available in other XSL files. Also, you must modify at least one of the other views so that this view can be reached from somewhere in the user interface, by way of a link or a tab.

Starting the Process: Request for a View

A server receives a request for a particular view as part of a URL command line. For example:

```
http://servername:port/command.shtml?view=viewname:subview&
id=12345678&......
```

This command is parsed and the server picks up the `view` name and the optional `subview`. If there is no `subview`, this is said to be a *frameset pass*.

View file names must directly correspond to the value of the `view` parameter. For example, in the sample command above, the value of `view` is `viewname`, thus the XML file name must be `viewname.xml`. Once the view XML file is identified, it is loaded and preprocessed.

Preprocessing expands each of the recognized `<calendar>` tags (see "Calendar Express XML Tags," on page 7) as appropriate according to the following four criteria:

1. Current date/time (time context).
2. Current user (user context).
3. Any additional arguments that were in the original URL line (attributes).
4. The current subview (group).

Each tag is expanded according to its own unique rules. The rules for each tag are detailed in the "Calendar Express XML Tags" section that follows. The preprocessed and expanded XML file is then transformed into formatted HTML output by the XSLT processor as specified by the `viewname.xsl` file.

The current time context only ever refers to one day, defined as no more than 24 hours, from midnight of the day to one second before midnight of the next day. (It can be limited to a smaller period.).

Component information, that is, event and task information, is only ever provided for the current time context. When collecting information for longer time periods, such as a week, you are really collecting information for a set of days. Tags are provided to make this convenient for obvious sets of days, such as weeks, and months. It is more complicated, but possible, to request data one day at a time.

NOTE Keep in mind there may be multiple ways to request the data that you need.

Calendar Express XML Tags

This section lists the iPlanet Calendar Server XML tags. iPlanet Calendar Server's UI generator code recognizes only these tags. You may not create your own tags. Any unrecognized tags will be treated as data and passed through to the file unmodified.

Special Framework Tags

Four of the iPlanet Calendar Server XML tags are special tags that you use to provide a framework for the other tags:

- <calendar>

  The `<calendar>` tag starts each of the view XML files. This tag sets up general calendaring information. All of the XSL files start processing nodes that live below this node. If it is not present, no processing will happen. This is the only tag that checks the frameset pass.

  For a frameset pass, that is, when an entire view, such as the overview, has been requested, global information is set up under the `<calendar>` node. Then, appropriate subview (group) commands are set up which are passed back to the server. One command is set up for each of the `<group>` nodes that exist under the calendar tag. Each subview command is added as a new frame text node.

  For a non-frameset pass, The same general information is set up including the current view information. In addition if there are any errors those are added to the file under the `<calendar>` node.

- <group>

  Use `<group>` tags within the `<calendar></calendar>` tag set to indicate all the subviews that should get processed for this view.

  - In the frameset pass, a command is set up for each group using the current view name plus the subview corresponding to the name of the group. For example:

    ```
    http://calendarservername/command.shtml?view=viewname:
       groupname&id=12345678&....
    ```

    The command also maintains any other arguments that may have existed in the original command except the username and password. Nothing below the `<group>` tags will get processed.

- On a non-frameset pass, only that group matching the name of the subview given in the original command will be processed.The only exception is if the subview being processed is *main* and we are creating a printable version of the view.

- <usrctx>

  The `<usrctx>` tag can exist anywhere in the XML file, but it is customarily placed somewhere after the `<calendar>` tag, and before the `<group>` tags. It contains information specific to this particular user. This information will not change at any point during the view. Usually only one `<usrctx>` tag exists in any given view file.

- <timectx>

  The `<timectx>` tag must occur at least once after the `<calendar>` tag and before any other tags, with the exception of the `<usrctx>` tag. In the course of processing this tag, the server sets up internal time state information. Other tags may be dependent on this information. That is why it must be inserted before these dependent tags. Since the `<usrctx>` tag is not dependent on any time state information, its processing is not effected by its placement relative to the `<timectx>` tag.

XML Tags Recognized by Calendar Express

Table 0-1 lists the recognized tags, gives a short description of each, lists otherat.

Table 0-1 XML Tags Recognized by Calendar Express

XML Tag	Description	Changes Server State	Expanded by Server	Attributes * = required	Arguments Read from URL Line

`<button/>`	Defines the action attributed to a button *onclick.*	No	Yes	`name*`
				`type*`
				`target*`	

`<caldata/>`	Calendar components such as events and todos.		Yes		`e_newCalCalID`
					`newCalCalID`
					`calendarID`

`<calendar/>`	Sets up general calendar framework and information.	Yes	Yes	`tab` `viewname`	

Table 0-1 XML Tags Recognized by Calendar Express

XML Tag	Description	Changes Server State	Expanded by Server	Attributes * = required	Arguments Read from URL Line
`<calgroup/>`	Provides information about calendars within a specified group, or by default, all subscribed calendars.	No	Yes	`name*`, `addr*`, `editCommand`, `unsubscribeCommand`, `viewCommand`	`group` `groupEditor` `isAllGroup`
`<command/>`	Defines a command to be executed.	No	Yes	`type*`, `name*`, `dialog`, `prevView`, `date`, `width`, `height`, `additionalAttributes`, `dontSetTab`	
`<componentlist/>`	Contains all component data for the time context. With the includeAvailability attribute, it also expands to include the Free/Busy time blocks for all specified calendars.	No	Yes	`sort`, `minor`, `defaultStartHour`, `defaultEndHour`, `includeAvailability`, `showbusylist`	`date`
`<config/>`	Used for grouping.	No	No		
`<datactx/>`	Contains the current (or default) `calid` or `groupid`.		Yes		`calid` `group`
`<day/>`	Used for grouping.	No	No		
`<disable/>`	Specifies a view or subview that should not be displayed. For customization. Not used in the shipped UI.	Yes	Yes	`view*`, `tab`	
`<errorslist/>`	Provides errors for current view.	No	Yes		`calid` `group` `option`

Table 0-1 XML Tags Recognized by Calendar Express

XML Tag	Description	Changes Server State	Expanded by Server	Attributes * = required	Arguments Read from URL Line
`<eventdata/>`	Contains information about the event specified by the command line (or default) `uid`.		Yes	`defaultStartHour, defaultEndHour, include Availability`	`date uid rid`
`<exportdata/>`	Provides a list of exportable calendars.	No	Yes	`calid`	`e_ExportCalid _#`
`<formdata/>`	Sets up a formdata element. Information placed in form comes from both the command line and is state dependent.	No	Yes	`type*`	`group`
`<group/>`	Sets up control of processing for tags within the group.	No	No	`type*`	`group`
`<invitationslist/>`	Finds all outstanding invitations for specified user.	No	Yes	`Event, Invitation`	`calid group option`
`<minical/>`	Contains information about the time context's current month, along with commands that vary depending on the attributes used.	No	Yes	`jump_to`	
`<monthcal/>`	Contains information about the time context's current month, along with event and task information for each day in the month.	No	Yea	`showbusylist`	`date`
`<optionsdata/>`	Contains error information. Only exists if command line sets `error` to a value.	No	Yes		`error`

Table 0-1 XML Tags Recognized by Calendar Express

XML Tag	Description	Changes Server State	Expanded by Server	Attributes * = required	Arguments Read from URL Line
`<panel/>`	Delimiter for setting context scope. Groups tags within its boundaries by pushing the internal state onto a stack.	Yes	No		
`<pref_group/>`	Groups and controls access to separate preferences.	No	No		
`<tab/>`	Child of tabs tag set. Adds an attribute to a tabs set.	No	Yes	`view`	
`<tabs/>`	Creates commands to load each tab tag within its boundaries.	No	Yes	`count*, explicit`	
`<taskdata/>`	Contains information about the task specified by the command line `uid` (or default).		Yes		`data uid rid`
`<tasklist/>`	Contains information about all incomplete tasks in chronological order.		Yes	`count`	`date`
`<timectx/>`	Sets a time context state for the server and contains information about the current time context.	Yes	Yes	`add, range`	`date`
`<timezone/>`	Passed through to XSLT without processing.	No	No		
`<userpref/>`	Sets a default user value for the specified user preference; or used to turn on certain debug flags.	Yes	No	`name*, default_value*`	
`<userprefs/>`	Groups multiple userpref tags together.	No	No		
`<usrctx/>`	Contains information about the current user.	No	Yes		

Table 0-1 XML Tags Recognized by Calendar Express

XML Tag	Description	Changes Server State	Expanded by Server	Attributes * = required	Arguments Read from URL Line
`<weekcal/>`	Contains information about the time context's current week for the specified user. Optionally includes component information for each weekday listed. When the optional attribute `format` is set to `optimize`, the populated XML produced for the XSL is optimized for faster rendering.		Yes	`option`, `format`, `showbusylist`	`date`, `optimize`
`<yearcal/>`	Builds up series of minical children. Contains information about the time context's current year for the specified user.		Yes		`date`

Tag Details

<button/>

Use this tag to produce the action that will be used for this button's `onclick`.

The attributes tell the server what type of command to set up:

- *name*, which is used for identification and can be any text.
- *type*, which indicates the type of command the server should create.
- *target*, which must either be set to the value "main", or it will be assumed to be a URL.

  If it is a URL location, the command is set up to point to that target directly if it begins with "http", or to the indicated file in the server's standard http location.

If the target is “main”, the command will be set up as a text node of the original button. It will have the form:

```
javascript:x('<type>');
```

where <type> is replaced by whatever the type attribute value was set to. This is a function set up in the javascript.xsl.

This tag is used in many of the calendar XML files, such as overview.xml.

The following examples illustrate the types the are understood:

```
<button name="ok" type="store" target="main"/>
<button name="cancel" type="cancel" target="main"/>
<button name="help" type="help" target="main"/>
<button  name="delete_calendar_group" type="delete_calendar_group"
   target="main" />
<button  name="unsubscribe" type="unsubscribe" target="main" />
<button name="ok" type="ok" target="main" />
<button name="find" type="find" target="main" />
<button  name="add_user" type="add_user" target="main" />
<button  name="remove_user" type="remove_user" target="main" />
<button  name="add_owner" type="add_owner" target="main" />
<button  name="remove_owner" type="remove_owner" target="main" />
<button name="delete" type="delete" target="main"/>
<button name="add_cal" type="add_cal" target="main" />
<button name="remove_cal" type="remove_cal" target="main" />
<button name="export_add_cal" type="export_add_cal" target="main" />
<button name="export_remove_cal" type="export_remove_cal"
   target="main" />
```

<caldata/>

This tag is expanded to contain all the data pertinent to the calendar indicated in the URL by either (in this order) e_newCalCalID, newCalCalID, or calendarID.

This tag is used by new_cal.xml

Example of tag usage:

```
<caldata/>
```

An example of the expanded tag follows:

```
<caldata e_writeAccessAllowed="true" shortCalID="jdoe"
   newCalCalID="jdoe" viewCommand="http://
   calendarservername/?calid=jdoe&security=1" e_isDefaultCalendar=
   "true" displayName="John Doe" description=
   "Work Calendar for John Doe"
   tzid="" freebusy="0" domain="calendardomainname">
   <OtherOwner name="fred"/>
   <privacyEntry type="otherOwners" name="Other Owners"
      entryIdentifier="5" grant="0" availability="1"
      schedule="1" read="1" delete="1" modify="1"/>
   <privacyEntry type="anyone" name="@" entryIdentifier="2"
      grant="1" availability="1" schedule="1" read="1" delete="0"
      modify="0"/>
</caldata>
```

If no `calid` was specified, a new calendar is created. The output is:

```
<caldata e_isNew=”1” e_writeAccessAllowed=”false”/>
```

<calendar/>

This is the base tag for all iPlanet Calendar Server XML documents. See general notes regarding the “Special Framework Tags,” on page 7.

This tag is used in all calendar view XML files, such as `overview.xml`.

Examples of tag usage:

- <calendar>
- <calendar tab="4">
- <calendar viewname="dayview">
- </calendar>

The following examples show the information added as the XML tag is expanded:

- For a frameset pass:

```
<calendar viewname="overview" staticBaseURL=
   "http://calendarservername.domain.com:port" commandBaseURL=
   "http://calendarservername.domain.com:port top="true"
   (As attributes on the calendar tag)
   <GlobalInfo gID="e2nm0b2qb2mr6s6w9" gCurView="overview"
      gOldView="overview" gDate=""/>
      (As child nodes:The global information that is used by the
      JavaScript functions.)
```

  For each named group, a subview command string is set up:

```
<frame>
   http://calendarservername.domain.com:port/command.shtml?
   view=overview:toolbar&amp;id=e2nm0b2qb2mr6s6w9&amp;date=&
   amp;prevView=overview&amp;view=overview&amp;
   id=e2nm0b2qb2mr6s6w9&amp;group=&amp;security=1

</frame>

<frame>
   http://calendarservername.domain.com:port/command.shtml?
   view=overview:main&amp;id=e2nm0b2qb2mr6s6w9&amp;date=&amp;
   prevView=overview&amp;view=overview&amp;id=e2nm0b2qb2mr6s6w9&
   amp;group=&amp;security=1

</frame>
```

- For a subview pass, the information added will be: (As attributes on the calendar tag)

  ```
  staticBaseURL="http://calendarservername.domain.com:port"
  commandBaseURL="http://calendarservername.domain.com:port"
  view="toolbar" tab="1">
  ```

  Note that in this case, the `top` attribute is missing and the `commandBaseURL` is different.

<calgroup/>

The <calgroup> tag is expanded to provide calendar information about a specified group of calendars, or as a default, all subscribed calendars. If the argument `isAllGroup=true` is part of the URL command line, then it takes precedence and all subscribed calendars will be included. If `isAllGroup` is not present and the argument `group` is present, the server attempts to find that group within the user's definitions. If the group is found, then the `<calgroup>` node will be filled with the calendars making up that group. If the server can't find the group, the server will default to all calendars.

When filling out this tag for a specific group, the server includes an `editCommand` and a `viewCommand`. But if filling it out for all calendars, no edit or view commands will be included. The URL command line argument `groupEditor` indicates which view to set in the `editCommand`.

Examples of tag usage:

```
<calgroup groupEditor="new_group"/>
```

```
<calgroup groupEditor="new_group" calendarEditor="new_cal"/>
```

An example of the tag expanded:

```
<calgroup groupEditor="new_group"
   editCommand="http://calendarservername:port/command.shtml?
      view=new_group&id=b8te8vx95ut9o3h5s&amp;date=20010207T161409
      &amp;group_index=0&amp;tab=1&amp;prevView=overview&amp;
      group=allgroups&amp;security=1"
   viewCommand="javascript:parent.jmain.newViewGroupCommand
      ('groupview','','true','allgroups')"
   name="allgroups" isGroup="true">

   <groupcal name="John Doe" addr="jdoe"
      editCommand="http:&#47;&#47;calendarservername:port&#38;
         newCalCalID=jdoe&#38;tzid=&#38;e_ACL=@@o^a^r^g;@@o^c^
         wdeic^g;@^a^sf^g&#38;tab=1&#38;prevView=overview&#38;
         group=allgroups&#38;security=1"
      unsubscribeCommand="http://calendarservername:port/
         command.shtml?view=calendar_groupview&security=1&amp;
         id=b8te8vx95ut9o3h5s&amp;editCommand=unsubscribe&amp;
         calID=jdoe"
      viewCommand="javascript:parent.jmain.newViewCommand
         ('overview','','true','calid=jdoe')"/>
</calgroup>
```

<command/>

This tag is used to establish an appropriate command for execution. Each command must have a defining type. The options are listed below. The <command> tag and its attributes are left in place for referencing, but the actual command is added as a text node of this command node. In some cases it will be a JavaScript command, otherwise it is a URL.

For convenience, these examples are divided by `type`:

- `type="cmd_dialog"`

  A `cmd_dialog` will set up a `javascript:parent.newPopupCommand()`. The dialog indicates which view to use for the popup. The rest of the arguments are established in the server, based on the dialog chosen and the current time context. Other attributes may only have meaning for the particular view, for example:

  - `width` and `height` will override default width and height settings.
  - `additionalAttributes` will be passed along directly to the `newPopupCommand`, which in turn adds them blindly in to the command string being built up.
  - `dontSetTab`

- `date=from_form` or `date=from_user` specifies where the date should be set from. `from_form` means get the date from the form in the same window. `from_user` (default behavior when not set) means get the date from the current time context.

Examples:

- `<command name="new_event" type="cmd_dialog" dialog="new_event"/>`
- `<command name="jump_to" type="cmd_dialog" dialog="jump_to" date="from_form" width="300" height="350"/>`
- `<command name="new_group" type="cmd_dialog" dialog="new_group" additionalAttributes="e_isNew=1"/>`
- `<command name="new_event" type="cmd_dialog" dialog="new_event" prevView="dayview" additionalAttributes="roundtime=1"/>`
- `<command name="recurrence" type="cmd_dialog" dialog="task_recurrence" date="from_form" dontSetTab=""/>`

- `type="cmd_event_date"`

  A `cmd_event_date` is a way of modifying a form's local date information relative to the `dtStart` of the current event.

  This is currently used in the availability popup for new events. The local date can be reset on the event. The `duration` attribute is required for this command type.

  ```
  <command name="back" type="cmd_event_date" dur="-P1DT"/>
  ```

- `type="cmd_view"`

  A `cmd_view` maintains the current time context, but changes the view as specified. It does this by creating a `newViewCommand` JavaScript function.

  If the `nodata` attribute is set then don't set the indicated value in the command line. Currently this only works for `calid`.

  The following two commands create the two JavaScript functions, respectively:

  ```
  <command name="calendar" type="cmd_view" view="overview"/>

  <command name="calendars" type="cmd_view" view="calendars"
     nodata="calid" />

  javascript:parent.jmain.newViewCommand('overview:main',
     '20010112T105051','true')
  ```

```
javascript:parent.jmain.newViewCommand('calendars',
   '20010112T105042','true')
```

- `type="cmd_url"`

  The `cmd_url` generates a url and sends it in a JavaScript call that opens a new window. The target indicates which file to open. If the target is not absolute (starting with "http://") then set it up to pick up out of the local `hostname:port/language_dir`.

  The command below generates the JavaScript call which follows it:

```
<command name="help" type="cmd_url" dialog="help"
   target="calhelp5.htm"/>
```

```
javascript:var newWindow=window.open('http://calendarservername
   .domain.com:port/en/calhelp2.htm');
```

- `type="cmd_action"`

  Currently `cmd_action` is only used for logging out and is in fact hardcoded for that. It generates the command indicated below.

```
<command name="logout" type="cmd_action" action="logout"
   option="$host_address"/>
```

  The following JavaScript call results:

```
javascript:parent.jmain.newViewCommand('logout','','true')
```

- `type="cmd_window"`

  The `cmd_window` is used to create a JavaScript function that opens a new window (which is different than a dialog in that it has "chrome" meaning it has a control panel and controls). Three important attributes are:

  - `dialog` indicates the view to put in the new window (not optional).
  - `width` and `height` can be used to override default sizing (optional).
  - `date=from_form` or `date=from_user` specifies where the date should be set from. `from_form` means get the date from the form in the same window. `from_user` (default behavior when not set) means get the date from the current time context. (optional)

  The command below generates the JavaScript call that follows it:

```
<command name="print" type="cmd_window" dialog="print"
   width="800" height="600"/>
```

```
javascript:parent.newPopupCommand('overview:print','','&amp;
   tab=1','extraargs','979325442','600','800','true')
```

- `type="cmd_date"`

  A `cmd_date` is used to change the date for the current view by modifying it using the `dur` string relative to the current time context. It does this by creating a `newViewCommand` JavaScript function with the view maintained and the date new.

  This command:

  ```
  <command name="back" type="cmd_date" dur="-P1DT"/>
  ```

  generates:

  ```
  javascript:parent.jmain.newViewCommand('overview:main',
     '20010111T000000','true')
  ```

- `type="cmd_subdialog"`

  A `cmd_subdialog` is handled in much the same way as the `cmd_dialog`.

  ```
  <command name="add_people" type="cmd_subdialog"
     dialog="add_people" width="400" height="200"/>
  ```

<componentlist/>

This tag expands to show all the component (task and event) data pertinent to the day specified in the time context. If the `includeavailability` option is set, the freebusy status for all the included calendars for that day is also included in the expansion. It can request the data in several formats. The output format is the same as that described by the `<eventdata>` and `<taskdata>` tags.

The following examples show various permutations in the use of this tag:

```
<componentlist/>

<componentlist sort="group" defaultStartHour="8"
   defaultEndHour="18"/>

<componentlist sort="group" defaultStartHour="8"
   defaultEndHour="18" includeAvailability="" option="minor"/>

<componentlist sort="overlap"/>

<componentlist sort="overlap" option="minor"/>
```

Table 2 explains the attributes for this tag.

Table 2 Attributes for <componentlist>

attribute	Purpose
`option=minor`	Optional attribute. Tells the processor not to include overdue tasks. Used for displaying small views.

Table 2 Attributes for <componentlist>

attribute	Purpose
`defaultStartHour`	Used to override the user's setting for start of day. Only used when attribute `sort` is set to the value "`group`"
`defaultEndHour`	Used to override the user's setting for end of day. Only used when attribute `sort` is set to the value "`group`".
includeAvailability	Sets freebusy information as an outer layer around the component group information. Used in the comparison view. Only used when attribute `sort` is set to the value "`group`".
showbusylist	If this tag is included, the populated XML contains freebusy block information for calendars with availability access but not read access.
sort={time, overlap, group}	Default is "`time`" and need not be explicitly specified. Determines how the data will be listed.

Showbusylist

The value of this attribute is always null (showbusylist=""). The presence of this attribute causes the populated XML to contain freebusy block information for calendars with availability access but not read access. The freebusy information is interspersed with the usual component information, but has a different format. The following example shows the format of such freebusy information in the expanded XML:

```
<Busy e_Caleid="jdoe" e_Calid_encoded="jdoe">

     <StartTime iso="20020328T120000" year="2002" month="03"
date="28"
hour="12" minute="00" seconds="00" dow="5" weeknum="13"/>

     EndTime iso="20020328T130000" year="2002" month="03 date="28"
hour="13" minute="00" seconds="00" dow="5" weeknum="13"/>

</Busy>
```

Sort Types

The three sort types cause the listed components to be ordered differently:

- Time Order

  The components are listed as individual children of the `<componentlist>` tag, but in this order:

  - Overdue incomplete tasks.

- Today's all day events.
- Chronological listing of tasks and events due today.
- Today's tasks with no time assigned to them.
- Upcoming incomplete tasks.

- Overlap Order

  Components are listed in the same order as with the "`time`" setting, but components of like type, or those sharing an overlapping time block, are listed as children of a specific component group. For example, all overdue tasks are children of a component group of type "`overdue_tasks`", which is itself a child of `<componentlist>`. Components that do no overlap still exist as children of a component group of type "`overlap`"; they are just the only child of that group. This type of <componentlist> is used in the `overview` view.

  The following is an example of the processed XML for overlap order:

```
<componentlist sort="overlap">

<ComponentGroup ComponentGroupType="overdue_tasks">
   <Task>... task data goes here...</Task>
</ComponentGroup>

<ComponentGroup ComponentGroupType="allday_events">
   <Event> ...event data goes here ...</Event>
</ComponentGroup>

<ComponentGroup ComponentGroupType="overlap">
   <StartTime iso="20010116T103000" year="2001" month="01"
date="16"
      hour="10" minute="30" seconds="00" dow="3" weeknum="3"/>
   <EndTime iso="20010116T113000" year="2001" month="01"
date="16"
      hour="11" minute="30" seconds="00" dow="3" weeknum="3"/>
   <Event ...event data...</Event>
</ComponentGroup>

<ComponentGroup ComponentGroupType="overlap">
   <StartTime iso="20010116T153000" year="2001" month="01"
date="16"
      hour="15" minute="30" seconds="00" dow="3" weeknum="3"/>
   <EndTime iso="20010116T163000" year="2001" month="01"
date="16"
      hour="16" minute="30" seconds="00" dow="3" weeknum="3"/>
   <ComponentGroup ComponentGroupType="overlap">
```

```
        <Event>....data ...</Event>
        <Task> ...data...</Task>
      </ComponentGroup>
   </ComponentGroup>

   </componentlist>
```

- Group Order

  This mode groups the component data into consistent blocks of time, such as one hour increments. Overdue and all day tasks precede the time blocks. Tasks with no time associated with them follow the time blocks. In addition, there is a <timeblockset>, which indicate the first and last time blocks that have data. This helps make the XSLT processing of the blocks more efficient. When a component group spans multiple time blocks, it is indicated in the value of "numIntervals". The dayview.xml and groupview.xml (comparison view) use this `sort` ordering.

  The following examples demonstrate "`group`" ordering:

```
<ComponentGroup ComponentGroupType="group" numIntervals="1">
   <StartTime iso="20010116T090000" year="2001" month="01" date="16"
      hour="09" minute="00" seconds="00" dow="3" weeknum="3"/>

   <EndTime iso="20010116T100000" year="2001" month="01" date="16"
      hour="10" minute="00" seconds="00" dow="3" weeknum="3"/>
</ComponentGroup>

....

<ComponentGroup ComponentGroupType="group" numIntervals="2">
   <ComponentGroup ComponentGroupType="group"
      <StartTime iso="20010116T103000" year="2001" month="01"
         date="16" hour="10" minute="30" seconds="00" dow="3"
         weeknum="3"/>
      <EndTime iso="20010116T113000" year="2001" month="01"
         date="16" hour="11" minute="30" seconds="00" dow="3"
         weeknum="3"/>
      <Event> ...data...</Event>
   </ComponentGroup>
   <StartTime iso="20010116T100000" year="2001" month="01" date="16"
      hour="10" minute="00" seconds="00" dow="3" weeknum="3"/>
   <EndTime iso="20010116T120000" year="2001" month="01" date="16"
      hour="12" minute="00" seconds="00" dow="3" weeknum="3"/>
</ComponentGroup>
```

```
<ComponentGroup ComponentGroupType="group" numIntervals="0">
   <StartTime iso="20010116T110000" year="2001" month="01" date="16"
      hour="11" minute="00" seconds="00" dow="3" weeknum="3"/>
   <EndTime iso="20010116T120000" year="2001" month="01" date="16"
      hour="12" minute="00" seconds="00" dow="3" weeknum="3"/>
</ComponentGroup>

....

<timeblockset>
   <timeblock number="9"/>
   <timeblock number="10"/>
   <timeblock number="11"/>
   <timeblock number="12"/>
   <timeblock number="13"/>
   <timeblock number="14"/>
   <timeblock number="15"/>
   <timeblock number="16"/>
   <timeblock number="17"/>
   <timeblock number="18"/>
   <timeblock number="19"/>
</timeblockset>
```

<config/>

This tag is not expanded. It provides a convenient way to group configuration choices. This tag is used in `ui_config.xml`, `simple_config.xml`, and the `nogroup_config.xml` files.

For example:

```
<config>
   <!--<userpref name="_DebugMode" default_value="on"/> -->

</config>
```

<datactx/>

This tag causes the current calid or group id (or default calid if none is specified on the argument line) to be added as a child. The `<datactx>` tag is used in the `overview.xml` file.

The `<datactx/>` tag expands to:

```
<datactx>
<Data calid="jdoe"/>
</datactx>
```

<day/>

This tag isn't expanded by the server, but is a useful way to group information. The related XSL file knows to look for particular information within this tagset.The <day> tag is used in the `overview.xml` file.

Examples of tag usage are:

```
<day name=”singleDayView” option=”listview”>
</day>
```

<disable/>

This tag is used to turn off certain features so that they are not available in the user interface. It is specifically for customization and is not used by the shipped product UI. It specifies either an entire view or a particular tab of a view that should not be displayed. When parsing this tag, the view and optional tab is added to a blocking list, which is checked before any particular view processing occurs. The `<disable>` tag is shown in two files: simple_config.xml and nogroup_config.xml.

In the first example below, only the specified tab of the `new_event` view would be disabled. In the second example, the entire view would be disabled.

```
<disable view="new_event" tab="4" />
<disable view="groupview_dialog" />
```

<errorslist/>

This tag is used to report all errors that the server has encountered back to the UI for display. If the URL command line argument `option` is set to `"count”` the server returns only the number of errors for the current calendars, and not the error messages. If this option is not set, the server picks up the user's current calendar list, either from a `calid` or `group` argument, or if neither exists, uses the default calendar, to retrieve any outstanding error information from the database for these calendars.

If the calendars specified do not exist, or there is a permission problem for the calendar, an error will be returned also. Recurring events will not cause multiple instances of the same error. However, all errors stemming from different causes will be reported.

This tag is used in most of the main XML files, such as `overview.xml`. The count-only option is used in the `dayview.xml` file.

The following examples show that a count of the errors may optionally be requested.

```
<errorslist option="count"/>
<errorslist/>
```

<eventdata/>

This tag is used to provide information about a specific event. It causes an event child to be added to the <eventdata> tag. The event detailed is either specified by the uid on the URL command line, or if none is specified, a default is built up. This tag is used by the new_event.xml and reply_event.xml files.

Note that the format of the event data is the same whenever an event is printed. So when an event is one of many in a <componentlist> tag, or anywhere else, it will always have the same format as the following example.

If the prototype XML states:

```
<eventdata defaultStartHour="8" defaultEndHour="16"
   includeAvailability=""/>
```

then the filled out XML looks like:

```
<eventdata>

   <Event>

      e_Summary="LaterMeeting" e_Summary_urlencoded="LaterMeeting"
         e_writeAccessAllowed="true"

      e_deleteAccessAllowed="true" e_Calid="jdoe" e_org_Calid=
         "jdoe" e_uid="3a64c0b2000062b800000000500000fd1"

      e_rid="0" e_Description="a description goes here" e_Location=
         "a location" e_OrganizerEmail="jdoe@domain.com"

      e_Organizer="jdoe" e_Language="en" e_DtStartTZID=
         "timezonename" e_allday="0" e_repeatModifier=""

      e_dtCreated="20010116T214418Z" e_dtModified=
         "20010116T224600Z" e_dtstart="20010116T233000Z"
         e_existingRRule=""

      e_rrule_notset="true" e_rruleChanged="false" e_dtend=
         "20010117T003000Z" e_durhour="01" e_durmin="00"

      e_iSequence="1" partstat="2" rsvp="TRUE" e_attendee_0="jdoe;
         PARTSTAT=2;STATUS=2;RSVP=1;CN=John Doe;X-NSCP-CALID=jdoe"
         AttendeeList="true">

      <StartTime iso="20010116T153000" year="2001" month="01" date=
         "16" hour="15" minute="30" seconds="00" dow="3"
         weeknum="3"/>

      <EndTime iso="20010116T163000" year="2001" month="01" date=
         "16" hour="16" minute="30" seconds="00" dow="3"
         weeknum="3"/>
```

```
      <attendee userid="jdoe" partstat="2" CN="John Doe" RSVP="TRUE"
         X-NSCP-CALID="jdoe"/>

      <edit_command>
         javascript:var x=newWindow=window.open
            ('http://calendarservername:port/command.shtml?view=
            new_event&amp;tab=1&amp;id=2mhs6bhb2meb&amp;date=
            20010116T142225&amp;uid=3a64c0b2000062b80000000500000fd
            1&amp;calid=jdoe&amp;i_tab=1&amp;prevView=new_event:
            main','2mhs6bhb2meb3a64c0b2000062b80000000500000fd1',
            'height=550,width=650','false')
      </edit_command>

   </Event>

</eventdata>
```

Support for Quick Delete Icon

The populated XML includes an attribute to determine whether to show the quick delete icon (the decision is based on whether there is Delete access granted for this calendar). The attribute is `e_deleteAccessAllowed="true"`. There are two delete commands, <delete_command>, that can be invoked with the delete icon, one for quick deletes of repeating events and tasks, and one for quick deletes of non-repeating events and tasks. This is an example of the code invoked for a quick delete for a repeating event or task:

```
<delete_command>
javascript:parent.newPopupCommand('repeating_event_delete','','&amp
;tab=1&amp;calid=jdoe&amp;')
</delete_command>
```

This is an example of the code invoked for a quick delete of non-repeating event or task:

```
<delete_command>
javascript:parent.jmain.delEventTaskCommand('new_event','jdoe',
'jdoe','overview','3ca3714d0')
</delete_command>
```

These javascript command are the same for events, <eventdata>, and tasks, <taskdata>.

<exportdata/>

This tag expands to list exportable calendars. The list is created from the argument `e_ExportCalid_#` in the URL command line (with the numbers starting at 0). Each `exportcal` item has a `calid` attribute set to the calendar name. There is a one-to-one correspondence to the number of `e_ExportCalid_#` argument entries

and the number of `exportcal` entries in the tag expansion. Thus, if the input argument listed three calendars (any subscribed-to calendar that you have the proper permissions for), then there would be the same three calendars in the expanded tag.

For example:

```
e_ExportCalid_0=”jdoe”;e_ExportCalid_1=”jdoe:secondaryname1”;
   e_ExportCalid_2=”jdoe:secondaryname2”
```

causes the tag to expand as follows:

```
<exportdata>
   <exportcal calid=”jdoe”/>
   <exportcal calid=”jdoe:secondaryname1”/>
   <exportcal calid=”jdoe:secondaryname2”/>
</exportdata>
```

<formdata/>

This tag causes a formdata element to be created. This element contains the information required to set up the form, and the context information that the server will use when processing each action. The information varies according to the command URL arguments. For example if a group is specified, then group information will be added as an attribute. This tag is also state dependent. That is, if a previous tag caused an event to be set up, then the event data will also be included. The `type` attribute will be maintained untouched for subsequent sorting during form processing. This tag is used by any view that needs form data, such as `new_task.xml`.

The following examples show the different types of forms available:

```
<formdata type="add_calendar"/>

<formdata type="add_people"/>

<formdata type="new_event"/>

<formdata type="calendars"/>

<formdata type="change_timezone"/>

<formdata type="errors_list"/>

<formdata type="event_recurrence"/>

<formdata type="search_for_cals"/>

<formdata type="groupview_dialog"/>

<formdata type="invitations_list"/>

<formdata type="new_cal"/>
```

```
<formdata type="new_event"/>

<formdata type="new_group"/>

<formdata type="new_task"/>

<formdata type="options"/>

<formdata type="reply_event"/>

<formdata type="search_for_components"/>

<formdata type="task_list"/>

<formdata type="task_recurrence"/>
```

The following is an example of the expanded tag after processing:

```
<formdata type="new_event" id="2mhs6bhb2meb" view="formstuff"
   calid="jdoe" date="20010116T144622"

   prevView="groupview" myaction="" action=
      "http://calendarservername:port/command.shtml" e_Summary=""

   e_Summary_urlencoded="" e_isNewEvent="true"
      e_writeAccessAllowed="true" e_deleteAccessAllowed="true"

   e_Calid="jdoe" e_org_Calid="jdoe" e_rid="0" e_Organizer="jdoe"
      e_DtStartTZID="timezonename"

   e_allday="0" e_repeatModifier="" e_dtstart="20010116T230000Z"
      e_existingRRule="" e_rrule_notset="true"

   e_rruleChanged="false" e_dtend="20010117T000000Z" e_durhour="01"
      e_durmin="00" e_iSequence="0"

   e_attendee_0="jdoe;PARTSTAT=2;STATUS=2;RSVP=1;CN=John Doe;
      X-NSCP-CALID=jdoe"

   partstat="2" AttendeeList="true" e_alarm_on="0" i_tab="1"
      i_tabswitch="0" i_mainview="main" i_subview="main">
```

<group/>

This tag is used to indicate all the subviews that should be processed for this view, and to define what frames are used in the HTML file. This tag is not expanded during processing.

The following is an example of the listing of subviews in a view using the <group> tag:

```
<group name="main">

<group name="main" view="dayview">

</group>
```

<invitationslist/>

This tag causes the server to output the count and a list of all outstanding invitations (those to which the user has not replied). The server picks up the user's current calendar list either from the `calid` or `group` arguments in the URL command line. If neither exists, it uses the default calendar. The server then retrieves all outstanding (unanswered) invitations from the database for these calendars. Each invitation generates an `<Event>` tag, which is expanded and output in the same form used for `<eventdata>`.Optionally, only the count of the outstanding invitations may be output.

This tag is used in the `invitations_list.xml` file and, as the count only, in most main views.

In the example below, both the count and the list of outstanding invitations will be output between the `<invitationslist>` tag delimiters:

```
<invitationslist/>
```

In the example below, only the count will be output:

```
<invitationslist option="count"/>
```

<minical/>

This tag expands to provide information about the month that the current time context is in. The basic month information is the same regardless of the option used, but the commands provided in the expanded tag vary depending on the option. The expanded tag contains the information necessary to create the miniature month calendar residing on the side of the main views. This tag is used in most main views, such as the `overview.xml`.

A `<minical>` tag consists of three JavaScript `newViewCommands`, one for previous month, the next month, and one to go back to "today". It also has four or five <row> children, each one containing seven (one week's worth) <Time> children. These children contain the information for the specific day along with a JavaScript `newViewCommand` to go to the overview for that day.

The following are examples of the <minical> tag:

```
<minical/>
```

```
<minical jump_to=""/>
```

If the "jump_to" option is used, the commands are set up to stay in the "jump_to" view. In addition, there are many more commands set up before the individual day information.

<Time> children have the following format:

```
<Time iso="20010107T000000" year="2001" month="01" date="7"
   hour="00" minute="00" seconds="00" dow="1" weeknum="2">
   <command>javascript:parent.jmain.newViewCommand
      ('overview:main','20010107T000000','true','calid=jdoe')
   </command>
</Time>
```

<monthcal/>

This tag is used by the `monthview.xml` file and provides information about the month the current time context is in. The tag expands with a hierarchy of children, each mapping to a day in the month. The children also include any days in the previous month and the next month that are needed to fill out the start and end weeks. A week is defined by the user's first-day-of-the-week settings choice.

This tag supports the `showbusylist` attribute as described in the `<componentlist/>` tag.

An example of the tag use:

```
<monthcal/>
```

Figure 2 Monthcal

Figure 2, above, shows the month calendar output by this tag. To get this output, data in the expanded tag is structured in this order:

1. monthcal
2. command for previous month
3. command for next month

4. row (one for each week - 5 or 6 of these)
    a. monthDay (one for each day of week)
        I. time element for this day
        II. componentlist information in overlap format
    b. monthDay

    ...etc.
5. row

    ...etc.

<optionsdata/>
This tag is only expanded if there is an argument in the command URL that sets "error" to a value. It is used in the `options.xml` file.

An example of the tag:

```
<optionsdata/>
```

Table 3 shows the attributes set for various values of "error". Note that all attributes are set to a value of "1".

Table 3 Attributes Set by Various Values for "error"

error= Value	Attribute Set
error="1"	sessionExpired
error="2"	noValidcalendar
error="3"	dateRangeError
error="4"	sourceFileError

<panel/>
This tag is not expanded, but its use helps improve XSLT processing.

An example of this tag:

```
<panel name="availability"> ... </panel>
```

<pref_group/>

This tag controls access to a series of separate preferences to be treated as a group. Each preference group has a name attribute to identify it. When processing, if the pref_group name matches a preference the user has chosen (for example, the large font size family), then the individual preferences within that preference group will be processed. Otherwise the server skips over the pref_group without doing anything.

This tag is used in the `default_user_prefs.xml` file.

An example of this tag's usage:

```
<pref_group name="pref_font_size_group_1"> ... </pref_group>
```

<tab/>

This tag can be used to name specific tags within an explicit tabs set. It is used only as a child of a `<tabs>` tag set, and only if the `<tabs>` tag has a `type="explicit"` attribute. Generally, the `<tab>` tags are inserted by the server as children of the `<tabs>` pair. It is possible to define your own individual `<tab>` tags in order to give them a name attribute. The additional information (command to load) will then be added by the server during processing of the `<tabs>` tag. This tag is used by the `task_list.xml` file.

For example:

```
<tab view="new_event"/>
```

<tabs/>

This tag is expanded to create URL commands that will load the individual tabs within a given view. The individual tabs within the tabs set can be named explicitly or referred to according to their position within the `<tabs>` set. The count indicates how many tabs there are. If `type="explicit"` is set, then the tabs to create are picked up as child text nodes of the `<tabs>` parent (and must therefore must exist as such). Using the explicit attribute is a way to give specific names to individual tabs in order. This tag is used in many files such as `task_list.xml`.

The following are examples of the unexpanded tag:

```
<tabs count="6" type="explicit"> ... </tabs>
```

```
<tabs count="3"/>
```

To load two explicit tabs:

```
<tabs count="2" type="explicit">
   <tab view="new_event"/>
   <tab view="new_event"/>
</tabs>
```

The URL command built to load these explicit tabs is:

```
<tab view="new_event">
   javascript:loadtab(1,'http://calendarservername:port/
      command.shtml?view=new_event:tabs&amp;id=2mhs6bhb2meb&amp;
      tab=1&amp;date=20010116T144622&amp;prevView=groupview&amp;
      calid=jdoe&amp;security=1','new_event');
</tab>
```

To load two generic tabs:

```
<tabs count="2"/>
```

The URL command built to load the generic tab is:

```
<tab>
   javascript:loadtab(1,'http://calendarservername:port/
      command.shtml?view=new_task:tabs&amp;id=2mhs6bhb2meb&amp;
      tab=1&amp;date=20010116T150454&amp;prevView=overview:main
      &amp;calid=jdoe&amp;security=1','new_task');
</tab>
```

<taskdata/>

This tag is used to provide information about a specific event. It causes a task child to be added to the `<taskdata>` tag. The task detailed is either the one specified by the `uid` on the URL command line, or a default is built up. This tag is used by `new_task.xml`. The same Quick Delete support exists for this tag as is found for <eventdata> (see "Support for Quick Delete Icon," on page 26).

The following is an example of the tag:

```
<taskdata/>
```

The format of the task data is the same whenever a task is printed. An example of the format of the expanded tag follows:

```
<taskdata>
   <Task e_Summary="DueLaterTodayTask" e_writeAccessAllowed="true"
      e_Calid="jdoe"
      e_uid="3a64c021000031600000000200000fd1" e_repeatModifier=""
      e_rid="0" e_Description="This is a task due later today"
      e_OrganizerEmail="jdoe@domain.com" e_Organizer="jdoe"
      e_Language="en" e_dtCreated="20010116T134153"
      e_dtModified="20010116T134153" e_existingRRule=""
      e_rrule_notset="true" e_rruleChanged="false"
      e_dtdue="20010116T160000" e_notdue="0" e_allday="0"
      e_iSequence="0">
      <DueTime iso="20010116T160000" year="2001" month="01"
         date="16" hour="16" minute="00" seconds="00" dow="3"
         weeknum="3"/>
```

```
      <edit_command>javascript:var newWindow=window.open
         ('http://calendarservername:port/command.shtml?
         view=new_task&amp;id=2mhs6bhb2meb&amp;date=20010116T142225
         &amp;uid=3a64c021000031600000000200000fd1&amp;calid=jdoe
         &amp;i_tab=1&amp;prevView=new_task:main',
         '2mhs6bhb2meb3a64c021000031600000000200000fd1',
         'height=550,width=650','false')
      </edit_command>
   </Task>
</taskdata>
```

NOTE If the task were overdue, there would be an additional attribute on the <Task> of overdue="TRUE".

<tasklist/>

This tag is expanded to provide information about all incomplete tasks in chronological order by due date. In addition, the task count and overdue task count are added as attributes. They are added as children of the tasklist. This tab is used in the task_list.xml file.

The following are examples of the tag:

<tasklist option="count"/>

```
<tasklist/>
```

If the option="count" attribute is used, no task data is listed, and only attributes for the total task count and overdue task count are added.

For example:

```
<tasklist option="count" taskcount="3" overduetaskcount="1"/>
```

<timectx/>

This is one of the special tags described in "Special Framework Tags," on page 7.

This tag is used to set a time state for the server. This tag must exist at least once after the <calendar> tag. Other tags may depend on being able to use the information from this tag. It may exist more than once in an XML view. This tag is used in most view XML files, such as overview.xml.

The following are examples of the tag:

```
<timectx/>
```

```
<timectx add="+P7DT"/>
```

```
<timectx add="-P3MT" range="P6MT"/>

<timectx add="P1D"/>
```

The following example shows the expanded time context:

```
<timectx tzid="America/Los_Angeles">
   <CurrTime iso="20010112T105031" year="2001" month="01" date="12"
      hour="10" minute="50" seconds="31" dow="6" weeknum="2"/>
   <SelectedTime iso="20010112T105031" year="2001" month="01"
      date="12" hour="10" minute="50" seconds="31" dow="6
      weeknum="2"/>
   <StartTime iso="20010112T000000" year="2001" month="01" date="12"
      hour="00" minute="00" seconds="00" dow="6" weeknum="2"/>
   <EndTime iso="20010112T235959" year="2001" month="01" date="12"
      hour="23" minute="59" seconds="59" dow="6" weeknum="2"/>
</timectx>
```

The CurrTime is always the current time picked up when processing the tag. The SelectedTime can be set either with or without the `add` attribute.

With the `add` attribute, the SelectedTime can be changed relative only to the previous time context. Therefore a `<timectx>` tag with an `add` attribute can not be the first instance of the `<timectx>` tag in an XML file. The value of the add attribute follows the standard duration format.

When there is no `add` attribute on the `<timectx>` tag, and if there is a "&date=..." argument on the URL command line, then the date indicated is the SelectedTime. If there is no date indicated, then the SelectedTime is the CurrTime.

The StartTime and EndTime provide the range around the SelectedTime. Currently these are always the midnight previous to one second before midnight of the following day indicated by SelectedTime.

Only one day is specified by the <timectx> tag.

<timezone/>

```
<timezone type="americas" displayName="America/Adak"
tzid="America/Adak" offset="-10:00" daylightOffset="-09:00"/>

<timezone type="asiaPacific" displayName="Pacific/Kiritimati"

tzid="Pacific/Kiritimati" offset="+14:00"/>
```

<userpref/>

This tag is used to preset default user preference values. This tag is not expanded. It is used in `default_user_prefs.xml`. In addition, there are four preferences that can set up debugging runtime values.

Table 4 shows the four user preferences used to set debugging runtime values.

Table 4 Preferences Used to Set Debugging Runtime Values

User Preferences	Debugging Runtime Values
_DebugMode	gDebug_Mode
_TimingMode	gTiming_Mode
_DetailTimingMode	gTimingMode and gDetailTiming_Mode
_SimulateOutputDataMode	gSimulateOutputData_Mode and gSimulateOutputDataMax

The following are examples of the tag use:

```
<userpref name="icsFirstDay" default_value="1" type="number"
   min_value="1" max_value="7"/>

<userpref name="icsTimeZone" default_value="America/New_York" />

<userpref name="ceDefaultAgenda" default_value="" />

<userpref name="ceToolText" default_value="1" />

<userpref name="ceToolImage" default_value="1" />
```

<userprefs/>

This tag is used as a delimiter around `<userpref>` tags. It is not expanded. It facilitates processing of the `<userpref>` tags. This tag is used in the `default_user_prefs.xml` file.

The following is an example of the tag:

```
<userprefs>
```

<usrctx/>

This is one of the special tags described in “Special Framework Tags,” on page 7.

This tag is expanded with information about the current user. The tag is used in most XML files, such as the `overview.xml` file. The following is an example of the tag:

```
<usrctx/>
```

When the tag is processed, an attribute is added to the `<usrctx>` node to indicate the `selectedCalendar` or `selectedGroup` to display, but not both. In addition four other nodes are added.

Table 5 shows the nodes added between the `<usrctx>` tag set.

Table 5 Nodes Added Between the <usrctx> Tag Set.

Nodes	Node Attributes
<user>	name, firstName, lastName, fullName, mail, language For example: <user name="fred" firstName="Fred" lastName="Smith" fullName="Fred Smith" mail="fred.smith@sesta.com" language="ja"
<userprefs>	Attributes as key-value pairs representing the user preferences settings, such as `ceDayHead=8`
<calendarList>	A list of all calendars for this user. List items take the form of a <calID> node for each of the user's subscribed calendars. An example of the <calID> node is: <callID name="jdoe" printableName= "John Doe"
<usergroup>	Contains a command within the tag itself. For example: <usergroup name="test" description="testgroup" editCommand="http://*calendarservername:port*/command.shtml?view=new_group&id=e2nm0b2qb2mr6s6w9&date=&e_originalDisplayName=test&displayName=test&description=testgroup&e_calDisplayName_0=&e_calAddress_0=jdoe:num2&e_tzid=&tab=1&prevView=overview&security=1" viewCommand="javascript:parent.jmain.newViewGroupCommand('groupview','', 'true','test')"/>

<weekcal/>

This tag is used to provide information about the tasks and events for the week that includes the current time context. It is used in the `weekview.xml` file. In addition, for each day in the week, the expanded XML has two javascript strings that can be used as commands for some other action.

If the option attribute occurs on the tag, then no actual component data is added, only the information about the week that includes the current time context is added. If no `option` attribute exists on the tag, then it provides the appropriate component information for each day of the week. This component information is the same as would be added for a `componentlist` tag in group order format with time blocks.

If the `format="optimize"` attribute occurs on the tag, rendering time is reduced dramatically.

This tag supports the `showbusylist` attribute as described in the `<componentlist/>` tag.

Example 1 Option Dates

An example of the expanded XML for option="dates" follows:

```
<weekcal option="dates" iso="20010116T080000Z" year="2001"
   month="01" date="16" hour="08" minute="00" seconds="00" dow="3"
   weeknum="3">

      <weekday iso="20010114T080000Z" year="2001" month="01"
         date="14" hour="08" minute="00" seconds="00" dow="1"
         weeknum="3"/>
      <weekday iso="20010115T080000Z" year="2001" month="01"
         date="15" hour="08" minute="00" seconds="00" dow="2"
         weeknum="3"/>
      <weekday iso="20010116T080000Z" year="2001" month="01"
         date="16" hour="08" minute="00" seconds="00" dow="3"
         weeknum="3"/>
      <weekday iso="20010117T080000Z" year="2001" month="01"
         date="17" hour="08" minute="00" seconds="00" dow="4"
         weeknum="3"/>
      <weekday iso="20010118T080000Z" year="2001" month="01"
         date="18" hour="08" minute="00" seconds="00" dow="5"
         weeknum="3"/>
      <weekday iso="20010119T080000Z" year="2001" month="01"
         date="19" hour="08" minute="00" seconds="00" dow="6"
         weeknum="3"/>
      <weekday iso="20010120T080000Z" year="2001" month="01"
         date="20" hour="08" minute="00" seconds="00" dow="7"
         weeknum="4"/>
      <timeblockset/>
</weekcal>
```

Example 2

The following example is the populated XML for `weekcal` with format="optimize", showbusylist="", and the two javascript strings:

```
<weekcal format="optimize" showbusylist="">
<weekday iso="20020303T000000Z" year="2002"
   month="03" date="03" hour="00" minute="00" seconds="00" dow="1"
   weeknum="10">
```

```
<command>
javascript:parent.jmain.newViewCommand('dayview:main','20020303T000
000','true','calid','jdoe')
</command>
<command>
javascript:parent.newPopupCommand('new_event','fillindate','&amp;ta
b=1&amp;calid=jdoe','ex')
</command>
</weekday>
```

...one of these for each weekday in the week. Saturday and Sunday can be excluded if the option is set. These weekday declarations also include any allday events and notime tasks associated with the given day.

```
     <timeblock>
       <StartTime iso="20020303T090000" year="2002" month="03"
date="03" hour="09" minute="00" seconds="00" dow="1" weeknum="10"/>
       <EndTime iso="20020303T100000" year="2002" month="03"
date="03" hour="10" minute="00" seconds="00" dow="1" weeknum="10"/>
       <weekday numIntervals="1"/>
       <weekday numIntervals="1"/>
       <weekday numIntervals="1"/>
       <weekday numIntervals="1"/>
       <weekday numIntervals="1"/>
       <weekday numIntervals="1"/>
       <weekday numIntervals="1"/>
     </timeblock>
```

...one `timeblock` for each valid timeblock in the week. The earliest timeblock is defined by either the user's start of day or the earliest event or task in any day of the week, whichever is earlier. The last valid timeblock is defined by either the user's end of day or the latest event or task in any day of the week, whichever is later. The `numIntervals` indicates how many table cells this event group needs. If the number is greater than 1 (one), then the event group spans multiple time blocks. If the number is 0 (zero), then the table cell was used in a previous row. Within a weekday, events are listed as `componentGroups`.

```
</weekcal>
```

<yearcal/>

This tag causes a series of minicalendars to be built as children of this node. The difference between the `<minical>` output and the `<yearcal>` output if that there is only one initial command in the yearcal. It is a command to go to the monthview for that minical month. No component data is output for the days in each minical, just the time and date information.

The following is an example of this tag's use:

```
<yearcal/>
```

Figure 5 on page 42 shows the year calendar produced by this tag. Compare Figure 3 and Figure 4 below and note that the year version of the minical and the month version of the minical differ. The month version appears on most of the other calendar views. The year version is produced only with this tag.

Figure 3 Year Minical

Figure 4 Month Minical

Figure 5 Year Calendar Produced by <yearcal> Tag

To produce this year calendar, data in the expanded tag is structured in this order:

6. row of year minicals (3 of these)
    a. year minical (4 of these)
        I. command for current month
        II. week row (one for each week position in month calendar - 5 or 6)

        -Day (one for each day of the week)
        +Time for this context
        +Command for this time context dayview

-Day
...etc. through 7 days

III. week row
...etc. through 5 or 6 weeks

b. year minical
...etc. through 4 months

7. row of year minicals
...etc. through 3 rows

Debug Mode

A runtime option is available to help you debug. This option causes the server to reload all of the XML and XSL on each hit to the server. (The files are normally cached.) It also writes a file to the directory where the server is running (`bin` directory) for each processed XML file. These files contain the exact code that is being displayed on the client.

The format of the name of the post-processed XML file is: _icsDebug_*current user*_*view*_ *subview*.xml. *Subview* is the frame being processed, such as *main*, *toolbar*, or *buttons*. In general, the *main* frame is usually the most data-rich. When the post-processed file name has the same name as the view, then it resulted from the frameset pass.

NOTE The server slows down noticeably in this mode.

For example, when debug mode is on, it creates the following three post-processed files from the overview.xml file:

`bin/_icsDebug_`*`currentuser`*`_overview_main.xml` (generated from the main frame)

`bin/_icsDebug_`*`currentuser`*`_overview_overview.xml` (generated from the frameset pass)

`bin/_icsDebug_`*`currentuser`*`_overview_toolbar.xml` (generated from the toolbar frame)

Turning Debug Mode On

To put the server in debug mode for XML and XSL:

1. Remove the comment markers from this line in the file `ui_config.xml`:

   ```
   <!-- <config="_DebugMode" default_value="on"/> -->
   ```

2. Then, in `ics.conf`, point the `ui.config.file` preference to `ui_config.xml`:

   ```
   ui.config.file = "ui_config.xml"
   ```

Turning Debug Mode Off

To turn off debug mode and return to normal function:

1. Choose one of the following methods for changing the preference back:
   - Turn off the `_DebugMode` user preference by changing it to read: `default_value=”off”`.
   - Turn off the `_DebugMode` user preference by commenting it out.
   - Remove the preference `ui.config.file` from the `ics.conf` file.
   - Change the value of the `ui.config.file` to “”.
2. Restart the server.

Example of Pre- and Post-Processed XML Files

From Pre-processed overview.xml:

```
<panel name="overview-panel">
  <timectx/>
    <panel name="singleDayTimeHeader">
      <timectx/>
      <command name="back" type="cmd_date" dur="-P1DT"/>
```

From _icsDebug-JoeUser_overview_main.xml:

```
<panel name="overview-panel">
     <timectx tzid="America/Los_Angeles">
```

```
      <CurrTime iso="20001128T105624" year="2000" month="11"
       date="28" hour="10" minute="56" seconds="24" dow="3"
       weeknum="48"/>

      <SelectedTime iso="20001128T105624" year="2000" month="11"
       date="28" hour="10" minute="56" seconds="24" dow="3"
       weeknum="48"/>

      <StartTime iso="20001128T000000" year="2000" month="11"
       date="28" hour="00" minute="00" seconds="00" dow="3"
       weeknum="48"/>

      <EndTime iso="20001128T235959" year="2000" month="11"
       date="28" hour="23" minute="59" seconds="59" dow="3"
       weeknum="48"/>

   </timectx>

   <panel name="singleDayTimeHeader">

      <timectx tzid="America/Los_Angeles">

         <CurrTime iso="20001128T105624" year="2000" month="11"
          date="28" hour="10" minute="56" seconds="24" dow="3"
          weeknum="48"/>

         <SelectedTime iso="20001128T105624" year="2000"
          month="11" date="28" hour="10" minute="56" seconds="24"
          dow="3" weeknum="48"/>

         <StartTime iso="20001128T000000" year="2000"
          month="11"date="28" hour="00" minute="00" seconds="00"
          dow="3" weeknum="48"/>

         <EndTime iso="20001128T235959" year="2000" month="11"
          date="28" hour="23" minute="59" seconds="59" dow="3"
          weeknum="48"/>

      </timectx>

      <command name="back" type="cmd_date"
       dur="-P1DT">javascript:parent.jmain.newViewCommand
       ('overview:main','20001127T000000','true')</command>
```

XML/XSL Links

The primary source of information on the internet for XML and XSL is the World Wide Web Consortium found at: `www.w3.org`.

W3 Related links

- General information on XSL is at: `http://www.w3.org/Style/XSL`.
- The XSL Working Draft is at: `http://www.w3.org/TR/xsl`.
- The XSLT Recommendation is at: `http://www.w3.org/TR/xslt`.
- The XPath Recommendation is at: `http://www.w3.org/TR/xpath`.
- The original XSL note is at: `http://www.w3.org/TR/NOTE-XSL`.

Other Links

- An excellent source of information about XSL is the XSL page of Robin Cover's "the XML Cover Pages" found at: http://`www.oasis-open.org/cover`.
- Dave Pawson's XSL FAQ is at: `http://www.dpawson.freeserve.co.uk/xsl/xslfaq.html`.
- Mulberry Technologies hosts the XSL-List, an open forum for discussion of XSL, at: `http://www.mulberrytech.com/xsl/xsl-list`.
- Zvon, a web organization dedicated to free information exchange centered on XML, has several tutorials, and an XSLT reference. Their web URL is: `http://www.zvon.org`.
- "XML for the absolute beginner" is an excellent overview found at: `http://www.javaworld.com/javaworld/jw-04-1999/jw-04-xml.html`. This site also contains valuable links to other XML/XSL resources.

Tutorial Links

Finding a web tutorial on XML/XSL should not be difficult, but here are two sites to get you started:

- Click on the `TUTORIALS` link at: `http://www.zvon.org`.
- Several are offered at: `http://www.w3schools.com`.